**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра теории вероятностей и математической статистики

**Крагель Алины Олеговны**

**Моделирование базовой случайной величины**

Отчет по лабораторной работе №1

(«Имитационное и статистическое моделирование»)

Студентки 4 курса 9 группы

**Преподаватель**

*Гайдук Антон Николаевич*

# Теоретическая часть

## Моделирование БСВ

### Линейный конгруэнтный метод

Согласно этому методу, псевдослучайная последовательность реализаций  БСВ определяется по рекуррентным формулам:

где , , ... , – выходная последовательность генератора длины *n*, – начальное значение, *a* ≠ 0 – множитель, *с* – приращение, *M* – модуль.

Период датчика Т.

### Метод Маклорена-Марсальи

Метод основан на комбинировании двух простейших программных датчиков БСВ (например, линейных конгруэнтных).

Пусть – псевдослучайные последовательности, порождаемые независимо работающими датчиками; – результирующая псевдослучайная последовательность реализаций БСВ; – вспомогательная таблица  чисел.

Процесс вычисления включает следующие этапы:

* первоначальное заполнение таблицы :



* случайный выбор из таблицы:



* обновление табличных значений:

.

Данный метод позволяет ослабить зависимость между членами псевдослучайной последовательности и получить сколь угодно большие значения её периода Т при условии, что периоды Т1, Т2 исходных датчиков являются взаимно простыми числами.

## Проверка точности моделирования

### Тест «совпадения моментов»

Пусть в результате -кратного обращения к датчику БСВ получена выборка значений . Известно, что БСВ имеет среднее значение  и дисперсию . Обозначим случайные отклонения выборочных оценок от истинных характеристик  как:

,  (1.1)

где

,  (1.2)

Тест «совпадения моментов» – это программа для ЭВМ, реализующая статистические критерии проверки по выборке А гипотез:

** ** (1.3)

**, ** (1.4)

Тогда решающее правило имеет вид:

принимается **** (1.5)

где – нормировочные множители; – порог критерия.

Если  верна, а >>1 (практически ≥20), то в силу ЦПТ: ~Ν1(0,1) (распределено приближённо по стандартному нормальному закону). С учётом этого из ограничения на вероятность ошибки первого рода:

 (1.6)

находится выражение для порога критериев:

Δ = Ф-1(1 - ), (1.7)

где Ф-1  – квантиль стандартного нормального закона, – заданный уровень значимости.

В лабораторной работе реализована эквивалентная форма решающих правил, связывающей задаваемый пользователем уровень значимости  и вычисляемые по выборке А критические вероятности -значения):

принимается ,  (1.8)

### Тест «ковариация»

Ковариационной функцией случайной последовательности  называется функция целочисленной переменной :

 (1.9)

Если  – независимые, одинаково распределённые по закону R(0,1) случайные величины, то  и  независимы для любого  и следовательно:

 (1.10)

Пусть  – оценка  по выборке , полученной в результате - кратного обращения к исследуемому датчику:

 (1.11)

где 1<t<< – выборочное среднее. Заметим, что - выборочная дисперсия).

Тест «ковариация» позволяет проверить свойство (1.10) (гипотезу ) для последовательности  и описывается следующим решающим правилом:

принимается  (1.12)

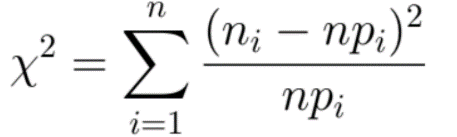
где:   для  Δ – порог, определённый для заданного уровня значимости  по формуле:

Δ = Ф-1(1 - ). (1.13)

В лабораторной работе использована эквивалентная форма правила (1.12):

принимается ,  (1.14)

### Критерий хи-квадрат Пирсона

Отрезок области определения делится на k подотрезков. Оценивается вероятность попадания случайной величины в i-ый подотрезок.

Решающее правило в таком случае имеет вид:

Если   
, иначе нет.

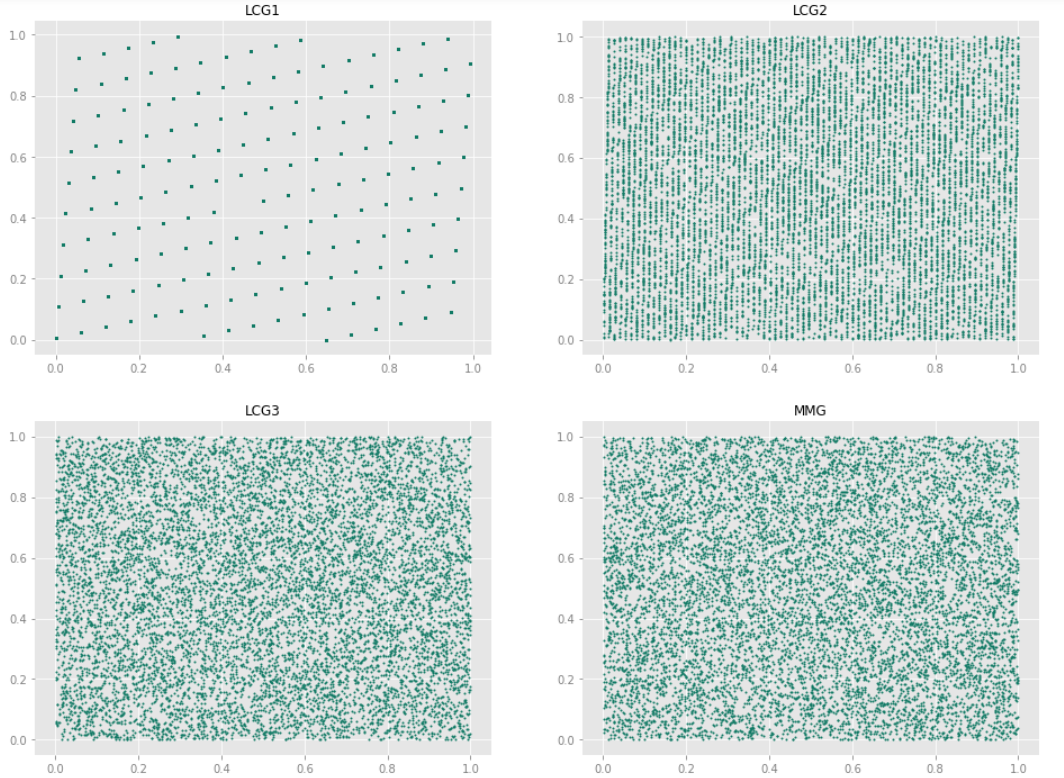
Порог:   
.

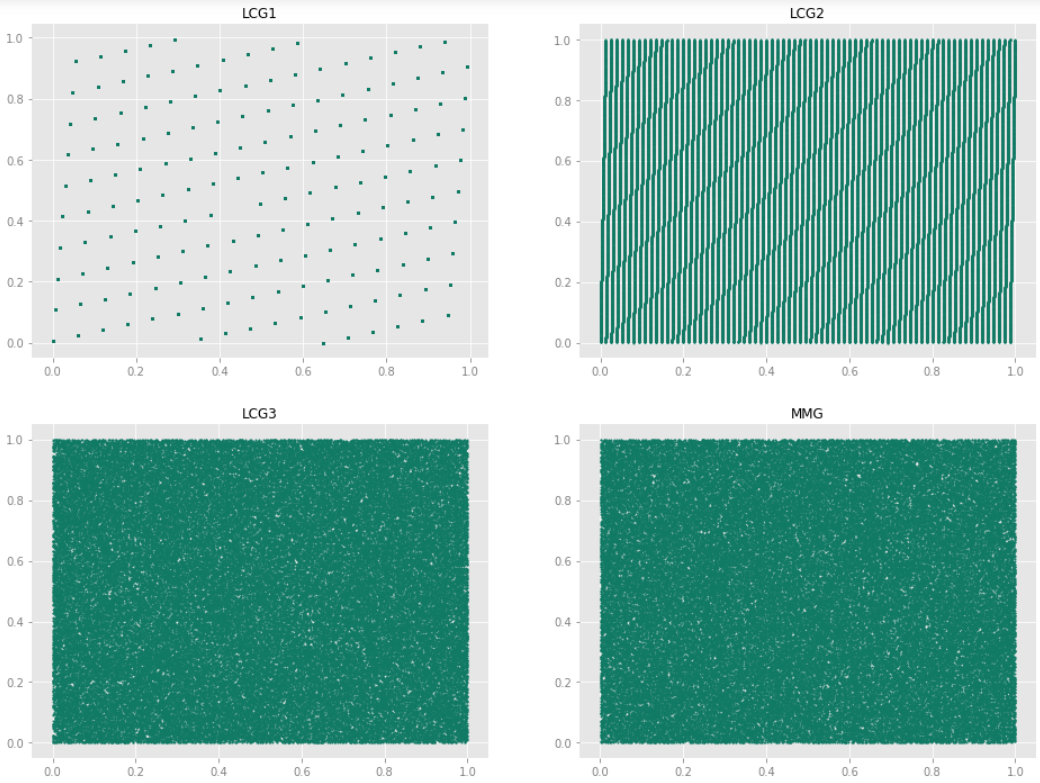
# Результаты экспериментов

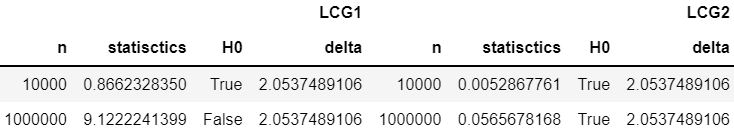
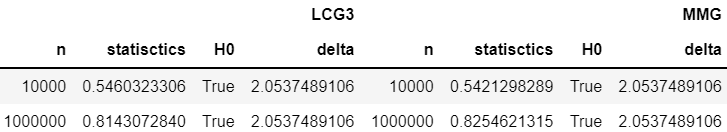
Были протестированы генераторы со следующими параметрами:

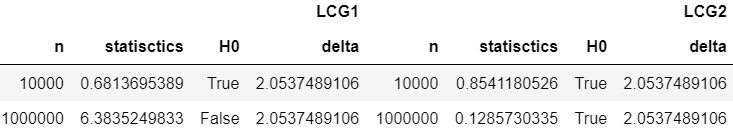
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Генератор** | **M** | | **a** | | **c** | |  |
| LCG1 | 173 | | 174 | | 1 | | 19 |
| LCG2 | 2^16 + 1 | | 75 | | 74 | | 2^8 |
| LCG3 | 2^24 | | 1140671485 | | 12 820 16 | | 2^12 |
| **Генератор** | | **Генератор индексов массива** | | **Генератор чисел из [0,1]** | | **K (длина массива)** | |
| MMG | | LCG3 | | LCG2 | | 100 | |

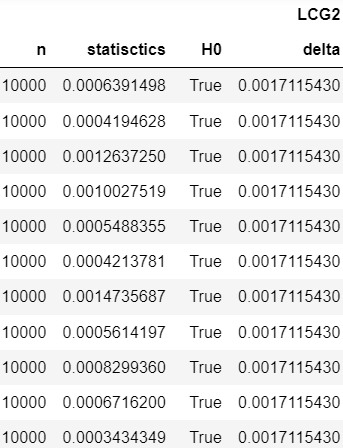
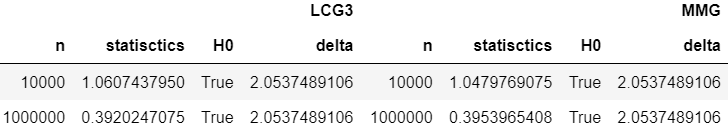
Диаграммы Рассеяния:

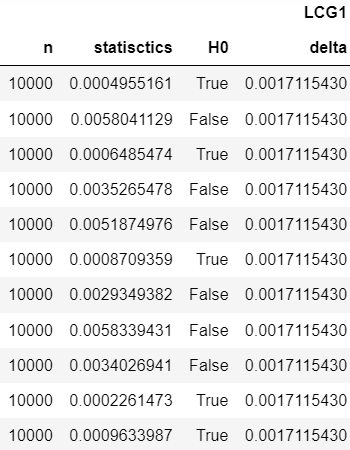
n = 10\*\*3

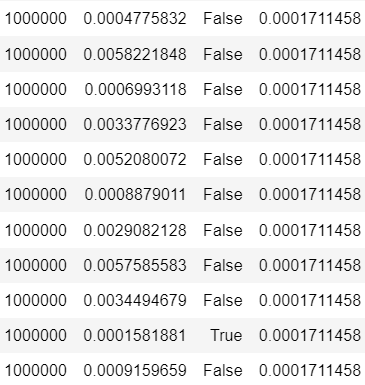
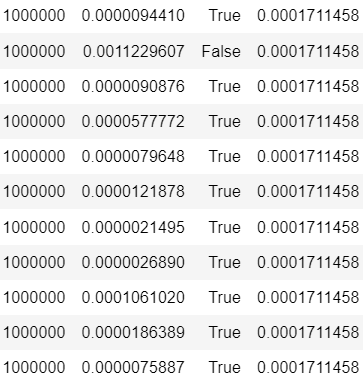
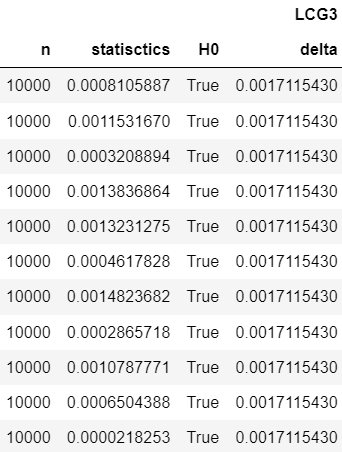
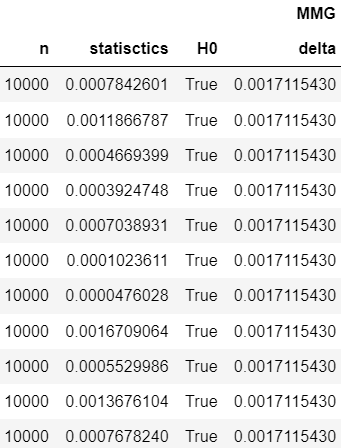
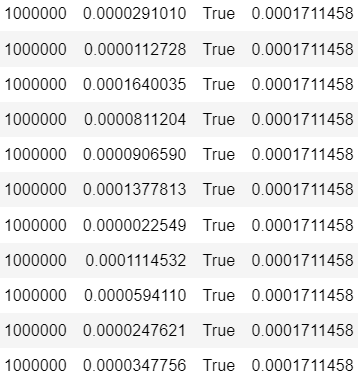
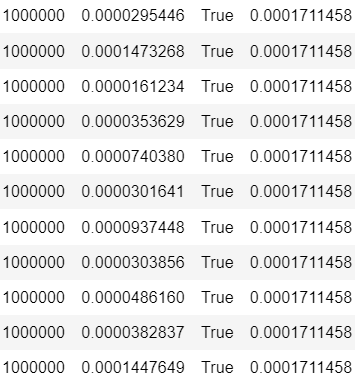
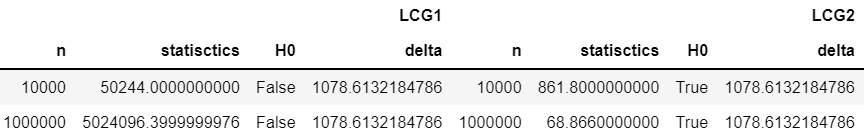
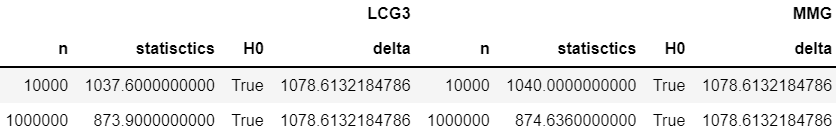
n = 10\*\*4

Тест совпадения матожидания:

Тест совпадения дисперсии:

  
Тест ковариации:



  
  
  
  
  
  
  
  
  
  
  
  
  
Критерий хи-квадрат Пирсона:

# Вывод

По результатам проведенных экспериментов можно сделать вывод, что линейный конгруэнтный генератор с максимальным периодом и построенный на основе двух линейных конгруэнтных генераторов генератор Макларена-Марсальи являются наиболее устойчивыми к статистическим тестам. При этом тест ковариаций оказался более чувствительным, а, следовательно, более строгим тестом, далее – критерий хи-квадрат.

*#!/usr/bin/env python  
# coding: utf-8  
# # S&SM  
# ## L1. Alina Kragel, gr. 9  
# In[1]:*from matplotlib import pyplot as plt  
import numpy as np  
import pandas as pd  
import math  
import enum  
from matplotlib import cycler  
from functools import reduce  
  
from math import gcd  
from scipy.stats import norm, chi2  
  
  
*# In[2]:*colors = cycler(**'color'**, [**'#117A65'**])  
plt.rc(**'axes'**, facecolor=**'#E6E6E6'**, edgecolor=**'none'**,  
 axisbelow=True, grid=True, prop\_cycle=colors)  
plt.rc(**'grid'**, color=**'w'**, linestyle=**'solid'**)  
plt.rc(**'xtick'**, direction=**'out'**, color=**'gray'**)  
plt.rc(**'ytick'**, direction=**'out'**, color=**'gray'**)  
plt.rc(**'patch'**, edgecolor=**'#E6E6E6'**)  
plt.rc(**'lines'**, linewidth=2)  
  
  
*# In[3]:*def set\_styles(df):  
 s = df.style.format(precision=10)  
 return s  
  
  
*# Шпаргалка для LCG  
# In[4]:*def getA(m):  
 def deliteli(val):  
 return [\*filter(lambda x: val % x == 0, range(1, val // 2 + 1)), val]  
  
 def isSimple(val):  
 return val == 1 or len(deliteli(val))==2  
  
 sd = list(filter(lambda x: isSimple(x), deliteli(m)))  
 b = reduce(lambda a, x: a \* x, sd)  
 if (m % 4 == 0 and b % 4 != 0):  
 b \*= 4  
 return b + 1  
  
  
*# ##### 1. Реализация линейного конгруэнтного генератора  
# In[6]:*class LCG:  
  
 def \_\_init\_\_(self, x0, a, c, M):   
 self.a = a  
 self.c = c  
 self.M = M  
 self.x = x0  
   
 def \_\_call\_\_(self):  
 self.x = (self.a \* self.x + self.c) % self.M  
 return self.x / self.M  
  
  
*# ##### 2. Реализация генератора Маклорена-Марсальи  
# In[7]:*class MMG:  
 def \_\_init\_\_(self, g1, g2, k):  
 assert k > 0  
 self.k = k  
 self.g1 = g1  
 self.g2 = g2  
 self.v = [g1() for \_ in range(k)]  
   
 def \_\_call\_\_(self):  
 s = int(self.g2() \* self.k)  
 rx = self.v[s]  
 self.v[s] = self.g1()  
 return rx  
  
  
*# In[8]:*class MMT\_Options(enum.Enum):  
 mean\_option = 0  
 var\_option = 1  
 mean\_and\_var\_option = 2  
  
  
*# ##### - Диаграмма рассеяния  
# In[9]:*def draw\_scatter(rng, n):  
 vals = [rng() for \_ in range(n)]  
 plt.scatter(vals[:-1], vals[1:], s=1)  
   
def draw\_scatter\_for\_gens(n, gens):  
 plt.figure(figsize=(20, 10), dpi=150)  
 plt.suptitle(**f"n =** {n}**"**)  
 plt.figure(figsize=(16, 12))  
 for i, (gn, gg) in enumerate(gens):  
 plt.subplot(2, 2, i + 1)  
 plt.title(gn)  
 draw\_scatter(gg(), n)  
  
  
*# ##### - Тест совпадения моментов:  
# In[10]:*def test\_matching\_moments(rng, n, eps, result\_spec: MMT\_Options):  
 s1 = 0  
 s2 = 0  
   
 for \_ in range(n):  
 ai = rng()  
 s1 += ai  
 s2 += ai \*\* 2  
   
 m = s1 / n  
 s = (n \* s2 - s1 \*\* 2) / (n \* (n - 1))  
   
 xi1 = abs(m - 1 / 2)  
 xi2 = abs(s - 1 / 12)  
   
 c1 = math.sqrt((12 \* n))  
 c2 = (n - 1) / math.sqrt(0.0056 \* n + 0.0028 - 0.0083 / n)  
   
 delta = norm.ppf(1 - eps / 2)  
   
 result = {}  
 if result\_spec is MMT\_Options.mean\_option:  
 result.update({**"statistic"**: round(xi1 \* c1, 10),   
 **"H0"**: xi1 \* c1 < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps})  
 if result\_spec is MMT\_Options.var\_option:  
 result.update({**"statistic"**: round(xi2 \* c2, 10),   
 **"H0"**: xi2 \* c2 < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps})  
 if result\_spec is MMT\_Options.mean\_and\_var\_option:  
 result.update({**"statistic"**: round(xi1 \* c1, 10),   
 **"H0"**: xi1 \* c1 < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps})  
 result.update({**"statistic"**: round(xi2 \* c2, 10),   
 **"H0"**: xi2 \* c2 < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps})  
 return result  
  
  
*# ##### - Тест 'ковариация':  
# In[11]:*def test\_covariation(rng, n, j, eps):  
 pai = rng()  
   
 data\_saver = []  
 for \_ in range(n + j):  
 data\_saver.append(rng())  
   
 rh = sum([data\_saver[i] \* data\_saver[i + j] for i in range(n-j)]) / (n - j - 1) - (n \* np.mean(data\_saver) \*\* 2) / (n - 1)  
 if j == 0:  
 stat = np.fabs(rh - 1 / 12)  
 else:  
 stat = np.fabs(rh)  
 delta = norm.ppf(1 - eps / 2) / (12 \* np.sqrt(n - 1))  
   
 return {**"statistic"**: round(stat, 10),   
 **"H0"**: stat < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps}  
  
  
*# ##### - Критерий хи-квадрат Пирсона:  
# In[12]:*def test\_chi2(rng, n, k, eps):  
 nu = [0] \* k  
 for \_ in range(n):  
 nu[int(rng() \* k)] += 1  
   
 chi2s = 0  
 for vi in nu:  
 chi2s += (vi - n / k) \*\* 2 / (n / k)  
 delta = chi2(k - 1).ppf(1 - eps)  
   
 return {**"statistic"**: chi2s,   
 **"H0"**: chi2s < delta,   
 **"delta"**: round(delta, 10),   
 **"eps"**: eps}

*# ##### Функция для генерации выборок заданными генераторами  
# LCG1: x0 = 19, a = 174, c = 1, M = 173\  
# LCG2: x0 = 2 \*\* 8, a = 75, c = 74, M = 2 \*\* 16 + 1\  
# LCG3: x0 = 2 \*\* 12, a = 1 140 671 485, c = 12 820 163, M = 2 \*\* 24\  
# MMG: LCG3, LCG2, k = 100  
  
# In[13]:*def get\_gens(ext=False):  
   
 gens = []  
 if not ext:  
 gens.append((**"LCG1"**, lambda: LCG(19, 17, 1, 167)))  
 gens.append((**"LCG2"**, lambda: LCG(2 \*\* 8, 75, 74, 2 \*\* 16 + 1)))  
 gens.append((**"LCG3"**, lambda: LCG(2 \*\* 12, 1\_140\_671\_485, 12\_820\_163, 2 \*\* 24)))  
 gens.append((**"MMG"**, lambda: MMG(gens[2][1](), gens[1][1](), 100)))  
 else:  
 gens.append(LCG(19, 17, 1, 167))  
 gens.append(LCG(2 \*\* 8, 75, 74, 2 \*\* 16 + 1))  
 gens.append(LCG(2 \*\* 12, 1140671485, 12820163, 2 \*\* 24))  
 gens.append(MMG(gens[2], gens[1], 100))  
 return gens  
  
  
*# ##### Функция для оптимального проведения тестов  
  
# In[14]:*def get\_data(test, MMT\_Spec: MMT\_Options, eps=0.04):   
 n1 = 10 \*\* 4  
 n2 = 10 \*\* 6  
 gens = get\_gens()  
 bunch = []  
 additional\_data = []  
 gen\_bunch1 = []  
 gen\_bunch2 = []  
   
   
 if test == **'MMT'**:  
 for title, gen in gens:  
 mm\_res\_1 = test\_matching\_moments(gen(), n1, eps, MMT\_Spec)  
 mm\_res\_2 = test\_matching\_moments(gen(), n2, eps, MMT\_Spec)  
 gen\_bunch1.extend([n1, mm\_res\_1[**'statistic'**], str(mm\_res\_1[**'H0'**]), mm\_res\_1[**'delta'**]])  
 gen\_bunch2.extend([n2, mm\_res\_2[**'statistic'**], str(mm\_res\_2[**'H0'**]), mm\_res\_2[**'delta'**]])  
 bunch.append(gen\_bunch1)  
 bunch.append(gen\_bunch2)  
   
   
 if test == **'CT'**:  
 cells = 11  
 bunch = [[] for \_ in range(cells \* 2)]  
 for title, gen in gens:  
 gen\_bunch = []  
 for size in [n1, n2]:  
 for cell in range(cells):  
 cov\_res = test\_covariation(gen(), size, cell, eps)  
 gen\_bunch.append([size, cov\_res[**'statistic'**], str(cov\_res[**'H0'**]), cov\_res[**'delta'**]])  
   
 for gen\_el, bunch\_el in zip(gen\_bunch, bunch):  
 bunch\_el.extend(gen\_el)  
  
   
 if test == **'CHI'**:  
 for title, gen in gens:  
 chi1\_res = test\_chi2(gen(), n1, 1000, eps)  
 chi2\_res = test\_chi2(gen(), n2, 1000, eps)  
 gen\_bunch1.extend([n1, chi1\_res[**'statistic'**], str(chi1\_res[**'H0'**]), chi1\_res[**'delta'**]])  
 gen\_bunch2.extend([n2, chi2\_res[**'statistic'**], str(chi2\_res[**'H0'**]), chi2\_res[**'delta'**]])  
 bunch.append(gen\_bunch1)  
 bunch.append(gen\_bunch2)  
   
 return bunch, additional\_data  
  
  
*# ##### Отчетная функция  
# In[15]:*def get\_report\_for\_test(test, MMT\_Spec=MMT\_Options.mean\_option):  
 data, temp\_data = get\_data(test, MMT\_Spec)  
 if temp\_data:  
 temp\_df = pd.DataFrame(temp\_data,  
 index=pd.Index([**'LCG1'**, **'LCG2'**, **'LCG3'**, **'MMG'**]),  
 columns=[**'Mean'**, **'Variance'**])  
  
 df = pd.DataFrame(data,  
 columns=pd.MultiIndex.from\_product([[**'LCG1'**, **'LCG2'**, **'LCG3'**, **'MMG'**],  
 [**'n'**, **'statisctics'**, **'H0'**, **'delta'**]],   
 names=[**'Gens:'**, **' '**]))  
 if temp\_data:  
 return set\_styles(df), set\_styles(temp\_df)  
 return set\_styles(df)  
  
  
*# ### Проведение эксперимента  
# In[16]:*n = 10 \*\* 6  
  
  
*# In[17]:*gens = get\_gens(ext=True)  
data = []  
table\_data = []  
  
for gen in gens:  
 vals = [gen() for \_ in range(n)]  
 data.append(vals)  
   
for el in data:  
 table\_data.append([np.mean(el), np.var(el, ddof=1)])  
  
  
*# Рассчитанные математические ожидания и дисперсии для сгенерированных выборок  
# In[18]:*temp\_df = set\_styles(pd.DataFrame(table\_data,  
 index=pd.Index([**'LCG1'**, **'LCG2'**, **'LCG3'**, **'MMG'**]),  
 columns=[**'Mean'**, **'Variance'**]))  
temp\_df  
  
  
*# #### Диаграммы рассеяния  
# In[19]:*draw\_scatter\_for\_gens(10\_000, get\_gens())  
  
  
*# In[27]:*draw\_scatter\_for\_gens(100\_000, get\_gens())

*# #### Результаты проведения тестов совпадения моментов для математического ожидания  
# In[21]:*get\_report\_for\_test(**'MMT'**, MMT\_Options.mean\_option)  
  
  
*# #### Результаты проведения тестов совпадения моментов для дисперсии  
# In[22]:*get\_report\_for\_test(**'MMT'**, MMT\_Options.var\_option)  
  
  
*# #### Результаты проведения теста ковариаций  
# In[23]:*get\_report\_for\_test(**'CT'**)  
  
  
*# #### Результаты проведения критерия хи-квадрат  
# In[24]:*get\_report\_for\_test(**'CHI'**)