

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ  
Кафедра технологий программирования

Толкун Кирилл Юрьевич

**Отчёт по лабораторным работам по курсу  
“Имитационное и статистическое моделирование”**

студента 4 курса 8 группы

**Вариант 10**

**Преподаватель:**  
Лобач Сергей Викторович  
ассистент кафедры ММАД

Работа сдана \_\_\_\_\_ 2020 г.

Зачтена \_\_\_\_\_ 2020 г.

---

(подпись преподавателя)

Минск  
2020

# 1 Лабораторная 1

## 1.1 Условие

Согласно варианту:  $X_0 = \alpha = 16807, K = 64$ .

Используя метод Маклерена-Марсальи построить датчик БСВ (1 датчик должен быть мультипликативно конгруэнтный, второй – на выбор). Исследовать точность построенной БСВ.

1. Осуществить моделирование  $n = 1000$  реализаций БСВ с помощью мультипликативного конгруэнтного метода (МКМ) с параметрами  $X_0, \alpha, m = 231$ ;
2. Осуществить моделирование  $n = 1000$  реализаций БСВ с помощью метода Макларена-Марсальи (один датчик должен быть мультипликативно конгруэнтный (п. 1), второй – на выбор).  $K$  – объем вспомогательной таблицы;
3. Проверить точность моделирования обоих датчиков (п. 1 и п. 2) с помощью критерия согласия Колмогорова и  $\chi^2$  - критерия Пирсона с уровнем значимости  $\varepsilon = 0.05$ .

## 1.2 Теория

### 1.2.1 Датчики БСВ

Для моделирования на ЭВМ реализаций *Базовой случайной величины* используются специальные программы, называемые программными датчиками БСВ.

В основе программных датчиков БСВ лежат рекуррентные формулы вида:

$$x_n = \varphi(x_{n-1}, \dots, x_{n-p}), n = 1, 2, \dots, \quad (1.1)$$

где  $x_{1-p}, x_{2-p}, \dots, x_0$  ( $p \geq 1$ ) - заданные стартовые значения. Описанное соотношение (??) описывает детерминированный алгоритм, однако при соответствующем подборе преобразования  $\varphi(\cdot)$  получаемые на его основе псевдослучайные числа  $x_n$  по своим функциональным и числовым характеристикам близки к БСВ.

Алгоритмы моделирования вида (??) обладают общим недостатком: начиная с некоторого момента  $t_0$ , последовательность псевдослучайных чисел образует цикл, который повторяется бесконечное число раз. Длина  $T$  циклически повторяющейся последовательности называется *периодом датчика* БСВ ( $T \leq m - 1$ ).

Период  $T$  и *коэффициент использования* БСВ  $k$  являются основными показателями качества программных датчиков БСВ. Лучшим датчикам соответствуют большие значения  $T$  и  $k$ .

### 1.2.2 Линейный конгруэнтный метод

**Линейный конгруэнтный метод** - один из методов генерации псевдослучайных чисел. Применяется в простых случаях и не обладает криптографической стойкостью. Входит в библиотеки различных компиляторов.

Суть метода заключается в вычислении последовательности случайных чисел  $X_n$  следующим образом:

$$X_{n+1} = \frac{\alpha X_n + c) \bmod m}{m}, \quad (1.2)$$

где:

- |   |   |                             |
|---|---|-----------------------------|
| <ol style="list-style-type: none"> <li>1. <math>X_0</math> - начальное значение (<math>0 \leq X_0 &lt; m</math>)</li> <li>2. <math>\alpha</math> - множитель (<math>0 \leq \alpha &lt; m</math>)</li> <li>3. <math>c</math> - приращение (<math>0 \leq c &lt; m</math>)</li> <li>4. <math>m \geq 2</math> - модуль</li> </ol> | } | - <i>параметры датчика.</i> |
|---|---|-----------------------------|

**Типовые значения параметров:**  $m = 2^{31}$ ,  $x_0 = \alpha = 65539$ .

### 1.2.3 Мультипликативный конгруэнтный метод

Метод генерации *линейной конгруэнтной последовательности* (раздел ??) при  $c = 0$  называют *мультипликативным конгруэнтным методом*.

### 1.2.4 Метод Макларена-Марсальи

**Генератор Макларена-Марсальи** - криптографически стойкий генератор псевдослучайных чисел, который основан на комбинации двух конгруэнтных генераторов и вспомогательной матрице, с помощью которой происходит перемешивание двух последовательностей, полученных от двух генераторов.

Данный генератор псевдослучайных чисел оперирует с тремя объектами: двумя конгруэнтными генераторами, которые порождают последовательности  $\langle \mathbf{X}_n \rangle$ ,  $\langle \mathbf{Y}_n \rangle$ , и массива  $\mathbf{V}$ , состоящей из  $k$  элементов, обычно  $k \in \{64, 28, 256\}$ . На выходе последовательность  $\langle \mathbf{Z}_n \rangle$ .

Генератор состоит из четырёх основных стадий:

1. Инициализация  $\mathbf{V}$  и  $\mathbf{Z}$  первыми  $k$  элементами последовательности  $\langle \mathbf{X}_n \rangle$  - выполняется один раз;
2. Выборка  $\mathbf{X}$ ,  $\mathbf{Y}$  из  $\langle \mathbf{X}_n \rangle$ ,  $\langle \mathbf{Y}_n \rangle$ , то есть  $\mathbf{X}$ ,  $\mathbf{Y}$  - очередные члены последовательностей  $\langle \mathbf{X}_n \rangle$ ,  $\langle \mathbf{Y}_n \rangle$ ;
3. Вычисление  $\mathbf{j} = \lfloor \mathbf{k} \cdot \mathbf{Y} \rfloor$ , где  $\mathbf{j} \in [0, k)$  - случайное число, определяемое  $\mathbf{Y}$ ;
4. Присвоение  $\mathbf{Z}_i = \mathbf{V}_i$  и замена  $\mathbf{V}_j = \mathbf{X}$ .

Последние три стадии могут повторяться необходимое число раз.

Данный метод позволяет ослабить зависимость между членами последовательности  $\mathbf{Z}_n$  и получить сколь угодно большие значения её периода  $T$  при условии, что периоды  $T_1, T_2$  исходных датчиков являются взаимно простыми числами. Коэффициент использования БСВ для данного датчика  $k = \frac{1}{2}$  (за исключением первой реализации, для моделирования которой используется  $K + 1$  реализация).

### 1.2.5 $\chi^2$ критерий согласия Пирсона

**Критерий согласия Пирсона** - это непараметрический метод, который позволяет оценить значимость различий между фактическим (выявленным в результате исследования) количеством исходов или качественных характеристик выборки, попадающих в каждую категорию, и теоретическим количеством, которое можно ожидать в изучаемых группах при справедливости нулевой гипотезы. Метод позволяет оценить статистическую значимость различий двух или нескольких относительных показателей (частот, долей).

Данный критерий применяют для проверки гипотезы о соответствии эмпирического распределения предполагаемому теоретическому распределению  $F(X)$  при большом объёме выборки ( $n \geq 100$ ). Критерий применим для любых видов функции  $F(x)$ , даже при неизвестных значениях их параметров, что обычно имеет место при анализе результатов механических испытаний.

Статистика критерия проверки гипотез имеет вид:

$$\chi^2 = \sum_{i=1}^p \frac{(n_i - n \cdot p_i)^2}{n \cdot p_i}, \quad (1.3)$$

где  $n_i$  - наблюдаемые частоты,  $n \cdot p_i$  - ожидаемые частоты.

Чем больше  $\chi^2$ , тем сильнее выборка  $X$  не согласуется с гипотезой  $H_0$  (**нулевая гипотеза**: наблюдаемые частоты соответствуют ожидаемым).

Чтобы проверить гипотезу по критерию Пирсона необходимо сравнить статистику критерия с критическим значением, которой находится в таблице для соответствующего уровня значимости и количеству степеней свободы.

**Пример:** при уровне значимости  $\alpha = 0.05$  и количестве степеней свободы  $\nu = 9$  критерий Пирсона согласуется с нулевой гипотезой при  $\chi^2 < 16.919$ .

$\nu \backslash \alpha$	0,99	0,98	0,95	0,90	0,80	0,70	0,50	0,30	0,20	0,10	0,05	0,02	0,01	$\alpha / \nu$
1	0,00016	0,00628	0,00393	0,0158	0,0642	0,148	0,455	1,074	1,642	2,706	3,841	5,412	6,635	1
2	0,0201	0,0404	0,103	0,211	0,446	0,713	1,386	2,408	3,219	4,605	5,991	7,824	9,210	2
3	0,115	0,185	0,352	0,584	1,005	1,424	2,366	3,605	4,642	6,251	7,815	9,837	11,345	3
4	0,297	0,429	0,711	1,064	1,649	2,195	3,357	4,878	5,989	7,779	9,488	11,668	13,277	4
5	0,554	0,752	1,145	1,610	2,343	3,000	4,351	6,064	7,289	9,236	11,070	13,388	15,086	5
6	0,872	1,134	1,635	2,204	3,070	3,828	5,348	7,231	8,558	10,645	12,592	15,033	16,812	6
7	1,239	1,564	2,167	2,833	3,822	4,671	6,346	8,383	9,803	12,017	14,067	16,622	18,475	7
8	1,646	2,032	2,733	3,490	4,594	5,527	7,344	9,524	11,030	13,362	15,507	18,168	20,090	8
9	2,088	2,532	3,325	4,168	5,380	6,393	8,343	10,656	12,242	14,684	16,919	19,679	21,666	9
10	2,558	3,059	3,940	4,865	6,179	7,267	9,342	11,781	13,442	15,987	18,307	21,161	23,209	10
11	3,053	3,609	4,575	5,578	6,989	8,148	10,341	12,899	14,631	17,275	19,675	22,618	24,725	11
12	3,571	4,178	5,226	6,304	7,807	9,034	11,340	14,011	15,812	18,549	21,026	24,054	26,217	12
13	4,107	4,765	5,892	7,042	8,634	9,926	12,340	15,119	16,985	19,812	22,362	25,472	27,688	13
14	4,660	5,368	6,571	7,790	9,467	10,821	13,339	16,222	18,151	21,064	23,685	26,873	29,141	14
15	5,229	5,985	7,261	8,547	10,307	11,721	14,339	17,322	19,311	22,307	24,996	28,259	30,578	15
16	5,812	6,614	7,962	9,312	11,152	12,624	15,338	18,418	20,465	23,542	26,296	29,633	32,000	16
17	6,408	7,255	8,672	10,085	12,002	13,531	16,338	19,511	21,615	24,769	27,587	30,995	33,409	17
18	7,015	7,906	9,390	10,865	12,857	14,440	17,338	20,601	22,760	25,989	28,869	32,346	34,805	18
19	7,633	8,567	10,117	11,651	13,716	15,352	18,338	21,689	23,900	27,204	30,144	33,687	36,191	19
20	8,260	9,237	10,851	12,443	14,578	16,266	19,337	22,775	25,038	28,412	31,410	35,020	37,566	20

Рис. 1: Значения  $\chi^2$  при различных  $P_{\chi^2}$  в зависимости от числа степеней свобод  $\nu$ .

### 1.2.6 Критерий согласия Колмогорова

**Критерий согласия Колмогорова** предназначен для проверки гипотезы о принадлежности выборки некоторому закону распределения, то есть проверки того, что эмпирическое распределение соответствует предполагаемой модели.

**Эмпирическая функция распределения**  $F_n$ , построенная по выборке  $X = (X_1, \dots, X_n)$ , имеет вид:

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n I_{X_i \leq x} \quad (1.4)$$

где  $I_{X_i \leq x}$  указывает, попало ли наблюдение  $X_i$  в область  $(-\infty, x]$ :

$$I_{X_i \leq x} = \begin{cases} 1, & X_i \leq x \\ 0, & X_i > x \end{cases} \quad (1.5)$$

**Статистика критерия** для эмпирической функции распределения  $F_n(x)$  определяется следующим образом:

$$D_n = \sup_{x \in R} |F_n(x) - F(x)| \quad (1.6)$$

**Принятие решения по критерию Колмогорова:** В случае справедливости *нулевой гипотезы* ( $H_0$ ) при  $n \rightarrow +\infty$  статистика  $D_n$  имеет распределение Колмогорова:

$$\lim_{n \rightarrow \infty} P(\sqrt{n}D_n < x) = K(x) \quad (1.7)$$

здесь

$$K(x) = \sum_{k=-\infty}^{+\infty} (-1)^k e^{-2k^2 x^2} \approx 1 - 2e^{-2x^2}, x \geq 0 \quad (1.8)$$

- функция Колмогорова.

При *уровне значимости*  $\alpha$  пороговое значение  $C_\alpha$ , находится из соотношения:

$$K(C_\alpha) = 1 - \alpha \quad (1.9)$$

Таким образом, для проверки гипотезы о виде распределения получаем:

$$\rho(X) = \begin{cases} H_0, & \sqrt{n}D_n \leq \alpha, \\ H_1, & \sqrt{n}D_n > \alpha. \end{cases} \quad (1.10)$$

$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$	$C_\alpha$	$\alpha$
<b><math>\alpha \leq 0,29</math></b>	1,00000	<b>0,62</b>	0,8368	<b>0,95</b>	0,3275	<b>1,28</b>	0,0755	<b>1,61</b>	0,0112	<b>1,94</b>	0,0011	<b>2,27</b>	0,0001
<b>0,30</b>	0,99999	<b>0,63</b>	0,8222	<b>0,96</b>	0,3154	<b>1,29</b>	0,0717	<b>1,62</b>	0,0105	<b>1,95</b>	0,0010	<b>2,28</b>	0,0001
<b>0,31</b>	0,99998	<b>0,64</b>	0,8073	<b>0,97</b>	0,3036	<b>1,30</b>	0,0681	<b>1,63</b>	0,0098	<b>1,96</b>	0,0009	<b>2,29</b>	0,0001
<b>0,32</b>	0,99995	<b>0,65</b>	0,7920	<b>0,98</b>	0,2921	<b>1,31</b>	0,0646	<b>1,64</b>	0,0092	<b>1,97</b>	0,0009	<b>2,30</b>	0,0001
<b>0,33</b>	0,99991	<b>0,66</b>	0,7764	<b>0,99</b>	0,2809	<b>1,32</b>	0,0613	<b>1,65</b>	0,0086	<b>1,98</b>	0,0008	<b>2,31</b>	0,000046
<b>0,34</b>	0,99993	<b>0,67</b>	0,7604	<b>1,00</b>	0,2700	<b>1,33</b>	0,0582	<b>1,66</b>	0,0081	<b>1,99</b>	0,0007	<b>2,32</b>	0,000042
<b>0,35</b>	0,9997	<b>0,68</b>	0,7442	<b>1,01</b>	0,2594	<b>1,34</b>	0,0551	<b>1,67</b>	0,0076	<b>2,00</b>	0,0007	<b>2,33</b>	0,000038
<b>0,36</b>	0,9995	<b>0,69</b>	0,7278	<b>1,02</b>	0,2492	<b>1,35</b>	0,0522	<b>1,68</b>	0,0071	<b>2,01</b>	0,0006	<b>2,34</b>	0,000035
<b>0,37</b>	0,9992	<b>0,70</b>	0,7112	<b>1,03</b>	0,2392	<b>1,36</b>	0,0495	<b>1,69</b>	0,0066	<b>2,02</b>	0,0006	<b>2,35</b>	0,000032
<b>0,38</b>	0,9987	<b>0,71</b>	0,6945	<b>1,04</b>	0,2296	<b>1,37</b>	0,0469	<b>1,70</b>	0,0062	<b>2,03</b>	0,0005	<b>2,36</b>	0,000030
<b>0,39</b>	0,9981	<b>0,72</b>	0,6777	<b>1,05</b>	0,2202	<b>1,38</b>	0,0444	<b>1,71</b>	0,0058	<b>2,04</b>	0,0005	<b>2,37</b>	0,000027
<b>0,40</b>	0,9972	<b>0,73</b>	0,6609	<b>1,06</b>	0,2111	<b>1,39</b>	0,0420	<b>1,72</b>	0,0054	<b>2,05</b>	0,0004	<b>2,38</b>	0,000024
<b>0,41</b>	0,9960	<b>0,74</b>	0,6440	<b>1,07</b>	0,2024	<b>1,40</b>	0,0397	<b>1,73</b>	0,0050	<b>2,06</b>	0,0004	<b>2,39</b>	0,000022
<b>0,42</b>	0,9945	<b>0,75</b>	0,6272	<b>1,08</b>	0,1939	<b>1,41</b>	0,0375	<b>1,74</b>	0,0047	<b>2,07</b>	0,0004	<b>2,40</b>	0,000020
<b>0,43</b>	0,9926	<b>0,76</b>	0,6104	<b>1,09</b>	0,1857	<b>1,42</b>	0,0354	<b>1,75</b>	0,0044	<b>2,08</b>	0,0004	<b>2,41</b>	0,000018
<b>0,44</b>	0,9903	<b>0,77</b>	0,5936	<b>1,10</b>	0,1777	<b>1,43</b>	0,0335	<b>1,76</b>	0,0041	<b>2,09</b>	0,0003	<b>2,42</b>	0,000016
<b>0,45</b>	0,9874	<b>0,78</b>	0,5770	<b>1,11</b>	0,1700	<b>1,44</b>	0,0316	<b>1,77</b>	0,0038	<b>2,10</b>	0,0003	<b>2,43</b>	0,000014
<b>0,46</b>	0,9840	<b>0,79</b>	0,5605	<b>1,12</b>	0,1626	<b>1,45</b>	0,0298	<b>1,78</b>	0,0035	<b>2,11</b>	0,0003	<b>2,44</b>	0,000013
<b>0,47</b>	0,9800	<b>0,80</b>	0,5441	<b>1,13</b>	0,1555	<b>1,46</b>	0,0282	<b>1,79</b>	0,0033	<b>2,12</b>	0,0002	<b>2,45</b>	0,000012
<b>0,48</b>	0,9753	<b>0,81</b>	0,5280	<b>1,14</b>	0,1486	<b>1,47</b>	0,0266	<b>1,80</b>	0,0031	<b>2,13</b>	0,0002	<b>2,46</b>	0,000011
<b>0,49</b>	0,9700	<b>0,82</b>	0,5120	<b>1,15</b>	0,1420	<b>1,48</b>	0,0250	<b>1,81</b>	0,0029	<b>2,14</b>	0,0002	<b>2,47</b>	0,000010
<b>0,50</b>	0,9639	<b>0,83</b>	0,4962	<b>1,16</b>	0,1356	<b>1,49</b>	0,0236	<b>1,82</b>	0,0027	<b>2,15</b>	0,0002	<b>2,48</b>	0,000009
<b>0,51</b>	0,9572	<b>0,84</b>	0,4806	<b>1,17</b>	0,1294	<b>1,50</b>	0,0222	<b>1,83</b>	0,0025	<b>2,16</b>	0,0002	<b>2,49</b>	0,000008
<b>0,52</b>	0,9497	<b>0,85</b>	0,4653	<b>1,18</b>	0,1235	<b>1,51</b>	0,0209	<b>1,84</b>	0,0023	<b>2,17</b>	0,0002	<b>2,50</b>	0,0000075
<b>0,53</b>	0,9415	<b>0,86</b>	0,4503	<b>1,19</b>	0,1177	<b>1,52</b>	0,0197	<b>1,85</b>	0,0021	<b>2,18</b>	0,0001	<b>2,55</b>	0,0000044
<b>0,54</b>	0,9325	<b>0,87</b>	0,4355	<b>1,20</b>	0,1122	<b>1,53</b>	0,0185	<b>1,86</b>	0,0020	<b>2,19</b>	0,0001	<b>2,60</b>	0,0000026
<b>0,55</b>	0,9228	<b>0,88</b>	0,4209	<b>1,21</b>	0,1070	<b>1,54</b>	0,0174	<b>1,87</b>	0,0019	<b>2,20</b>	0,0001	<b>2,65</b>	0,0000016
<b>0,56</b>	0,9124	<b>0,89</b>	0,4067	<b>1,22</b>	0,1019	<b>1,55</b>	0,0164	<b>1,88</b>	0,0017	<b>2,21</b>	0,0001	<b>2,70</b>	0,0000010
<b>0,57</b>	0,9013	<b>0,90</b>	0,3927	<b>1,23</b>	0,0970	<b>1,56</b>	0,0154	<b>1,89</b>	0,0016	<b>2,22</b>	0,0001	<b>2,75</b>	0,0000006
<b>0,58</b>	0,8896	<b>0,91</b>	0,3791	<b>1,24</b>	0,0924	<b>1,57</b>	0,0145	<b>1,90</b>	0,0015	<b>2,23</b>	0,0001	<b>2,80</b>	0,0000003
<b>0,59</b>	0,8772	<b>0,92</b>	0,3657	<b>1,25</b>	0,0879	<b>1,58</b>	0,0136	<b>1,91</b>	0,0014	<b>2,24</b>	0,0001	<b>2,85</b>	0,00000018
<b>0,60</b>	0,8643	<b>0,93</b>	0,3527	<b>1,26</b>	0,0836	<b>1,59</b>	0,0127	<b>1,92</b>	0,0013	<b>2,25</b>	0,0001	<b>2,90</b>	0,00000010
<b>0,61</b>	0,8508	<b>0,94</b>	0,3399	<b>1,27</b>	0,0794	<b>1,60</b>	0,0120	<b>1,93</b>	0,0012	<b>2,26</b>	0,0001	<b>2,95</b>	0,00000006

Рис. 2: Значения  $C_\alpha$  при различных  $\alpha$ .

**Пример:** при уровне значимости  $\alpha = 0.05$  и пороговое значение из соотношений (??), (??)  $C_\alpha \approx 1.359$ .

**Критерий согласия Колмогорова для непрерывного равномерное распределения**

*Непрерывное равномерное распределение* - распределение случайной вещественной величины, принимающей значения, принадлежащие некоторому промежутку конечно длины, характеризующая тем, что плотность вероятности на этом промежутке почти всюду постоянна.

**Функция распределения:**

$$F_X(x) \equiv P(X \leq x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases} \quad (1.11)$$

*Значения теоретической функции распределения* для интервала  $[0, 1)$ :

$$F(x) = \frac{x - 0}{1 - 0} = x \quad (1.12)$$

*Значения эмпирической функции распределения.*

Для  $x_i$  из выборки  $X$ , значение эмпирической функции распределения:

$$F_n(x) = \frac{n_i}{n} \quad (1.13)$$

где  $n$  - количество элементов в выборке,  $n_i$  - количество элементов в выборке меньших  $x_i$ .

### 1.3 Код программы

---

```
1 import math
2
3 import matplotlib.pyplot as plt
4
5
6 def linear_congruential_generator(x, alpha, c, m):
7     while (True):
8         x = (alpha * x + c) % m
9         yield x / m
10
11
12 def multiplexial_congruential_generator(x, alpha, m):
13     generator = linear_congruential_generator(x, alpha, 0, m)
14     while (True):
15         yield next(generator)
16
17
18 def mclaren_marsaglia_generator(x_generator, y_generator, k):
19     V = [next(x_generator) for _ in range(k)]
20     while (True):
21         X = next(x_generator)
22         Y = next(y_generator)
23         j = math.floor(k * Y)
24         yield V[j]
25         V[j] = X
26
27
28 def hi_squared_test(values, k, critical_value):
29     nu = [0] * k
30     for value in values:
31         nu[math.floor(value * k)] += 1
32     p_k = len(values) / k
33     hi_squared = 0
34     for value in nu:
35         hi_squared += ((value - p_k) ** 2) / p_k
36     # Для уровня значимости 0.05 при 9-степенях свободы .
37     return hi_squared < critical_value, hi_squared
```

```

38
39
40 def kolmogorov_test(values, critical_value):
41     values.sort()
42     Dn = 0
43     i = 0
44     n = len(values)
45     for value in values:
46         i += 1
47         #  $F(X) = (x-a)/(b-a)$  = для [ a = 0 и b = 1 ] = x.
48         theoretical_func_res = value
49         # колво- значение в выборке меньше текущего значения из выборки
50         empirical_function_result = i / n
51         Dn = max(Dn, abs(theoretical_func_res - empirical_function_result))
52     Dn *= math.sqrt(n)
53     return Dn < critical_value, Dn
54
55
56 x0 = 16807
57 alpha0 = 16807
58 K = 64
59
60 x1 = 8195
61 alpha1 = 8195
62 c = 46
63 k = 64
64
65 m = 2 ** 31
66
67 hi_squared_critical_value = 16.919
68 kolmogorov_critical_value = 1.359
69
70 mult_congr_gen = multiplexial_congruential_generator(x0, alpha0, m)
71 x = [next(mult_congr_gen) for _ in range(1000)]
72 # print('\n'.join(map(str, x)))
73 hi_squa_test1 = hi_squared_test(x, 10, hi_squared_critical_value)
74 kolm_test1 = kolmogorov_test(x, kolmogorov_critical_value)
75
76 print('Multiplexial congruential generator:')
77 print('Hi Squared Pirson criteria: ' + str(hi_squa_test1[1]) + ' <= '
78       + str(hi_squared_critical_value) if hi_squa_test1[0] else
79       'Zero hypothesis fails by Hi Squared Pirson criteria.')
80 print('Kolmogorov criteria: ' + str(kolm_test1[1]) + ' <= '
81       + str(kolmogorov_critical_value) if kolm_test1[0]
82       else 'Zero hypothesis fails by Kolmogorov criteria.')
83
84 plt.hist(x, 10, ec='#993300', facecolor='#ff9900')
85 plt.title('Multiplexial congruential generator')
86 plt.show()
87
88 x = linear_congruential_generator(x0, alpha0, 0, m)
89 y = linear_congruential_generator(x1, alpha1, c, m)
90 mclar_mars_gen = mclaren_marsaglia_generator(x, y, k)
91 z = [next(mclar_mars_gen) for _ in range(1000)]
92 # print('\n'.join(map(str, z)))

```



```

93 hi_squa_test2 = hi_squared_test(z, 10, hi_squared_critical_value)
94 kolm_test2 = kolmogorov_test(z, kolmogorov_critical_value)
95
96 print('\nMcLaren marsaglia generator:')
97 print('Hi Squared Pirson criteria: ' + str(hi_squa_test2[1]) + ' <= '
98       + str(hi_squared_critical_value) if hi_squa_test2[0]
99       else 'Zero hypothesis fails by Hi Squared Pirson criteria.')
100 print('Kolmogorov criteria: ' + str(kolm_test2[1]) + ' <= '
101       + str(kolmogorov_critical_value) if kolm_test2[0]
102       else 'Zero hypothesis fails by Kolmogorov criteria.')
103
104 plt.hist(z, 10, ec='#666633', facecolor="#99ff33")
105 plt.title('McLaren marsaglia generator')
106 plt.show()

```

---

## 1.4 Результат выполнения

Multiplexial congruential generator:

Hi Squared Pirson criteria: 7.300000000000001 <= 16.919

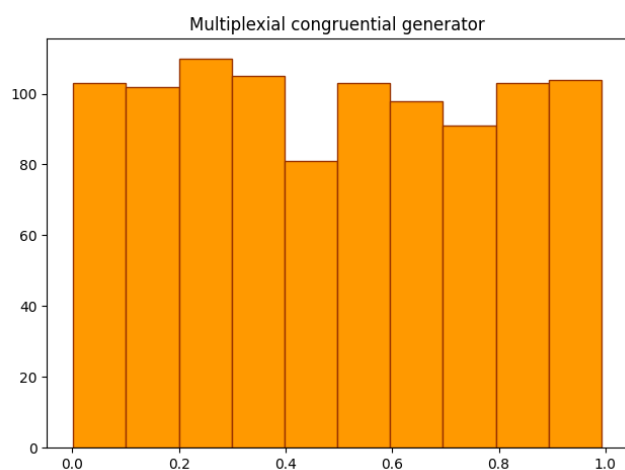
Kolmogorov criteria: 0.12043782997824995 <= 1.359

McLaren marsaglia generator:

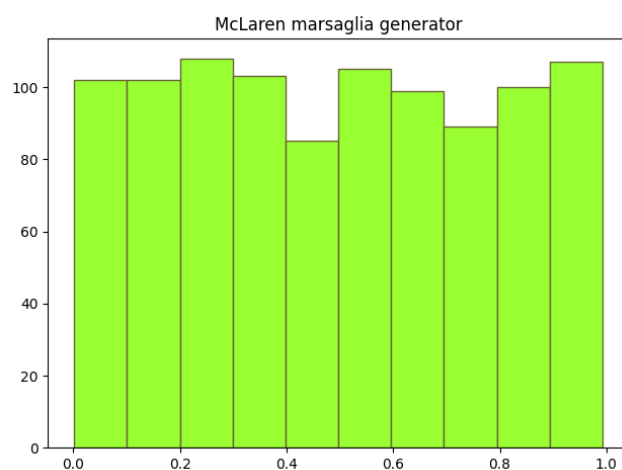
Hi Squared Pirson criteria: 4.840000000000001 <= 16.919

Kolmogorov criteria: 0.1608008491673147 <= 1.359

Рис. 3: Результат выполнения программы: проверка критерием согласия Пирсона и критерием согласия Колмогорова.



(а) Диаграмма выборки, полученной мультипликативным конгруэнтным методом.



(б) Диаграмма выборки, полученной методом Макларена-Марсальи.

## 2 Лабораторная 2

### 2.1 Условие

Согласно варианту 10:

1. Пуассона –  $P(\lambda)$ ,  $\lambda = 0.7$ ; Геометрическое –  $G(p)$ ,  $p = 0.2$ ;
2. Бернулли –  $Bi(1, p)$ ,  $p = 0.75$ ; Пуассона –  $P(\lambda)$ ,  $\lambda = 1$ ;

Смоделировать дискретную случайную величину. Исследовать точность моделирования.

1. Осуществить моделирование  $n = 1000$  реализаций СВ из заданных дискретных распределений;
2. Вывести на экран несмещённые оценки математического ожидания и дисперсии, сравнить их с истинными значениями;
3. Для каждой из случайных величин построить свой  $\chi^2$ -критерий Пирсона с уровнем значимости  $\varepsilon = 0.05$ . Проверить, что вероятность ошибки I рода стремится к 0.05;
4. Осуществить проверку каждой из сгенерированных выборок каждым из построенных критериев.

### 2.2 Теория

#### 2.2.1 Датчик случайной величины распределения Пуассона

**Распределение Пуассона** - распределение дискретного типа случайной величины, представляющей собой число событий, произошедших за фиксированное время, при условии, что данные события происходят с некоторой фиксированной средней интенсивностью и независимо друг от друга.

**Функция распределения:**

$$\frac{\Gamma(k+1, \lambda)}{k!}. \quad (2.1)$$

**Функция вероятности:**

$$\frac{e^{-\lambda} \lambda^k}{k!}. \quad (2.2)$$

**Математическое ожидание:**  $\lambda$ .

**Дисперсия:**  $\lambda$ .

## Алгоритм моделирования:

При моделировании будем использовать свойство пуассоновского процесса, состоящего в том, что время ожидания появления события имеет показательное распределение:

$$F_{\tau}(t) = 1 - e^{-\lambda t}. \quad (2.3)$$

Следовательно, последовательность наступления событий в пуассоновском процессе можно задать через *последовательность времён ожидания* этих событий. При этом надо проверять, чтобы суммарное время *суммарное время ожидания событий в очереди не превышала единицы*.

Последовательность времён ожидания можно получить методом обратных функций:

$$\tau_i = -\frac{1}{\lambda} \cdot \ln(1 - u_i), \quad (2.4)$$

где  $u_i = rnd(1)$  - *случайные числа*, т.е. значения случайной величины (СВ), равномерно распределённой на  $[0, 1]$ .

Последовательность (??) следует продолжать, пока не нарушается условие:

$$\sum_{i=1}^j \tau_i = \sum_{i=1}^j \left(-\frac{1}{\lambda} \cdot \ln(1 - u_i)\right) \leq 1. \quad (2.5)$$

Максимально возможное количество слагаемых в сумме (??) и будет равно числу появления событий в данной серии, т.е. эти числа и есть значения случайной величины, имеющей распределение Пуассона.

1. Во-первых, в (??) заменим выражение  $1 - u_i$  просто на  $u_i$ , поскольку они имеют один и тот же закон распределения;
2. Во-вторых, избавимся от операций логарифмирования в каждом слагаемом, для чего пропотенцируем выражение (??).

Таким образом, определим случайную величину:

$$\xi = \max \left\{ j : \prod_i^j u_i \geq e^{-\lambda} \right\}, \lambda > 0, \quad (2.6)$$

которая описывается распределением Пуассона. Элемент выборки можно получить последовательно увеличивая число членов ( $j$ ) в произведении до тех пор, пока не нарушится условие:

$$\prod_i^j u_i \geq e^{-\lambda}, \quad (2.7)$$

максимальное значение  $(j)$ , удовлетворяющее этому условию и есть очередное значение случайной величины.

**Программа создания выборки:**

$$D(\lambda, N) := \begin{array}{l} d \leftarrow \exp(-\lambda) \\ j \leftarrow 0 \\ \text{for } k \in 0 \dots N \\ \quad x \leftarrow \text{rnd}(1) \\ \quad \text{while } x > d \\ \quad \quad x \leftarrow x \cdot \text{rnd}(1) \\ \quad \quad j \leftarrow j + 1 \\ \quad P_k \leftarrow j \\ \quad j \leftarrow 0 \\ P \end{array}$$

Рис. 5: Псевдоалгоритм генерации СВ распределения Пуассона.

### 2.2.2 Датчик случайной величины геометрического распределения

Под **Геометрическим распределением** в теории вероятностей подразумевают одно из двух распределений дискретной случайной величины:

- распределение вероятностей случайной величины  $X$  равно номеру первого “успеха” в серии испытания Бернулли и принимающей значения  $n = 1, 2, 3, \dots$ ;
- распределение вероятностей случайной величины  $Y = X - 1$ , равно количеству “неудач” до первого “успеха” и принимающей значения  $n = 0, 1, 2, \dots$

**Функция распределения:**

$$1 - q^{n+1}. \quad (2.8)$$

**Функция вероятности:**

$$q^n p. \quad (2.9)$$

**Математическое ожидание:**

$$\frac{q}{p}. \quad (2.10)$$

**Дисперсия:**

$$\frac{q}{p^2}. \quad (2.11)$$

**Алгоритм моделирования:**

1. Моделирование реализации  $\alpha$  БСВ;

2. Принятие решения о том, что реализация  $\xi$  является значением  $x$ , определяемым соотношением:

$$x = \left\lceil \frac{\ln \alpha}{\ln q} \right\rceil, \quad (2.12)$$

где  $\lceil z \rceil$  - округление числа  $z$  в большую сторону до ближайшего целого значения.

### 2.2.3 Датчик случайной величины распределения Бернулли

**Распределение Бернулли** - дискретное распределение вероятностей, моделирующее случайный эксперимент произвольной природы, при заранее известной вероятности успеха или неудачи.:

**Функция распределения:**

$$\begin{cases} 0, k < 0 \\ q, 0 \leq k < 1 \\ 1, k \geq 1 \end{cases} . \quad (2.13)$$

**Функция вероятности:**

$$\begin{cases} q, k = 0 \\ p, k = 1 \end{cases} . \quad (2.14)$$

**Математическое ожидание:**

$$p. \quad (2.15)$$

**Дисперсия:**

$$pq. \quad (2.16)$$

**Алгоритм моделирования:**

1. Моделирование реализации  $\alpha$  БСВ;
2. Принятие решения о том, что реализация  $\xi$  является значением  $x$ , определяемым по правилу:

$$x = \begin{cases} 1, \alpha \leq p \\ 0, \alpha > p \end{cases} . \quad (2.17)$$

## 2.3 Код программы

---

```
1 import math
2 from collections import Counter
3 from functools import partial
4
5 import matplotlib.pyplot as plt
6
7
8 def linear_congruential_generator(x, alpha, c, m):
9     while True:
```

```

10     x = (alpha * x + c) % m
11     yield x / m
12
13
14 def poisson_generator(l, linear_gen):
15     d = math.exp(-l)
16     while True:
17         x = 1
18         j = 0
19         while x > d:
20             x *= next(linear_gen)
21             j += 1
22         yield j - 1
23
24
25 def geometric_generator(p, linear_gen):
26     while True:
27         yield math.floor(math.log(next(linear_gen)) / math.log(1 - p))
28
29
30 def bernoulli_generator(p, linear_gen):
31     while True:
32         yield int(next(linear_gen) <= p)
33
34
35 def hi_squared_test(values, distribution_func, critical_value):
36     distinct_map = Counter(values).most_common()
37     exampling_size = len(values)
38     hi_squared = 0
39     for pair in distinct_map:
40         empiric_freq = pair[1]
41         random_value = pair[0]
42         theoretic_freq = math.ceil(
43             exampling_size * distribution_func(random_value))
44         hi_squared += ((empiric_freq - theoretic_freq) ** 2) / theoretic_freq
45     # Для уровня значимости 0.05 при 9- степенях свободы .
46     return hi_squared < critical_value, hi_squared
47
48
49 def empirical_expectation_func(values):
50     return sum(values) / len(values)
51
52
53 def empirical_dispersion_func(values):
54     expectation = empirical_expectation_func(values)
55     result = 0
56     for value in values:
57         result += (value - expectation) ** 2
58     return result / len(values) - 1
59
60
61 def poisson_distribution_func(l, value):
62     return l ** value * math.exp(-l) / math.factorial(value)
63
64

```

```

65 def geometric_distribution_func(p, unique_x_geometric, value):
66     return (1 - p) ** unique_x_geometric.index(value) * p
67
68
69 def bernoulli_distribution_func(p, value):
70     return p if value == 1 else 1 - p
71
72
73 x0 = 79507
74 alpha0 = 79507
75 m = 2 ** 31
76
77 # POISSON SAMPLE, LAMBDA = 0.7.
78 l = 0.7
79 poisson_gen = poisson_generator(l, linear_congruential_generator(x0, alpha0,
80                                                                    0, m))
81 x_poisson = [next(poisson_gen) for _ in range(1000)]
82 # print('\n'.join(map(str, x_poisson)))
83
84 unique_x_poisson = sorted(list(Counter(x_poisson).keys()))
85 # Колво- степенейсвободыдля ( 10 варианта 6 - 1 = 5 степенейсвободы)
86 k_poisson = len(unique_x_poisson)
87 critical_value_poisson = 11.07
88 hi_squa_test1 = hi_squared_test(x_poisson,
89                                 partial(poisson_distribution_func, l),
90                                 critical_value_poisson)
91 print('Poisson generator, lambda = 0.7:')
92 print('Hi Squared Pirson criteria: ' + str(hi_squa_test1[1]) + ' <= '
93       + str(critical_value_poisson) if hi_squa_test1[0] else
94       'Zero hypothesis fails by Hi Squared Pirson criteria.')
95
96 theoretical_expectation = l
97 empirical_dispersion = empirical_dispersion_func(x_poisson)
98 theoretical_dispersion = l
99 empirical_expectation = empirical_expectation_func(x_poisson)
100 print('theoretical expectation: ', theoretical_expectation)
101 print('empirical expectation: ', empirical_expectation)
102 print('theoretical dispersion: ', theoretical_dispersion)
103 print('empirical dispersion: ', empirical_dispersion)
104 print('')
105
106 unique_x_poisson.append(unique_x_poisson[k_poisson - 1] + 1)
107 plt.hist(x_poisson, bins=unique_x_poisson, ec='#666633',
108         facecolor="#99ff33")
109 plt.title('Poisson generator, $\lambda = 0.7$')
110 plt.show()
111
112 # POISSON SAMPLE, LAMBDA = 1
113 l = 1
114 poisson_gen = poisson_generator(l, linear_congruential_generator(x0, alpha0,
115                                                                    0, m))
116 x_poisson = [next(poisson_gen) for _ in range(1000)]
117 # print('\n'.join(map(str, x_poisson)))
118
119 unique_x_poisson = sorted(list(Counter(x_poisson).keys()))

```

```

120 # Колво- степенейсвободыдля ( 10 варианта 6 - 1 = 5 степенейсвободы)
121 k_poisson = len(unique_x_poisson)
122 critical_value_poisson = 11.07
123 hi_squa_test2 = hi_squared_test(x_poisson,
124                                 partial(poisson_distribution_func, 1),
125                                 critical_value_poisson)
126 print('Poisson generator, lambda = 1:')
127 print('Hi Squared Pirson criteria: ' + str(hi_squa_test2[1]) + ' <= '
128       + str(critical_value_poisson) if hi_squa_test2[0] else
129       'Zero hypothesis fails by Hi Squared Pirson criteria.')
130
131 theoretical_expectation = 1
132 empirical_dispersion = empirical_dispersion_func(x_poisson)
133 theoretical_dispersion = 1
134 empirical_expectation = empirical_expectation_func(x_poisson)
135 print('theoretical expectation: ', theoretical_expectation)
136 print('empirical expectation: ', empirical_expectation)
137 print('theoretical dispersion: ', theoretical_dispersion)
138 print('empirical dispersion: ', empirical_dispersion)
139 print('')
140
141 unique_x_poisson.append(unique_x_poisson[k_poisson - 1] + 1)
142 plt.hist(x_poisson, bins=unique_x_poisson, ec='#666633',
143          facecolor="#99ff33")
144 plt.title('Poisson generator, $\lambda = 1$')
145 plt.show()
146
147 # GEOMETRIC SAMPLE.
148 p = 0.2
149 geometric_gen = geometric_generator(p, linear_congruential_generator(x0, alpha0,
150                               0, m))
151 x_geometric = [next(geometric_gen) for _ in range(1000)]
152 # print('\n'.join(map(str, x_geometric)))
153
154 unique_x_geometric = sorted(list(Counter(x_geometric).keys()))
155 # Колво- степенейсвободыдля ( 10 варианта 27 - 1 = 26 степенейсвободы)
156 k_geometric = len(unique_x_geometric)
157 critical_value_geometric = 38.89
158 hi_squa_test3 = hi_squared_test(x_geometric,
159                                 partial(geometric_distribution_func, p,
160                                 unique_x_geometric),
161                                 critical_value_geometric)
162 print('Geometric generator, p = 1:')
163 print('Hi Squared Pirson criteria: ' + str(hi_squa_test3[1]) + ' <= '
164       + str(critical_value_geometric) if hi_squa_test3[0] else
165       'Zero hypothesis fails by Hi Squared Pirson criteria.')
166 theoretical_expectation = 1 / p
167 empirical_dispersion = empirical_dispersion_func(x_geometric)
168 theoretical_dispersion = (1 - p) / p ** 2
169 empirical_expectation = empirical_expectation_func(x_geometric)
170 print('theoretical expectation: ', theoretical_expectation)
171 print('empirical expectation: ', empirical_expectation)
172 print('theoretical dispersion: ', theoretical_dispersion)
173 print('empirical dispersion: ', empirical_dispersion)
174 print('')

```



```

175
176 unique_x_geometric.append(unique_x_geometric[k_geometric - 1] + 1)
177 plt.hist(x_geometric, bins=sorted(list(unique_x_geometric)), ec='#666633',
178         facecolor="#99ff33")
179 plt.title('Geometric generator, $p = 0.2$')
180 plt.show()
181
182 # BERNOULLI SAMPLE.
183 p = 0.75
184 bernoulli_gen = bernoulli_generator(p, linear_congruential_generator(x0, alpha0,
185                             0, m))
186 x_bernoulli = [next(bernoulli_gen) for _ in range(1000)]
187 # print('\n'.join(map(str, x_bernoulli)))
188
189 critical_x_bernoulli = 10
190 unique_x_bernoulli = sorted(list(Counter(x_bernoulli).keys()))
191 # Колво- степенейсвободыдля ( 10 варианта 2 - 1 = 1 степенейсвободы)
192 k_bernoulli = len(unique_x_bernoulli)
193 critical_value_bernoulli = 3.841
194 hi_squa_test4 = hi_squared_test(x_bernoulli,
195                                 partial(bernoulli_distribution_func, p),
196                                 critical_value_bernoulli)
197 print('Geometric generator, p = 1:')
198 print('Hi Squared Pirson criteria: ' + str(hi_squa_test4[1]) + ' <= '
199       + str(critical_value_bernoulli) if hi_squa_test4[0] else
200       'Zero hypothesis fails by Hi Squared Pirson criteria.')
201
202 theoretical_expectation = p
203 empirical_dispersion = empirical_dispersion_func(x_poisson)
204 theoretical_dispersion = p * (1 - p)
205 empirical_expectation = empirical_expectation_func(x_poisson)
206 print('theoretical expectation: ', theoretical_expectation)
207 print('empirical expectation: ', empirical_expectation)
208 print('theoretical dispersion: ', theoretical_dispersion)
209 print('empirical dispersion: ', empirical_dispersion)
210
211 unique_x_bernoulli.append(unique_x_bernoulli[k_bernoulli - 1] + 1)
212 plt.hist(x_bernoulli, bins=unique_x_bernoulli, ec='#666633',
213         facecolor="#99ff33")
214 plt.title('Bernoulli generator, $p = 0.75$')
215 plt.show()

```

---

## 2.4 Результат выполнения

Poisson generator, lambda = 0.7:

Hi Squared Pirson criteria: 1.4769448422208398 <= 11.07  
theoretical expectation: 0.7  
empirical expectation: 0.71  
theoretical dispersion: 0.7  
empirical dispersion: -0.31010000000000315

Poisson generator, lambda = 1:

Hi Squared Pirson criteria: 3.8721949509116405 <= 11.07  
theoretical expectation: 1  
empirical expectation: 1.021  
theoretical dispersion: 1  
empirical dispersion: 0.008559000000012418

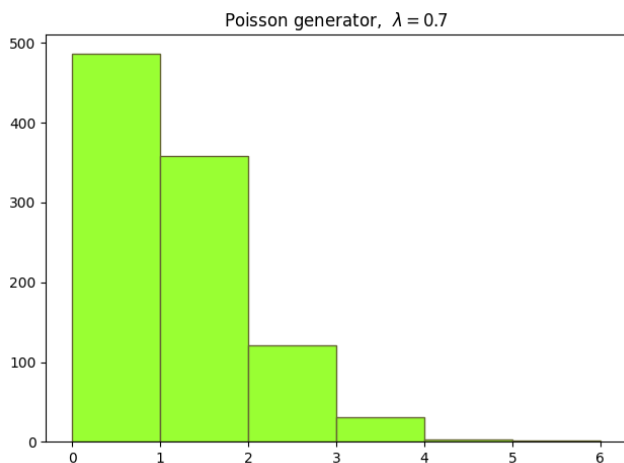
Geometric generator, p = 1:

Hi Squared Pirson criteria: 20.886693627653305 <= 38.89  
theoretical expectation: 5.0  
empirical expectation: 4.121  
theoretical dispersion: 19.999999999999996  
empirical dispersion: 20.328358999999953

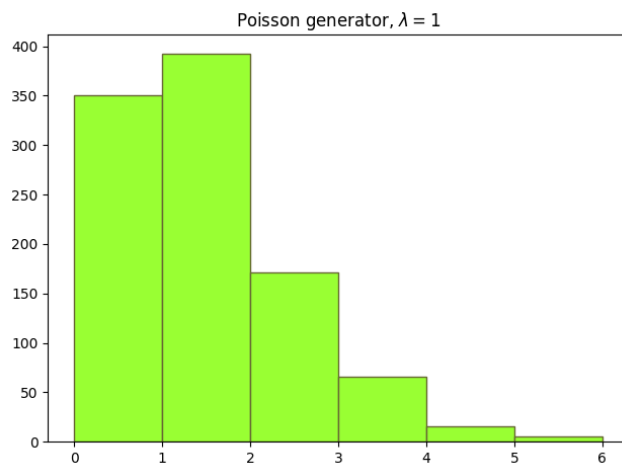
Geometric generator, p = 1:

Hi Squared Pirson criteria: 0.08533333333333333 <= 3.841  
theoretical expectation: 0.75  
empirical expectation: 1.021  
theoretical dispersion: 0.1875  
empirical dispersion: 0.008559000000012418

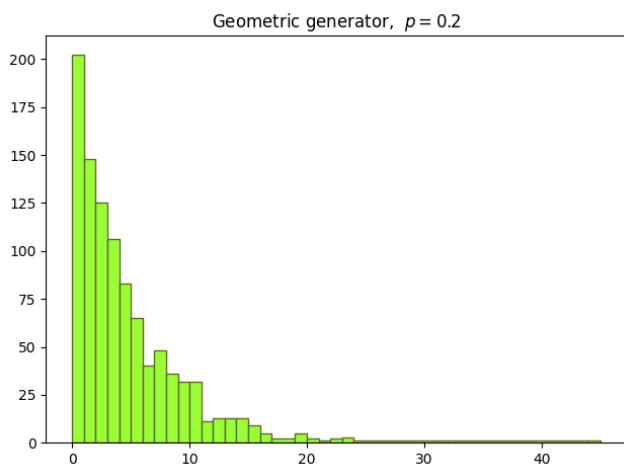
Рис. 6: Результат выполнения программы: проверка критерием согласия Пирсона и подсчёт несмещённых оценок математического ожидания и дисперсии.



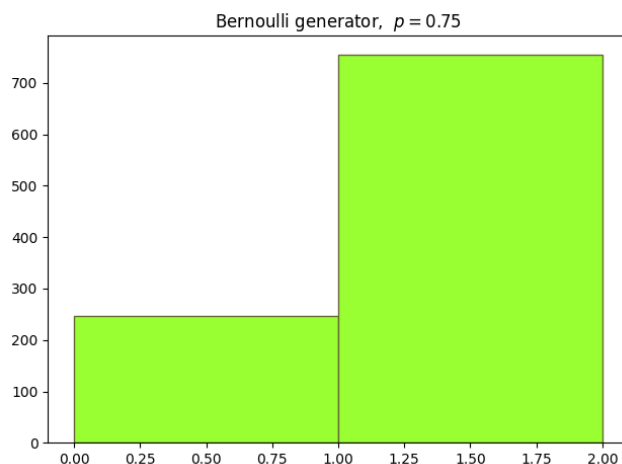
(a) Диаграмма выборки, полученной генератором распределения Пуассона при  $\lambda = 0.7$ .



(b) Диаграмма выборки, полученной генератором распределения Пуассона при  $\lambda = 1$ .



(c) Диаграмма выборки, полученной генератором геометрического распределения.



(d) Диаграмма выборки, полученной генератором распределения Бернулли.

## 3 Лабораторная 3

### 3.1 Условие

**Согласно варианту 10:**

1. Нормальное  $N(m, s^2)$ ,  $m = 1, s^2 = 9$ ; Логонормальное  $LN(m, s^2)$ ,  $m = 1, s^2 = 9$ ; Экспоненциальное  $E(\alpha)$ ,  $\alpha = 2$ ;
2. Нормальное  $N(m, s^2)$ ,  $m = 0, s^2 = 1$ ; Лапласа  $L(\alpha)$ ,  $\alpha = 0.5$ ; Вейбула  $W(a, b)$ ,  $a = 1, b = 0.5$ ;

Смоделировать непрерывную случайную величину. Исследовать точность моделирования.

1. Осуществить моделирование  $n = 1000$  реализаций СВ из нормального закона распределения  $N(m, s^2)$  с заданными параметрами. Вычислить несмещённые оценки математического ожидания и дисперсии, сравнить их с истинными;
2. Смоделировать  $n = 1000$  СВ из заданных абсолютно непрерывных распределений. Вычислить несмещённые оценки математического ожидания и дисперсии, сравнить их с истинными значениями (если это возможно);
3. Для каждой из случайных величин построить свой критерий Колмогорова с уровнем значимости  $\varepsilon = 0.05$ . Проверить, что вероятность ошибки I рода стремится к 0.05;
4. Для каждой из случайных величин построить свой  $\chi^2$ -критерий Пирсона с уровнем значимости  $\varepsilon = 0.05$ . Проверить, что вероятность ошибки I рода стремится к 0.05;
5. Осуществить проверку каждой из сгенерированных выборок каждым из построенных критериев.

### 3.2 Теория

#### 3.2.1 Датчик случайной величины нормального распределения

**Нормальное распределение** - распределение вероятностей, которое в одномерном случае задаётся функцией плотности вероятности, совпадающей с функцией Гаусса.

**Функция распределения:**

$$\frac{1}{2}(1 + \operatorname{erf} \left[ \frac{x - \mu}{\sqrt{2\sigma^2}} \right]). \quad (3.1)$$

**Функция плотности:**

$$\frac{1}{\sigma\sqrt{2\pi}} \cdot \exp \left[ -\frac{(x - \mu)^2}{2\sigma^2} \right]. \quad (3.2)$$

**Математическое ожидание:**  $\mu$ .

**Дисперсия:**  $\sigma^2$ .

**Алгоритм моделирования:**

При моделировании будем использовать **преобразование Бокса-Мюллера**. Пусть  $r$  и  $\phi$  - независимые случайные величины, равномерно распределённые на интервале  $(0, 1]$ . Вычислим  $z_0, z_1$  по формулам:

$$\begin{aligned} z_0 &= \cos(2\pi\phi) \sqrt{-2 \ln(r)}, \\ z_1 &= \sin(2\pi\phi) \sqrt{-2 \ln(r)}. \end{aligned} \quad (3.3)$$

Тогда  $z_0, z_1$  будут независимы и распределены нормально с математическим ожиданием 0 и дисперсией 1.

### 3.2.2 Датчик случайной величины логнормального распределения

**Логнормальное распределение** - это двухпараметрическое семейство абсолютно непрерывных распределений. Если СВ имеет логнормальное распределение, то её логарифм имеет нормальное распределение.

**Функция распределения:**

$$\frac{1}{2} + \frac{1}{2} \operatorname{Erf} \left[ \frac{\ln(x) - \mu}{\sigma \sqrt{2}} \right]. \quad (3.4)$$

**Функция плотности:**

$$\exp \left[ -\frac{\left( \frac{\ln(x) - \mu}{\sigma} \right)^2}{2} \right] / (x \sigma \sqrt{2\pi}). \quad (3.5)$$

**Математическое ожидание:**  $e^{\mu + \sigma^2/2}$ .

**Дисперсия:**  $(e^{\sigma^2} - 1) \cdot e^{2\mu + \sigma^2}$ .

**Алгоритм моделирования:**

Для моделирования обычно используется связь с нормальным распределением. Поэтому, достаточно сгенерировать нормально распределённую СВ, например, используя **преобразование Бокса-Мюллера**, и вычислить её экспонент.

### 3.2.3 Датчик случайной величины экспоненциального распределения

**Экспоненциальное распределение** - абсолютно непрерывное распределение, моделирующее время между двумя последовательными свершениями одного и того же события.

**Функция распределения:**

$$1 - e^{-\lambda x}. \quad (3.6)$$

**Функция плотности:**

$$\lambda e^{-\lambda x}. \quad (3.7)$$

**Математическое ожидание:**

$$\lambda^{-1}. \quad (3.8)$$

**Дисперсия:**

$$\lambda^{-2}. \quad (3.9)$$

**Алгоритм моделирования:**

1. Моделирование реализации  $\alpha$  БСВ;
2. Вычисление экспоненциально распределённой СВ:

$$x = -\frac{1}{\lambda} \ln(\alpha) \quad (3.10)$$

### 3.2.4 Датчик случайной величины распределения Лапласа

**Распределение Лапласа** - в теории вероятностей это непрерывное распределение случайной величины, при котором плотность вероятности есть  $f(x) = \frac{\alpha}{2} \cdot e^{-\alpha|x-\beta|}$ ,  $-\infty < x < +\infty$ , где  $\alpha > 0$  - параметр масштаба,  $-\infty < \beta < +\infty$  - параметр сдвига.

**Функция распределения:**

$$\begin{cases} \frac{1}{2}e^{\alpha(x-\beta)} \\ 1 - \frac{1}{2}e^{\alpha(x-\beta)} \end{cases} \quad (3.11)$$

**Функция плотности:**

$$\frac{\alpha}{2} e^{-\alpha|x-\beta|}. \quad (3.12)$$

**Математическое ожидание:**

$$\beta. \quad (3.13)$$

**Дисперсия:**

$$\frac{2}{\alpha^2}. \quad (3.14)$$

**Алгоритм моделирования:**

Алгоритм моделирования  $\xi \sim L(\lambda)$  основан на методе обратной функции. Обратная для функции распределения  $F_\xi(x)$  функция имеет вид:

$$\begin{cases} x = F_\xi^{-1}(y) = \frac{1}{\lambda} \ln(2y) < 0, y \in [0, 0.5) \\ x = F_\xi^{-1}(y) = \frac{1}{\lambda} \ln(2(1-y)) < 0, y \in [0.5, 1) \end{cases} \quad (3.15)$$

Для моделирования реализация  $x$  СВ  $\xi \sim L(\lambda)$  выполняются следующие действия:

1. Моделирование реализации  $\alpha$  БСВ;
2. Принимается решение о том, что реализацией СВ  $\xi$  является величина  $x$ , вычисляемая по формулам (1.15) согласно отрезку, которому принадлежит  $y$ .

**3.2.5 Датчик случайной величины распределения Вейбула**

**Распределение Вейбула** - в теории вероятностей это двухпараметрическое семейство абсолютно непрерывных распределений.

**Функция распределения:**

$$1 - e^{-(x/\lambda)^k}. \quad (3.16)$$

**Функция плотности:**

$$\frac{k}{\lambda} \cdot \left(\frac{x}{\lambda}\right)^{k-1} \cdot e^{-(x/\lambda)^k}. \quad (3.17)$$

**Математическое ожидание:**

$$\lambda \Gamma\left(1 + \frac{1}{k}\right). \quad (3.18)$$

**Дисперсия:**

$$\lambda^2 \Gamma\left(1 + \frac{2}{k}\right) - \mu^2. \quad (3.19)$$

**Алгоритм моделирования:**

Алгоритм моделирования  $\xi \sim WG(\lambda, k)$  основан на методе обратной функции. Обратная для функции распределения  $F_\xi(x)$  функция имеет вид:

$$x = F_\xi^{-1}(y) = \left(-\frac{1}{\lambda} \ln(y)\right)^{1/k}. \quad (3.20)$$

Для моделирования реализация  $x$  СВ  $\xi \sim L(\lambda)$  выполняются следующие действия:

1. Моделирование реализации  $\alpha$  БСВ;
2. Принимается решение о том, что реализацией СВ  $\xi$  является величина  $x$ , вычисляемая по формулам (1.20), где  $y = \alpha$ .

### 3.3 Код программы

---

```
1 import math
2 import random
3 from functools import partial
4
5 import matplotlib.pyplot as plt
6 from scipy.stats import norm, chi2, kstwobign, lognorm, expon, laplace, \
7     weibull_min
8
9
10 def linear_congruential_generator(x, alpha, c, m):
11     while True:
12         x = (alpha * x + c) % m
13         yield x / m
14
15
16 def normal_generator(mu, sigma, linear_gen):
17     while True:
18         u1 = next(linear_gen)
19         u2 = next(linear_gen)
20         z0 = math.sqrt(-2.0 * math.log(u1)) * math.cos(2 * math.pi * u2)
21         # z1 = math.sqrt(-2.0 * math.log(u1)) * math.sin(2 * math.pi * u2)
22         yield mu + z0 * sigma
23
24
25 def exponential_generator(l, linear_gen):
26     while True:
27         yield -1 / l * math.log(next(linear_gen))
28
29
30 def lognormal_generator(mu, sigma, linear_gen):
31     normal_gen = normal_generator(mu, sigma, linear_gen)
32     while True:
33         yield math.exp(next(normal_gen))
34
35
36 def laplace_generator(alpha, linear_gen):
37     while True:
38         y = next(linear_gen)
39         if 0 <= y < 0.5:
40             yield 1 / alpha * math.log(2 * y)
41         else:
42             yield -1 / alpha * math.log(2 * (1 - y))
43
44
45 def weibull_generator(l, k, linear_gen):
46     while True:
47         yield l * ((-math.log(next(linear_gen))) ** (1 / k))
48
49
50 def hi_squared_test(frequencies, borders, distribution_func, p_value):
51     exampling_size = sum(frequencies)
52     hi_squared = 0
```

```

53     for i in range(1, len(frequencies) + 1):
54         empiric_freq = frequencies[i - 1]
55         theoretic_freq = (distribution_func(borders[i]) - distribution_func(
56             borders[i - 1])) * exempling_size
57         if theoretic_freq:
58             hi_squared += ((
59                 empiric_freq - theoretic_freq) ** 2) / theoretic_freq
60     degrees_of_freedom = len(frequencies) - 1
61     critical_value = chi2.ppf(1 - p_value, degrees_of_freedom)
62     return hi_squared < critical_value, hi_squared, critical_value
63
64
65 def kolmogorov_test(values, distribution_func, p_value):
66     values.sort()
67     Dn = 0
68     i = 0
69     n = len(values)
70     for value in values:
71         i += 1
72         theoretical_func_res = distribution_func(value)
73         empirical_function_res = i / n
74         Dn = max(Dn, abs(theoretical_func_res - empirical_function_res))
75     Dn *= math.sqrt(n)
76     critical_value = kstwobign.ppf(1 - p_value)
77     return Dn < critical_value, Dn, critical_value
78
79
80 def empirical_expectation_func(values):
81     return sum(values) / len(values)
82
83
84 def empirical_dispersion_func(values):
85     expectation = empirical_expectation_func(values)
86     result = 0
87     for value in values:
88         result += (value - expectation) ** 2
89     return result / len(values) - 1
90
91
92 def cumulative_norm_distrib_func(mu, sigma, value):
93     return norm.cdf(value, mu, sigma)
94
95
96 def cumulative_lognorm_distrib_func(mu, sigma, value):
97     return lognorm.cdf(value, scale=math.exp(mu), s=sigma)
98
99
100 def cumulative_exponential_distrib_func(l, value):
101     return expon.cdf(value, scale=1 / l)
102
103
104 def cumulative_laplace_distrib_func(alpha, betta, value):
105     return laplace.cdf(value, scale=1 / alpha, loc=betta)
106
107

```



```

108 def cumulative_weibull_distrib_func(l, k, value):
109     return weibull_min.cdf(value, k, scale=1)
110
111
112 def built_in_random():
113     while True:
114         yield random.random()
115
116
117 x0 = 79507
118 alpha0 = 79507
119 c = 63
120 m = 2 ** 31
121 p_value = 0.05
122
123 generator = linear_congruential_generator(x0, alpha0, c, m)
124
125 # NORMAL SAMPLE, MU = 1, SIGMA^2 = 9
126 mu = 1
127 sigma = 3
128
129 normal_gen = normal_generator(mu, sigma, generator)
130 x_normal = [next(normal_gen) for _ in range(1000)]
131 # print('\n'.join(map(str, x_normal)))
132
133 freq_normal, borders_normal, _ = plt.hist(x_normal, bins='auto',
134                                             ec='#666633',
135                                             facecolor="#99ff33")
136 plt.title('Normal generator, $\mu = 1, \sigma^2=9$')
137 plt.show()
138
139 hi_squa_test1 = hi_squared_test(freq_normal,
140                                borders_normal,
141                                partial(cumulative_norm_distrib_func,
142                                        mu, sigma),
143                                p_value)
144 print('Normal generator, mu = 1, sigma^2 = 9:')
145 print('Hi Squared Pirson criteria: ' + str(hi_squa_test1[1]) + ' <= '
146       + str(hi_squa_test1[2]) if hi_squa_test1[0] else
147       'Zero hypothesis fails by Hi Squared Pirson criteria.')
148 kolm_test1 = kolmogorov_test(x_normal,
149                              partial(cumulative_norm_distrib_func, mu, sigma),
150                              p_value)
151 print('Kolmogorov criteria: ' + str(kolm_test1[1]) + ' <= '
152       + str(kolm_test1[2]) if kolm_test1[0]
153       else 'Zero hypothesis fails by Kolmogorov criteria.')
154
155 theoretical_expectation = mu
156 empirical_dispersion = empirical_dispersion_func(x_normal)
157 theoretical_dispersion = sigma ** 2
158 empirical_expectation = empirical_expectation_func(x_normal)
159 print('theoretical expectation: ', theoretical_expectation)
160 print('empirical expectation: ', empirical_expectation)
161 print('theoretical dispersion: ', theoretical_dispersion)
162 print('empirical dispersion: ', empirical_dispersion)

```

```

163 print('')
164
165 # NORMAL SAMPLE, MU = 0, SIGMA^2 = 1
166 mu = 0
167 sigma = 1
168
169 normal_gen = normal_generator(mu, sigma, generator)
170 x_normal = [next(normal_gen) for _ in range(1000)]
171 # print('\n'.join(map(str, x_normal)))
172
173 freq_normal, borders_normal, _ = plt.hist(x_normal, bins='auto',
174                                           ec='#666633',
175                                           facecolor="#99ff33")
176 plt.title('Normal generator, $\mu = 0, \sigma^2=1$')
177 plt.show()
178
179 hi_squa_test2 = hi_squared_test(freq_normal,
180                                borders_normal,
181                                partial(cumulative_norm_distrib_func,
182                                       mu, sigma),
183                                p_value)
184 print('Normal generator, mu = 0, sigma^2 = 1:')
185 print('Hi Squared Pirson criteria: ' + str(hi_squa_test2[1]) + ' <= '
186       + str(hi_squa_test2[2]) if hi_squa_test2[0] else
187       'Zero hypothesis fails by Hi Squared Pirson criteria.')
188 kolm_test2 = kolmogorov_test(x_normal,
189                             partial(cumulative_norm_distrib_func, mu, sigma),
190                             p_value)
191 print('Kolmogorov criteria: ' + str(kolm_test2[1]) + ' <= '
192       + str(kolm_test2[2]) if kolm_test2[0]
193       else 'Zero hypothesis fails by Kolmogorov criteria.')
194
195 theoretical_expectation = mu
196 empirical_dispersion = empirical_dispersion_func(x_normal)
197 theoretical_dispersion = sigma ** 2
198 empirical_expectation = empirical_expectation_func(x_normal)
199 print('theoretical expectation: ', theoretical_expectation)
200 print('empirical expectation: ', empirical_expectation)
201 print('theoretical dispersion: ', theoretical_dispersion)
202 print('empirical dispersion: ', empirical_dispersion)
203 print('')
204
205 # LOGNORMAL SAMPLE, MU = 1, SIGMA^2 = 9
206 mu = 1
207 sigma = 3
208
209 lognormal_gen = lognormal_generator(mu, sigma, generator)
210 x_lognormal = [next(lognormal_gen) for _ in range(1000)]
211 # print('\n'.join(map(str, x_lognormal)))
212
213 freq_lognormal, borders_lognormal, _ = plt.hist(x_lognormal, bins='auto',
214                                                  ec='#666633',
215                                                  facecolor="#99ff33")
216 plt.title('Lognormal generator, $\mu = 1, \sigma^2=9$')
217 plt.show()

```

```

218
219 hi_squa_test3 = hi_squared_test(freq_lognormal,
220                                 borders_lognormal,
221                                 partial(cumulative_norm_distrib_func,
222                                         mu, sigma),
223                                 p_value)
224
225 print('Lognormal generator, mu = 1, sigma^2 = 9:')
226 print('Hi Squared Pirson criteria: ' + str(hi_squa_test3[1]) + ' <= '
227       + str(hi_squa_test3[2]) if hi_squa_test3[0] else
228       'Zero hypothesis fails by Hi Squared Pirson criteria.')
229 kolm_test3 = kolmogorov_test(x_lognormal,
230                              partial(cumulative_lognorm_distrib_func, mu,
231                                      sigma),
232                              p_value)
233 print('Kolmogorov criteria: ' + str(kolm_test3[1]) + ' <= '
234       + str(kolm_test3[2]) if kolm_test3[0]
235       else 'Zero hypothesis fails by Kolmogorov criteria.')
236
237 theoretical_expectation = math.exp(mu + sigma ** 2 / 2)
238 empirical_dispersion = empirical_dispersion_func(x_lognormal)
239 theoretical_dispersion = (math.exp(sigma ** 2) - 1) * math.exp(
240     2 * mu + sigma ** 2)
241 empirical_expectation = empirical_expectation_func(x_lognormal)
242 print('theoretical expectation: ', theoretical_expectation)
243 print('empirical expectation: ', empirical_expectation)
244 print('theoretical dispersion: ', theoretical_dispersion)
245 print('empirical dispersion: ', empirical_dispersion)
246 print('')
247
248 # EXPONENTIAL SAMPLE, LAMBDA = 2
249 l = 2
250
251 exponential_gen = exponential_generator(l, generator)
252 x_exponential = [next(exponential_gen) for _ in range(1000)]
253 # print('\n'.join(map(str, x_exponential)))
254
255 freq_exponential, borders_exponential, _ = plt.hist(x_exponential, bins='auto',
256                                                     ec='#666633',
257                                                     facecolor="#99ff33")
258 plt.title('Exponential generator, $\lambda=2$')
259 plt.show()
260
261 hi_squa_test4 = hi_squared_test(freq_exponential,
262                                 borders_exponential,
263                                 partial(cumulative_exponential_distrib_func, l),
264                                 p_value)
265
266 print('Exponential generator, lambda = 2:')
267 print('Hi Squared Pirson criteria: ' + str(hi_squa_test4[1]) + ' <= '
268       + str(hi_squa_test4[2]) if hi_squa_test4[0] else
269       'Zero hypothesis fails by Hi Squared Pirson criteria.')
270 kolm_test4 = kolmogorov_test(x_exponential,
271                              partial(cumulative_exponential_distrib_func, l),
272                              p_value)

```

```

273 print('Kolmogorov criteria: ' + str(kolm_test4[1]) + ' <= '
274       + str(kolm_test4[2]) if kolm_test4[0]
275       else 'Zero hypothesis fails by Kolmogorov criteria.')
276
277 theoretical_expectation = 1 / 1
278 empirical_dispersion = empirical_dispersion_func(x_exponential)
279 theoretical_dispersion = 1 / 1 ** 2
280 empirical_expectation = empirical_expectation_func(x_exponential)
281 print('theoretical expectation: ', theoretical_expectation)
282 print('empirical expectation: ', empirical_expectation)
283 print('theoretical dispersion: ', theoretical_dispersion)
284 print('empirical dispersion: ', empirical_dispersion)
285 print('')
286
287 # LAPLACE SAMPLE, ALPHA = 0.5, BETTA = 0
288 alpha = 0.5
289 beta = 0
290
291 laplace_gen = laplace_generator(alpha, generator)
292 x_laplace = [next(laplace_gen) for _ in range(1000)]
293 # print('\n'.join(map(str, x_laplace)))
294
295 freq_laplace, borders_laplace, _ = plt.hist(x_laplace, bins='auto',
296                                             ec='#666633',
297                                             facecolor="#99ff33")
298 plt.title(r'Laplace generator, $\alpha = 0.5, \beta = 0$')
299 plt.show()
300
301 hi_squa_test5 = hi_squared_test(freq_laplace,
302                                borders_laplace,
303                                partial(cumulative_laplace_distrib_func, alpha,
304                                       beta),
305                                p_value)
306
307 print('Laplace generator, alpha = 0.5, beta = 0:')
308 print('Hi Squared Pirson criteria: ' + str(hi_squa_test5[1]) + ' <= '
309       + str(hi_squa_test5[2]) if hi_squa_test5[0] else
310       'Zero hypothesis fails by Hi Squared Pirson criteria.')
311 kolm_test5 = kolmogorov_test(x_laplace,
312                              partial(cumulative_laplace_distrib_func, alpha,
313                                     beta),
314                              p_value)
315 print('Kolmogorov criteria: ' + str(kolm_test5[1]) + ' <= '
316       + str(kolm_test5[2]) if kolm_test5[0]
317       else 'Zero hypothesis fails by Kolmogorov criteria.')
318
319 theoretical_expectation = beta
320 empirical_dispersion = empirical_dispersion_func(x_laplace)
321 theoretical_dispersion = 2 / alpha ** 2
322 empirical_expectation = empirical_expectation_func(x_laplace)
323 print('theoretical expectation: ', theoretical_expectation)
324 print('empirical expectation: ', empirical_expectation)
325 print('theoretical dispersion: ', theoretical_dispersion)
326 print('empirical dispersion: ', empirical_dispersion)
327 print('')

```

```

328
329 # WEIBULL SAMPLE, LAMBDA = 1, K = 0.5
330 l = 1
331 k = 0.5
332
333 weibull_gen = weibull_generator(l, k, generator)
334 x_weibull = [next(weibull_gen) for _ in range(1000)]
335 # print('\n'.join(map(str, x_weibull)))
336
337 freq_weibull, borders_weibull, _ = plt.hist(x_weibull, bins='auto',
338                                             ec='#666633',
339                                             facecolor="#99ff33")
340 plt.title(r'Weibull generator, $\lambda = 1, k = 0.5$')
341 plt.show()
342
343 hi_squa_test = hi_squared_test(freq_weibull,
344                               borders_weibull,
345                               partial(cumulative_weibull_distrib_func, l, k),
346                               p_value)
347
348 print('Weibull generator, lambda = 1, k = 0.5:')
349 print('Hi Squared Pirson criteria: ' + str(hi_squa_test[1]) + ' <= '
350       + str(hi_squa_test[2]) if hi_squa_test[0] else
351       'Zero hypothesis fails by Hi Squared Pirson criteria.')
352 kolm_test = kolmogorov_test(x_weibull,
353                             partial(cumulative_weibull_distrib_func, l, k),
354                             p_value)
355 print('Kolmogorov criteria: ' + str(kolm_test[1]) + ' <= '
356       + str(kolm_test[2]) if kolm_test[0]
357       else 'Zero hypothesis fails by Kolmogorov criteria.')
358
359 theoretical_expectation = weibull_min.mean(k, scale=1)
360 empirical_dispersion = empirical_dispersion_func(x_weibull)
361 theoretical_dispersion = weibull_min.var(k, scale=1)
362 empirical_expectation = empirical_expectation_func(x_weibull)
363 print('theoretical expectation: ', theoretical_expectation)
364 print('empirical expectation: ', empirical_expectation)
365 print('theoretical dispersion: ', theoretical_dispersion)
366 print('empirical dispersion: ', empirical_dispersion)
367 print('')

```

---

### 3.4 Результат выполнения

```
Normal generator, mu = 1, sigma^2 = 9:
Hi Squared Pirson criteria: 28.358119998727172 <= 33.92443847144381
Kolmogorov criteria: 0.9679439050529556 <= 1.3580986393225505
theoretical expectation: 1
empirical expectation: 0.9294316472943027
theoretical dispersion: 9
empirical dispersion: 7.4959786131342

Normal generator, mu = 0, sigma^2 = 1:
Hi Squared Pirson criteria: 21.488365965347455 <= 33.92443847144381
Kolmogorov criteria: 0.729460343122114 <= 1.3580986393225505
theoretical expectation: 0
empirical expectation: -0.0021933952374124295
theoretical dispersion: 1
empirical dispersion: -0.0879810652662405

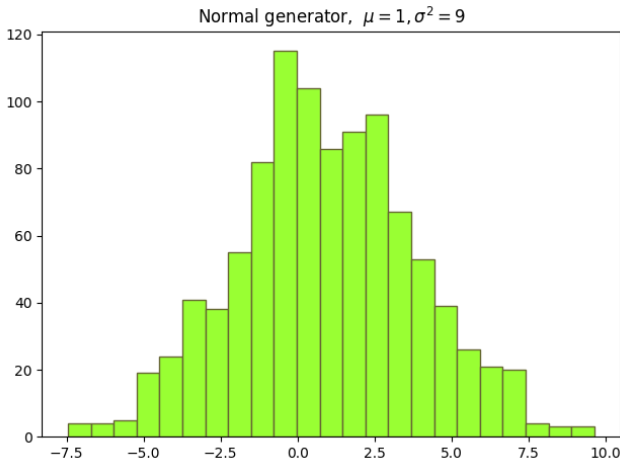
Lognormal generator, mu = 1, sigma^2 = 9:
Zero hypothesis fails by Hi Squared Pirson criteria.
Kolmogorov criteria: 0.45996235601524604 <= 1.3580986393225505
theoretical expectation: 244.69193226422038
empirical expectation: 214.6402940382709
theoretical dispersion: 485105321.26807505
empirical dispersion: 6708849.993968649

Exponential generator, lambda = 2:
Zero hypothesis fails by Hi Squared Pirson criteria.
Kolmogorov criteria: 0.865617143544657 <= 1.3580986393225505
theoretical expectation: 0.5
empirical expectation: 0.4964390006492622
theoretical dispersion: 0.25
empirical dispersion: -0.7272060875253951

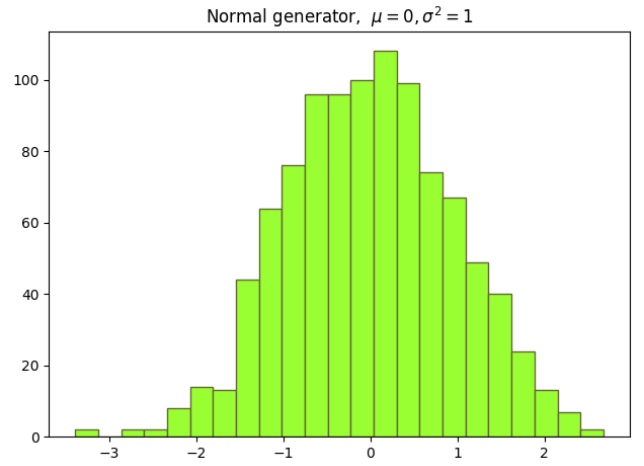
Laplace generator, alpha = 0.5, betta = 0:
Hi Squared Pirson criteria: 42.81270564557594 <= 70.99345283378227
Kolmogorov criteria: 0.8620742035777464 <= 1.3580986393225505
theoretical expectation: 0
empirical expectation: 0.012242333617161668
theoretical dispersion: 8.0
empirical dispersion: 6.039302085513919

Weibull generator, lambda = 1, k = 0.5:
Hi Squared Pirson criteria: 118.4822098553722 <= 118.75161175336736
Kolmogorov criteria: 0.4340890407142095 <= 1.3580986393225505
theoretical expectation: 2.0
empirical expectation: 1.9964551645252555
theoretical dispersion: 20.0
empirical dispersion: 17.104799371406074
```

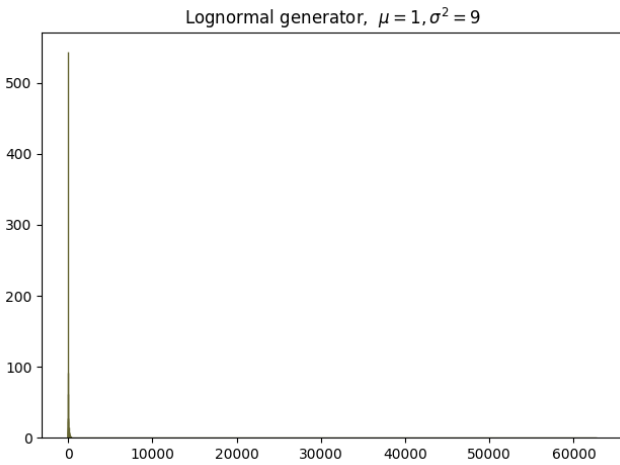
Рис. 8: Результат выполнения программы: проверка критерием согласия Пирсона и Колмогорова. Вывод теоретических и подсчёт эмпирических математических ожиданий и дисперсий.



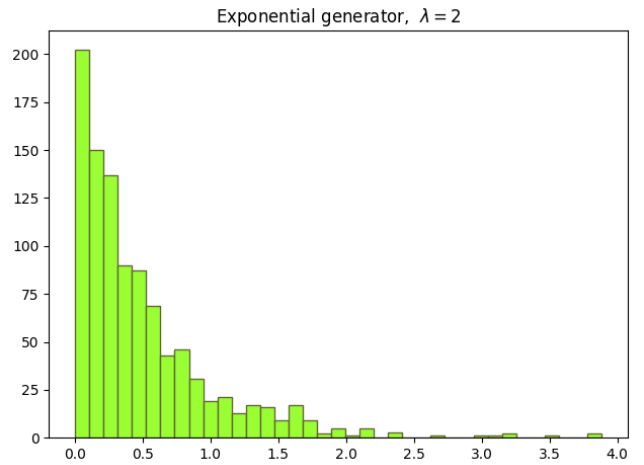
(a) Диаграмма выборки, полученной генератором нормального распределения при  $\mu = 1, \sigma^2 = 9$ .



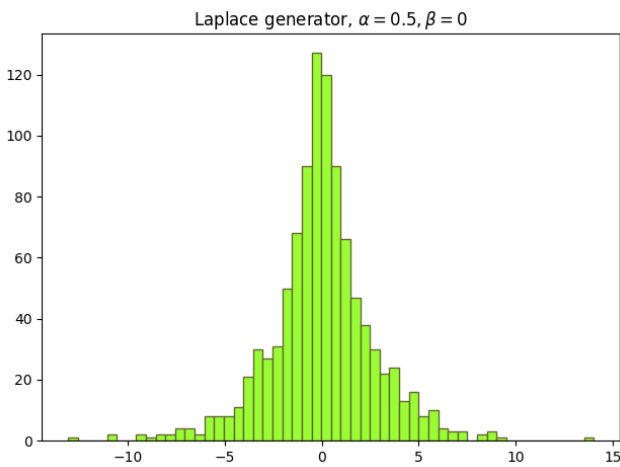
(b) Диаграмма выборки, полученной генератором нормального распределения при  $\mu = 0, \sigma^2 = 1$ .



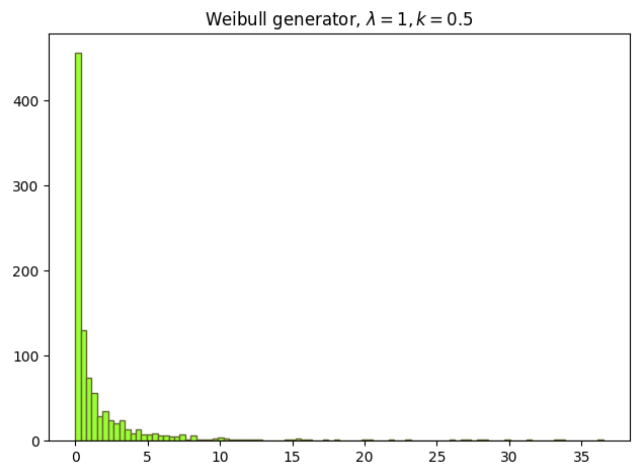
(c) Диаграмма выборки, полученной генератором логнормального распределения при  $\mu = 1, \sigma^2 = 9$ .



(d) Диаграмма выборки, полученной генератором экспоненциального распределения при  $\lambda = 2$ .



(e) Диаграмма выборки, полученной генератором распределения Лапласа при  $\alpha = 0.5, \beta = 0$ .



(f) Диаграмма выборки, полученной генератором распределения Вейбула при  $\lambda = 1, k = 0.5$ .