**Правительство Российской Федерации**

**Государственное образовательное бюджетное учреждение**

**высшего профессионального образования**

**«Научно-исследовательский университет –**

**Высшая школа экономики»**

**Факультет: МИЭМ**

**Направление: *Компьютерная безопасность***

**Отчет по лабораторной работе №5 (алгоритмы генерации псевдослучайных чисел)**

**по дисциплине**

**«Методы программирования»**

**Выполнил**

**Студент группы СКБ151**

**Михалицын Пётр**

**Москва, 2017.**

Содержание

[Описание проекта 2](#_Toc504685180)

[Спецификация модулей 3](#_Toc504685181)

[Алгоритмы вычисления псевдослучайного числа 6](#_Toc504685182)

[Результаты, полученные после статистического анализа алгоритма Вихрь Мерсенна. 7](#_Toc504685183)

[Коды модулей 8](#_Toc504685184)

[random\_generator.py 8](#_Toc504685185)

# Описание проекта

Проект заключался реализации псевдослучайных числе используя алгоритм «Вихрь Меерсона» и статистического анализа полученных с помощью него выборок

Проект содержит в себе один модуль random\_generator.py

Посмотреть полностью весь проект можно в репозитории http://github.com/lo1ol/random-generator

# Спецификация модулей

Модуль random\_generator.py содержит реализацию класса MersenneTwisterGenerator с тремя методами: конструктор, функция генерации следующего случайного числа и статического метода закалки.

Конструктор имеет один необязательный параметр – зерно для генерации случайных чисел, если он не указан, то генерируется зерно используя текущее время и текущую частоту процессора

class MersenneTwisterGenerator:  
 *"""  
 has one constructor  
 method gen for calculating next pseudo random number  
 and method tempering for tempering.  
 :attribute prev: attribute containt the list of last 624 number  
 """* def \_\_init\_\_(self, seed=None):  
 *"""  
 tke one optional parameter seed for set seed of pseudo random generator.  
 if seed not set, calculate prev using hash of multiplying current time and current processor frequency  
 if seed is set calculate prev using hash of seed firstly complemented by 123456 and on the next steps  
 using prev gotten number* ***:param*** *seed:  
 """*

Метод gen содержит реализацию генерации случайного числа, как результат – на выходе мы получаем следующее случайное число

def gen(self):  
 *"""  
 generate nest pseudo random number using Mersenne Twister Generator with numbers  
 r=31  
 n=624  
 m=397  
 a=0x9908B0DF  
 b=0x9D2C5680  
 c=0xEFC60000* ***:return****: pseudo random number  
 """*

Так же имеется статический метод закалки полученного случайного числа, он вызывается перед тем, как отправить полученное случайное число.

@staticmethod  
def \_tempering(x):  
 *"""  
 take the nu,ber and make tempering* ***:return****: number after tempering  
 """*

Так же в классе присутствует функция hash. Она нужна для того, чтобы привести умножение текущей частоты и текущего времени к целому числу и распределить все возможные перемножения чисел равномерно на множестве целых чисел от 0, до 2\*\*32-1, что является свойством хорошей хеш функции.

def hash(number):  
 *"""  
 Function calculate hash via rs method* ***:param*** *number tke the number for calculating hash:* ***:return****: the hash  
 """*

Более того, в модуле присутствует 3 функции: для подсчета выборочного среднего, получения несмещенной оценки корня выборочной дисперсии и коэф. Вариации

def mean(sample):  
 *"""  
 calculate mean of sample* ***:param*** *sample: sample* ***:return****: mean of sample  
 """*

def deviation(sample):  
 *"""  
 clculte deviation of sample* ***:param*** *sample: sample* ***:return****: deviation of sample  
 """*

def variation\_coeficient(sample):  
 *"""  
 calculate variation coefficient of sample* ***:param*** *sample:* ***:return****:  
 """*

В модуле присутствует вспомогательная функция, которая вычисляет минимальный уровень доверия для критерия хи квадрат используя функцию распределения хи квадрат с заданным количеством степеней свободы.

def level\_trust(df, V):  
 *"""  
 return tha minimal level trust for chi squre test* ***:param*** *df: degree free* ***:param*** *V: gotten number for getting quantile of chi sqr* ***:return****:  
 """*

Так же имеется реализация двух функций для подсчета минимального уровня доверия для критерия хи квадрат: одна разбивает интервал, на котором распределены числа, согласно формуле Стерджесса; вторая же производит тест хи квадрат без разбиения (стандартным методом)

def chi\_sqr\_int(sample, max):  
 *"""  
 make chi square test for sample hypothesising that the sample has distribution of discrete number from 0 to max - 1  
 the calculated level trust gotten via dividing sample in interval whit length = ceil(log2(n)+1)* ***:param*** *sample:* ***:param*** *max:* ***:return****: level of trust  
 """*

def chi\_sqr(sample, max):  
 *"""  
 make standard chi square test without dividing by intervals  
 for sample hypothesising that the sample has distribution of discrete number from 0 to max - 1* ***:param*** *sample:* ***:param*** *max:* ***:return****: level of trust  
 """*

Последний тест высчитывает однородность выборки и так же выводит минимально допустимый уровень доверия для прохождения выборки в тесте на однородность

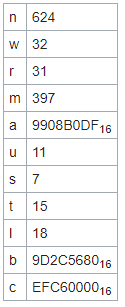
def uniformity(sample, max):  
 *"""  
 calculate uniformity of sample using Student*

*distribution* ***:param*** *sample: sample* ***:param*** *max:* ***:return****: level of trust  
 """*

После всех этих реализаций производится непосредственная генерация 10 выорок для каждой из которых вычисляются вышеприведенные характеристики. По окончанию вычисления всех этих вычислений создается файл results.txt. В который заносятся сами выборки и результаты вычислений. После этого пользователя просят ввести интересующий его уровень доверия. Если для заданного уровня доверия тест на однородность с использованием интервального критерия хвадрат и проверки на однородность оба полученных минимально допустимых уровня доверия оказываются ниже, чем заданный пользователем уровень доверия, то тест для этих выборок считается пройденным, в противном случае выборка не проходит тест с заданным уровнем доверия.

# Алгоритмы вычисления псевдослучайного числа

Перед началом вычисления псевдослучайных чисел задается зерно (вручную или используя произведение текущего времени и текущей частоты процессора и подсчета от полученного числа хеш суммы). После получения числа реализуется стандартный алгоритм вихря Мерсенна с использованием закалки полученных случайных чисел. Про сам алгоритм можно узнать используя ресурс https://ru.wikipedia.org/wiki/Вихрь\_Мерсенна . Числа, которые использовались как параметры данного алгоритма тоже были взяты из данного ресурса



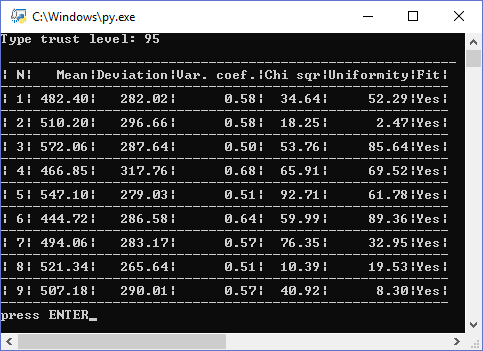
# Результаты, полученные после статистического анализа алгоритма Вихрь Мерсенна.

Как оказалось алгоритм Вихрь Мерсенна является довольно неплохой реализаций генератора псевдослучайных чисел, т.к. при прохождении критерии хи квадрат с уровнем доверия 95%. Примерно 5 выборок из 100 не прошли тест на неги, что является вполне предполагаемым числом выборок, которые не должны были его пройти (100 – 100\*0.95). При остальных уровнях доверия получались примерно те же результаты при тестировании (100 -100\*0.95)

То же самое касается при тестировании на однородность выборки. Что тоже является нормальным явлением при тестировании.

Значения выборочного среднего, несмещенной оценки корня выборочной дисперсии и коэффициента вариации тоже приближались в среднем к теоретически полученным значениям.

Ниже приведен пример полученной таблицы характеристик 10 выборок при заданном уровне доверии 95 и полученным для него вердиктом. Случайная величина моделировалась как дискретная равномерно распределённая случайная величина на отрезке от 0 до 1024



# Коды модулей

## random\_generator.py

from time import time  
from scipy.stats import chi2, t  
from math import log2, ceil  
from psutil import cpu\_freq  
  
"""  
three variances that are used in hash function.  
Global because changes on each calculation of hash  
"""  
  
a\_hash = 63689  
b\_hash = 378551  
res\_hash = 1  
  
  
def hash(number):  
 *"""  
 Function calculate hash via rs method* ***:param*** *number tke the number for calculating hash:* ***:return****: the hash  
 """* global a\_hash, b\_hash, res\_hash  
  
 for i in str(number)[::-1]:  
 res\_hash = res\_hash\*a\_hash + ord(i)  
 a\_hash = (a\_hash\*b\_hash) % 2 \*\* 32  
 res\_hash = res\_hash % 2 \*\* 32  
  
 return res\_hash  
  
  
class MersenneTwisterGenerator:  
 *"""  
 has one constructor  
 method gen for calculating next pseudo random number  
 and method tempering for tempering.  
 :attribute prev: attribute containt the list of last 624 number  
 """* def \_\_init\_\_(self, seed=None):  
 *"""  
 tke one optional parameter seed for set seed of pseudo random generator.  
 if seed not set, calculate prev using hash of multiplying current time and current processor frequency  
 if seed is set calculate prev using hash of seed firstly complemented by 123456 and on the next steps  
 using prev gotten number* ***:param*** *seed:  
 """* if seed == None:  
 self.prev = [hash(time()\*cpu\_freq()[0]) for \_ in range(624)]  
 else:  
 global a\_hash, b\_hash, res\_hash  
 a\_hash = 63689  
 b\_hash = 378551  
 res\_hash = 1  
 self.prev = []  
 prev = (seed+123456)  
 for i in range(624):  
 prev = hash(prev)  
 self.prev.append(prev)  
  
 self.last = 0  
  
 def gen(self):  
 *"""  
 generate nest pseudo random number using Mersenne Twister Generator with numbers  
 r=31  
 n=624  
 m=397  
 a=0x9908B0DF  
 b=0x9D2C5680  
 c=0xEFC60000* ***:return****: pseudo random number  
 """* xk = self.prev[self.last]  
 xk1 = self.prev[(self.last + 1) % 624]  
 xk397 = self.prev[(self.last + 397) % 624]  
 cat\_k\_and\_k1 = ((xk >> 31) << 31) + (xk1 % 2 \*\* 31)  
 if cat\_k\_and\_k1 % 2 == 0:  
 new = xk397 ^ (cat\_k\_and\_k1 >> 1)  
 else:  
 new = xk397 ^ ((cat\_k\_and\_k1 >> 1) ^ 0x9908B0DF)  
 new = self.\_tempering(new)  
  
 self.prev[self.last] = new  
 self.last = (self.last+1) % 624  
  
 return new  
  
 @staticmethod  
 def \_tempering(x):  
 *"""  
 take the nu,ber and make tempering* ***:return****: number after tempering  
 """* x = x ^ (x >> 11)  
 x = x ^ ((x << 7) & 0x9D2C5680)  
 x = x ^ ((x << 15) & 0xEFC60000)  
 x = x ^ (x >> 18)  
 return x  
  
  
def mean(sample):  
 *"""  
 calculate mean of sample* ***:param*** *sample: sample* ***:return****: mean of sample  
 """* total = 0  
 for i in sample:  
 total += i  
 return total / len(sample)  
  
  
def deviation(sample):  
 *"""  
 clculte deviation of sample* ***:param*** *sample: sample* ***:return****: deviation of sample  
 """* sample\_mean = mean(sample)  
 total = 0  
 for i in sample:  
 total += (i-sample\_mean)\*\*2  
  
 return (total/(len(sample)-1))\*\*0.5  
  
  
def variation\_coeficient(sample):  
 *"""  
 calculate variation coefficient of sample* ***:param*** *sample:* ***:return****:  
 """* sample\_mean = mean(sample)  
 sample\_deviation = deviation(sample)  
  
 return sample\_deviation/sample\_mean  
  
  
def level\_trust(df, V):  
 *"""  
 return tha minimal level trust for chi squre test* ***:param*** *df: degree free* ***:param*** *V: gotten number for getting quantile of chi sqr* ***:return****:  
 """* return chi2(df).cdf(V)  
  
  
def chi\_sqr\_int(sample, max):  
 *"""  
 make chi square test for sample hypothesising that the sample has distribution of discrete number from 0 to max - 1  
 the calculated level trust gotten via dividing sample in interval whit length = ceil(log2(n)+1)* ***:param*** *sample:* ***:param*** *max:* ***:return****: level of trust  
 """* total = 0  
 n = len(sample)  
 k = ceil(log2(n)+1)  
 ndiap = ceil(max/k)  
 for i in range(ndiap):  
 if k\*(i+1) < max:  
 total += ((len(list(filter((lambda x: k\*i <= x < k\*(i+1)), sample))) - n\*k/max)\*\*2)/(n\*k/max)  
 else:  
 total += ((len(list(filter((lambda x: k\*i <= x < k\*(i+1)), sample))) - n\*(max-k\*i)/max)\*\*2)/(n\*(max-k\*i)/max)  
  
 return level\_trust(ndiap-1, total)\*100  
  
  
def chi\_sqr(sample, max):  
 *"""  
 make standard chi square test without dividing by intervals  
 for sample hypothesising that the sample has distribution of discrete number from 0 to max - 1* ***:param*** *sample:* ***:param*** *max:* ***:return****: level of trust  
 """* total = 0  
 n = len(sample)  
 for i in range(max):  
 total += ((sample.count(i) - n/max)\*\*2)/(n/max)  
  
 return level\_trust(max-1, total)\*100  
  
  
def uniformity(sample, max):  
 *"""  
 calculate uniformity of sample using Student distribution* ***:param*** *sample: sample* ***:param*** *max:* ***:return****: level of trust  
 """* n = len(sample)  
 m1 = 1/2 \* (max-1)  
 s1 = ((max\*\*2-1)/12)\*\*0.5  
 m2 = mean(sample)  
 s2 = deviation(sample)  
  
 v = (abs(m2-m1)\*n\*\*0.5)/(s1\*\*2+s2\*\*2)\*\*0.5  
 return (t(2\*n-2).cdf(v)\*100 - 50)\*2  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 """  
 make 10 samples with n numbers from 0 to max - 1  
 """  
 samples = []  
 n = 100  
 max = 1024  
 for nsample in range(10):  
 gen = MersenneTwisterGenerator()  
 samples.append([])  
 for i in range(n):  
 samples[-1].append(gen.gen() % max)  
  
 properties = {}  
 """  
 calculate necessary parameters for each sample  
 """  
 for i, sample in enumerate(samples):  
 properties[i + 1] = {"mean": mean(sample),  
 "deviation": deviation(sample),  
 "variation": variation\_coeficient(sample),  
 "chi sqr int": chi\_sqr\_int(sample, max),  
 "chi sqr": chi\_sqr(sample, max),  
 "uniformity": uniformity(sample, max)}  
  
 field = """\  
 Mean: {0[mean]}  
 Deviation: {0[deviation]}  
 Variation coef.: {0[variation]}  
 Chi Sqr trust: {0[chi sqr int]}  
 Uniformity trust: {0[uniformity]}"""  
  
 """  
 make necessary file with samples and their properties  
 """  
 file = open("results.txt", 'w')  
 for i in range(10):  
 for j in range(len(samples[i])):  
 file.write("{:>4}".format(samples[i][j]))  
 if j != n-1:  
 file.write(",")  
 else:  
 file.write("\n")  
 break  
 if j % 20 == 19:  
 file.write("\n")  
  
 file.write(field.format(properties[i+1]) + '\n\n')  
 file.close()  
  
 """  
 check each sample for setting by keyboard level of trust.  
 if sample fits passes all test the sample is recognised, otherwise doesn't   
 after all tests makes the formatted table with results  
 """  
 tl = int(input("Type trust level: "))  
 print("\n", "-" \* 56)  
 print("| N| Mean|Deviation|Var. coef.|Chi sqr|Uniformity|Fit|")  
 print("-"\*56)  
  
 field = """|{0:>2}|{1[mean]:>7.2f}|{1[deviation]:>9.2f}|{1[variation]:>10.2f}|{1[chi sqr int]:>7.2f}|{1[uniformity]:>10.2f}|{2:>3}|"""  
  
 for i in range(1, 10):  
 if properties[i]["chi sqr int"] < tl and properties[i]["uniformity"] < tl:  
 print(field.format(i, properties[i], "Yes"))  
 else:  
 print(field.format(i, properties[i], "No"))  
 print("-" \* 56)  
  
 input("press ENTER")