**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра теории вероятностей и математической статистики

**Крагель Алины Олеговны**

**Моделирование непрерывной случайной величины**

Отчет по лабораторной работе №2

(«Имитационное и статистическое моделирование»)

Студентки 4 курса 9 группы

**Преподаватель**

*Гайдук Антон Николаевич*

# Теоретическая часть

## Моделирование НСВ

### Распределение геометрическое

Дискретная случайная величина *X = m* имеет геометрическое распределение с параметром *p*, если она принимает значения *1, 2, …, m, …* (бесконечное, но счетное множество значений) с вероятностями:

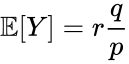
, .

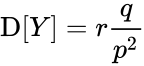
Математическое ожидание случайной величины *X*, имеющей геометрическое распределение с параметром *p*:

Дисперсия случайной величины *X*, имеющей геометрическое распределение:

### Распределение отрицательное биномиальное

Дискретная случайная величина *X = m* имеет отрицательное биномиальное распределение с параметрами *p* и *k*, где *k* – число «успехов», если она принимает значения *1, 2, …, m, …* (бесконечное, но счетное множество значений) с вероятностями:

Математическое ожидание случайной величины *X*, имеющей отрицательное биномиальное распределение с параметром *p*:

Дисперсия случайной величины *X*, имеющей отрицательное биномиальное распределение:

### Распределение Пуассона

Дискретная случайная величина имеет распределение Пуассона с параметром λ, если:



Математическое ожидание и дисперсия случайной величины, распределенной по закону Пуассона, совпадают и равны параметру этого закона:





### Тест «Хи-квадрат»

Область возможных значений случайной величины разбивается на интервалы .

Рассматривается следующая статистика:

,

где – объем выборки,

- количество элементов выборки, попавших в k-ый интервал,

- вероятность попадания случайной величины в k-ый интервал.

Статистика характеризует взвешенную сумму квадратов уклонений частот {} от гипотетических значений {}. Чем больше , тем “сильнее” выборка не согласуется с .

Проверяется условие:

Тест «Хи-квадрат» позволяет проверить гипотезу для последовательности  и описывается следующим решающим правилом:

где порог критерия ∆ находится из ограничения на ошибку первого рода и имеет вид:

,

где *G* – функция распределения распределения,

– уровень значимости (чаще всего принимаем за 0.05).

В рамках данной лабораторной работы рассматриваемый отрезок [0;1] разбивался на 10 интервалов.

# Практическая часть

## Отрицательное биномиальное распределение

## 

## Геометрическое распределение

## Распределение Пуассона

# Вывод

Результаты тестирования позволяют сделать вывод, что построенная с помощью генератора Маклорена-Марсальи последовательность действительно могут служить непрерывной случайной величиной. Практически на всех тестах принимается гипотеза о случайности последовательности. Однако, в силу наличия результата для отрицательного биномиального распределения, не проходящего критерий хи-квадрат, можно сделать вывод, что можно подобрать лучшие параметры для генератора Маклорена-Марсальи.

Были построены непрерывные случайные величины с геометрическим, отрицательным биномиальным законами распределения, с распределением Пуассона. Для них были подсчитаны статистики. На основе теста Хи-квадрат 2 из 3 последовательностей являются «хорошими».

*#!/usr/bin/env python  
# coding: utf-8  
# # S&SM  
# ## L2. 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 scipy.special import comb  
from scipy import stats  
from math import gcd  
from scipy.stats import norm, chi2  
  
  
*# In[2]:*colors = cycler(**'color'**, [**'#117A65'**, **'#2D6C5F'**])  
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=**'#2D6C5F'**)  
plt.rc(**'lines'**, linewidth=2)  
  
  
*# ##### Подготовительный этап  
# Генераторы прошлой лабы.  
# In[3]:*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.x \* self.a + self.c) % self.M  
 return self.x / self.M  
   
 def get\_period(self):  
 s = list()  
 while True:  
 x = self()  
 if x in s:  
 break   
 s.append(x)  
 return len(s)  
  
  
*# In[4]:*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 rand(self):  
 s = int(self.g2() \* self.k)  
 rx = self.v[s]  
 self.v[s] = self.g1()  
 return rx  
  
  
*# ##### Параметры генераторов  
# 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[5]:*MMG = MMG(LCG(2 \*\* 12, 1\_140\_671\_485, 12\_820\_163, 2 \*\* 24), LCG(2 \*\* 8, 75, 74, 2 \*\* 16 + 1), 100)  
  
  
*# Оценка МО  
# In[6]:*def get\_expectation(val):  
 e\_obs = 0  
 for x in val:  
 e\_obs += x   
 e\_obs /= len(val)  
 return e\_obs  
  
  
*# Оценка Дисперсии  
# In[7]:*def get\_dispersion(val):  
 d = 0  
 mean = 0  
 for x in val:  
 mean += x  
 mean /= len(val)  
 for x in val:  
 d += (x - mean)\*\*2  
 d /= (len(val) - 1)  
 return d  
  
  
*# Хи-Квадрат  
# In[8]:*def chisquare(frec\_obs, frec\_exp):  
 chisquare = 0.0  
 for i in range(len(frec\_obs)):  
 chisquare += ((frec\_obs[i] - frec\_exp[i])\*\*2 / frec\_exp[i])  
 return chisquare  
  
  
*# Длина и уровень значимости  
# In[9]:*n = 1000  
eps = 0.05  
  
  
*# #### Отрицательное биномиальное распределение  
# ##### Получаем случайное число через свой генератор по заданному распределению  
# In[10]:*def genDRV\_bin(m, p):  
 q = 1 - p  
 p = p\*\*m  
  
 r = MMG.rand() - p  
   
 z = 0  
 while(r >= 0):  
 z += 1  
 p = p \* q \* ((m - 1 + z) / z)  
 r -= p  
   
 return z  
  
  
*# ##### Выборка заданного распределения через встроенный генератор  
# In[11]:*def p\_binom(x, r, p):  
 if (x < 0):  
 return 0  
 else:  
 return comb(x + r - 1, x)\*(p\*\*r)\*((1-p)\*\*x)  
  
  
*# In[12]:  
# пределы гистограммы от 0 до 2\*E, считаем ожидаемые частоты попадания в каждую область гистограммы*def exp\_frec\_binom(r, p):  
 lim = 2 \* math.floor(r \* (1 - p) / p)  
 frec\_exp = []  
 p\_sum = 0  
  
 for i in range(lim):  
 \_p = p\_binom(i, r, p)  
 p\_sum += \_p  
 frec\_exp.append(\_p \* n)  
   
 frec\_exp.append(n \* (1 - p\_sum))  
 return frec\_exp  
  
  
*# In[13]:  
# считаем выборочные частоты попадания в области гистограммы*def obs\_frec\_binom(val\_obs, r, p):  
 lim = 2 \* math.floor(r \* (1 - p) / p)  
 f\_obs = []  
 for i in range(lim + 1):  
 f\_obs.append(0)  
   
 val\_sorted = sorted(val\_obs)  
 for val in val\_sorted:  
 if(val < lim):  
 f\_obs[val] += 1  
 else:  
 f\_obs[lim] += 1  
   
 return f\_obs  
  
  
*# In[14]:*r = 8  
p = 0.4  
  
  
*# In[15]:*val = list()  
for i in range(n):  
 val.append(genDRV\_bin(r, p))  
  
  
*# In[16]:*e\_obs = get\_expectation(val)  
d\_obs = get\_dispersion(val)  
  
  
*# In[17]:  
# для отрицательно биномиального распределения матожидание равно (r\*q) / p  
# для отрицательно биномиального распределения дисперсия равна (r\*q) / (p^2)*e\_exp = r \* (1 - p) / p  
d\_exp = r \* (1 - p) / (p\*\*2)   
  
  
*# In[18]:*frec\_exp = exp\_frec\_binom(r, p)  
frec\_obs = obs\_frec\_binom(val, r, p)  
  
  
*# In[19]:*k = len(frec\_exp) - 1  
delta = stats.chi2(k - 1).ppf(1 - eps)  
  
  
*# In[20]:*print(**'E\_exp = '** + format(e\_exp, **'.2f'**))  
print(**'E\_obs = '** + format(e\_obs, **'.2f'**))  
print(**'D\_exp = '** + format(d\_exp, **'.2f'**))  
print(**'D\_obs = '** + format(d\_obs, **'.2f'**))  
print(**'Chisq = '** + format(chisquare(frec\_obs, frec\_exp), **'.2f'**))  
print(**'Pass = '**, chisquare(frec\_obs, frec\_exp) < delta)  
  
  
*# In[21]:*X\_val = range(2 \* math.floor(r \* (1 - p) / p) + 1)  
X, Ye, Yo = np.array(X\_val), np.array(frec\_exp), np.array(frec\_obs)  
  
  
*# In[22]:*plt.figure(figsize=(15, 9))  
plt.plot(X, Ye, **'go'**)  
plt.plot(X, Ye, **'g-'**, label = **'expected'**)  
plt.plot(X, Yo, **'ro'**)  
plt.plot(X, Yo, **'r-'**, label = **'observed'**)  
plt.legend()  
plt.show()  
  
  
*# #### Геометрическое распределение  
# In[23]:*def genDRV\_geom(p):  
 a = MMG.rand()  
 return math.floor(math.log(a) / math.log(1 - p)) + 1  
  
  
*# In[24]:*def p\_geom(x, p=0.25):  
 if (x < 1):  
 return 0  
 else:  
 return p \* ((1 - p)\*\*(x - 1))  
  
  
*# In[25]:*lim = math.floor(6/p)  
  
  
*# In[26]:  
# пределы гистограммы от 1 до lim, считаем ожидаемые частоты попадания в каждую область гистограммы*def exp\_frec\_geom(p=0.25):  
 frec\_exp = []  
 p\_sum = 0  
  
 for i in range(1, lim):  
 \_p = p\_geom(i)  
 p\_sum += \_p  
 frec\_exp.append(\_p \* n)  
   
 frec\_exp.append(n \* (1 - p\_sum))  
 return frec\_exp  
  
  
*# In[27]:  
# считаем выборочные частоты попадания в области гистограммы*def obs\_frec\_geom(val\_obs, p=0.25):  
 f\_obs = []  
 for i in range(1, lim + 1):  
 f\_obs.append(0)  
   
 *# calculate observed frequences* val\_sorted = sorted(val\_obs)  
 for val in val\_sorted:  
 if(val < lim):  
 f\_obs[val - 1] += 1  
 else:  
 f\_obs[lim - 1] += 1  
   
 return f\_obs  
  
  
*# In[28]:*p = 0.25  
  
  
*# In[29]:*val = list()  
for i in range(n):  
 val.append(genDRV\_geom(p))  
  
  
*# In[30]:  
# найдем оценки матожидания и дисперсии по сгенерированной выборке*e\_obs = get\_expectation(val)  
d\_obs = get\_dispersion(val)  
  
  
*# In[31]:  
# для геометрического распределения матожидание равно 1 / p   
# для геометрического распределения дисперсия равна q / (p\*\*2)*e\_exp = 1 / p  
d\_exp = (1 - p) / (p\*\*2)  
  
  
*# In[32]:*frec\_exp = exp\_frec\_geom()  
frec\_obs = obs\_frec\_geom(val)  
  
  
*# In[33]:*k = len(frec\_exp) - 1  
delta = stats.chi2(k - 1).ppf(1 - eps)  
  
  
*# In[34]:*print(**'E\_exp = '** + format(e\_exp, **'.2f'**))  
print(**'E\_obs = '** + format(e\_obs, **'.2f'**))  
print(**'D\_exp = '** + format(d\_exp, **'.2f'**))  
print(**'D\_obs = '** + format(d\_obs, **'.2f'**))  
print(**'Chisq = '** + format(chisquare(frec\_obs, frec\_exp), **'.2f'**))  
print(**'Pass = '**, chisquare(frec\_obs, frec\_exp) < delta)  
  
  
*# In[35]:*X\_val = range(1, lim + 1)  
X, Ye, Yo = np.array(X\_val), np.array(frec\_exp), np.array(frec\_obs)  
  
  
*# In[36]:*plt.figure(figsize=(15, 9))  
plt.plot(X, Ye, **'go'**)  
plt.plot(X, Ye, **'g-'**, label = **'expected'**)  
plt.plot(X, Yo, **'ro'**)  
plt.plot(X, Yo, **'r-'**, label = **'observed'**)  
plt.legend()  
plt.show()  
  
  
*# #### Распределение Пуассона  
# In[37]:*def genDRV\_puas(l):  
 p = math.exp(-1 \* l)  
 r = MMG.rand() - p  
 x = 0  
 while(r >= 0):  
 x += 1  
 p = (p \* l) / x  
 r -= p  
 return x  
  
  
*# In[38]:  
  
  
# РАСПРЕДЕЛЕНИЕ ПУАССОНА*def p\_puas(x, l=3):  
 if (x < 0):  
 return 0  
 else:  
 return math.exp(-1 \* l) \* (l\*\*x) / math.factorial(x)  
  
  
*# In[39]:*def exp\_frec\_puas(l=3):  
 frec\_exp = []  
 p\_sum = 0  
  
 for i in range(2\*l + 1):  
 p = p\_puas(i)  
 p\_sum += p  
 frec\_exp.append(p \* n)  
   
 frec\_exp.append(n \* (1 - p\_sum))  
 return frec\_exp  
  
  
*# In[40]:*def obs\_frec\_puas(val\_obs, l=3):  
 f\_obs = []  
 for i in range(2\*l + 2):  
 f\_obs.append(0)  
   
 val\_sorted = sorted(val\_obs)  
 for val in val\_sorted:  
 if(val <= 2\*l):  
 f\_obs[val] += 1  
 else:  
 f\_obs[2\*l + 1] += 1  
   
 return f\_obs  
  
  
*# In[41]:*l = 3  
  
  
*# In[42]:*val = list()  
for i in range(n):  
 val.append(genDRV\_puas(l))  
  
  
*# In[43]:  
# найдем оценки матожидания и дисперсии по сгенерированной выборке*e\_obs = get\_expectation(val)  
d\_obs = get\_dispersion(val)  
  
  
*# In[44]:  
# для распределения Пуассона матожидание и дисперсия совпадают со значением l*e\_exp = l  
d\_exp = l  
  
  
*# In[45]:*frec\_exp = exp\_frec\_puas()  
frec\_obs = obs\_frec\_puas(val)  
  
  
*# In[46]:*k = len(frec\_exp) - 1  
delta = stats.chi2(k - 1).ppf(1 - eps)  
  
  
*# In[47]:*print(**'E\_exp = '** + format(e\_exp, **'.2f'**))  
print(**'E\_obs = '** + format(e\_obs, **'.2f'**))  
print(**'D\_exp = '** + format(d\_exp, **'.2f'**))  
print(**'D\_obs = '** + format(d\_obs, **'.2f'**))  
print(**'Chisq = '** + format(chisquare(frec\_obs, frec\_exp), **'.2f'**))  
print(**'Pass = '**, chisquare(frec\_obs, frec\_exp) < delta)  
  
  
*# In[48]:*X\_val = range(2\*l + 2)  
X, Ye, Yo = np.array(X\_val), np.array(frec\_exp), np.array(frec\_obs)  
  
  
*# In[49]:*plt.figure(figsize=(15, 9))  
plt.plot(X, Ye, **'go'**)  
plt.plot(X, Ye, **'g-'**, label = **'expected'**)  
plt.plot(X, Yo, **'ro'**)  
plt.plot(X, Yo, **'r-'**, label = **'observed'**)  
plt.legend()  
plt.show()