Долгосрочные домашние задания

Вишнёв Елисей, СКБ182

In [6]:
```
# импортирование библиотек

from random import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
import math
from math import sqrt
%matplotlib inline
```

Домашнее задание 1

1.1. Выбираем распределния

Дискретное – геометрическое: $P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\};$

Непрерывное – треугольное:

$$f(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ \frac{2 \cdot x}{p} & \text{если } x \in [0, p] \\ \frac{2 \cdot (1-x)}{1-p} & \text{если } x \in [p, 1] \\ 0 & \text{если } x \in [1, +\infty] \end{cases}$$

$$p \in (0, 1)$$

</h1>

Дополнительно будет рассматриваться распределение Ципфа: $P(x) = \frac{x^{-s}}{H_{N,s}}, , x \in \{1, 2...N\}, H_{N,s} = \sum_{n=1}^{N} n^{-s}$

1.2 Вывод основных характеристик распредлений

Геометрическое

Математическое ожидание:

$$E\xi = \sum_{i \geq 1} x_i \cdot p_i$$

Пусть случайная величина принимает значение, аналогичное числу неудачных попыток.

$$E\xi = \sum_{i=0}^{\infty} i \cdot p(i) = \sum_{i=0}^{\infty} i \cdot q^i \cdot p = \sum_{i=0}^{\infty} i \cdot (p-1)^i \cdot p = p \cdot \sum_{i=0}^{\infty} i \cdot (1-p)^i = p \cdot (1-p) \cdot \sum_{i=0}^{\infty} i \cdot (1-p)^{i-1} = p \cdot (1-p) \cdot \frac{1}{(1-(1}$$

Дисперсия:

$$E\xi^2 = \sum_{i=0}^{\infty} i^2 \cdot p(i) = \sum_{i=0}^{\infty} i^2 \cdot q^i \cdot p = \sum_{i=0}^{\infty} i^2 \cdot (1-p)^i \cdot p = p \cdot \sum_{i=0}^{\infty} i^2 \cdot (1-p)^i = p \cdot (1-p) \cdot \sum_{i=1}^{\infty} i^2 \cdot (1-p)^{i-1} = p \cdot (1-p) \cdot$$

$$D\xi = E\xi^2 - (E\xi)^2 = \frac{(1+q) \cdot q}{p^2} - \frac{q^2}{p^2} = \frac{q+q^2-q^2}{p^2} = \frac{q}{p^2}$$

Мода:

$$P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\};$$

Вероятности значений случайно величины образуют убывающую геометрическую прогрессию, где q – знаменатель прогрессии, который меньше единицы. Значит, вероятность каждого последующего значения случайной величины меньше, чем текущего. Следовательно, нет ни одного значения, вероятность которого больше вероятностей двух соседних значений. Исключение – самое первое зачение, оно больше своего единственного соседа, стоящего справа. В данном случае самое первое значение равно 0, поэтому 0 – мода распределения.

Медиана:

$$P(\xi \leq \mu) \geq \frac{1}{2}, P(\xi \geq \mu) \geq \frac{1}{2}$$

$$P(\xi \leq \mu) \geq 0.5$$

$$P(\xi \leq \mu) = \sum_{k=0}^{\mu} P(k) = \sum_{k=0}^{\mu} q^k \cdot p = \text{...свойство геометрической прогрессии...}$$

$$= \frac{p \cdot (q^{\mu}-1)}{q-1} = \frac{p \cdot (q^{\mu}-1)}{p} = -(q^{\mu}-1) = 0.5$$

$$q^{\mu} = 0.5$$

$$\mu = \log_q(0.5)$$

Производящая функция:

$$Ms^v = \sum_{k=0}^{\infty} s^k \cdot P(v=k) = \sum_{k=0}^{n} s^k \cdot (1-p)^k \cdot p = \sum_{k=0}^{n} s^k p \cdot q^k = p \cdot \sum_{k=0}^{n} s^k \cdot q^k = \frac{p}{1-q \cdot s}$$

Характеристическая функция:

$$f(t) = \sum_k e^{i \cdot t \cdot x_k} \cdot P(\xi = x_k) = \sum_{k=0}^{\infty} e^{i \cdot t \cdot k} \cdot (1-p)^k \cdot p = p \cdot \sum_{k=0}^{\infty} (e^{i \cdot t} \cdot (q))^k = p \cdot \sum_{k=1}^{\infty} (e^{i \cdot t} \cdot q)^{k-1} = p \cdot \cdot \frac{1}{1-e^{i \cdot t} \cdot q} = \frac{p}{1-e^{i \cdot t} \cdot q}$$

Функция распределния:

По определению, функцию распределения $F(x)$ можно посчитать как сумму вероятностей, при которых случайная величина меньше x:

$$F(x) = \sum_{k=0}^{x} P(k) = \sum_{k=0}^{x} p \cdot q^k$$

Заметим, что это сумма геометрической прогрессии для первых x членов. Воспользуемся соответствующей формулой:

$$S_x = \frac{b_1 \cdot (1 - q^x)}{1 - q} = \frac{p \cdot (1 - q^x)}{p} = 1 - q^x$$

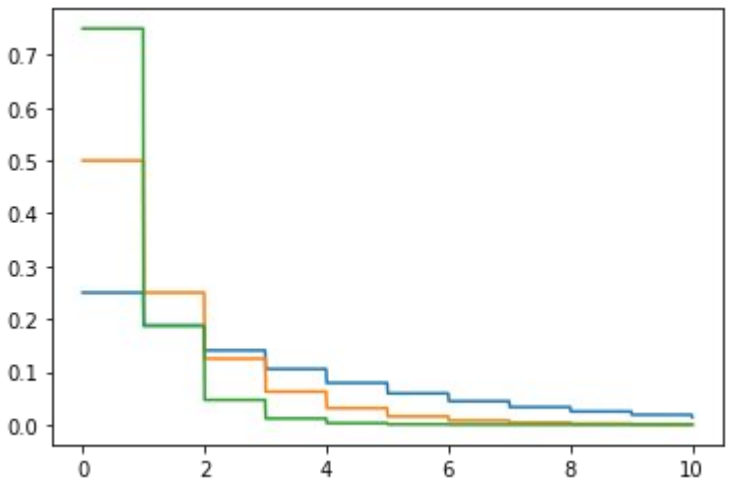
$$F(x) = 1 - q^x$$

Изображаем графически

Гистограмма вероятностей для $p = 0.25, 0.5, 0.75$

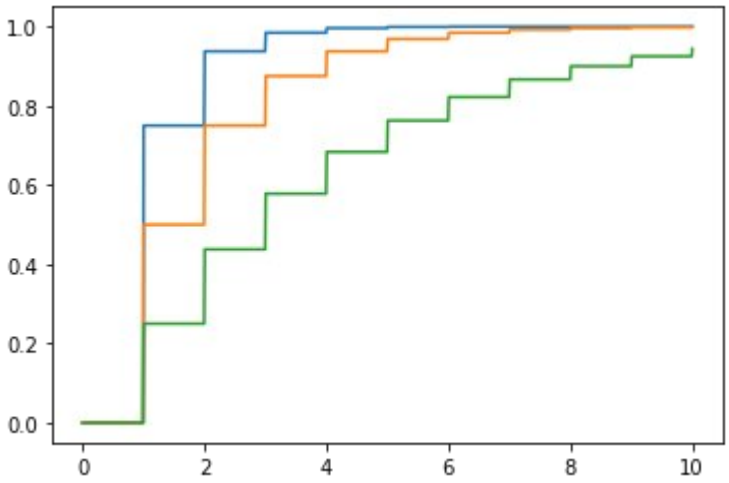
Изображена в виде графика функции, однако в пункте 1.4 будет гистограмма, построенная по полученным значениям функции.

In [2]:
```
x = np.linspace(0,10,1000)

def drawGeomPMF(p):
    geom_rv = sts.geom(p)
    pmf = geom_rv.pmf(x//1+1) #нужно приводить к целому, так как в нецелых зн
ачениях аргумента функция возвращает 0
    plt.plot(x, pmf)

drawGeomPMF(0.25)
drawGeomPMF(0.5)
drawGeomPMF(0.75)
```



Функции распределения для $p = 0.25, 0.5, 0.75$

In [103]:
```
x = np.linspace(0,10,1000)
def drawGeom(p):
    geom_rv = sts.geom(p)
    cdf = geom_rv.cdf(x)
    plt.plot(x, cdf)

drawGeom(0.75)
drawGeom(0.5)
drawGeom(0.25)
```



Треугольное распределние

Математическое ожидание:

$$E\xi = \sum_{i \geq 1} x_i \cdot p_i$$

$$E\xi = \int_{\mathrm{R}} x \cdot f(x) dx$$

$$E\xi = \int_{-\infty}^{0} x \cdot 0 dx + \int_{0}^{p} x \cdot \frac{2 \cdot x}{p} dx + \int_{p}^{1} x \cdot \frac{2 \cdot (1 - x)}{1 - p} dx + \int_{1}^{\infty} x \cdot 0 dx = \frac{2}{p} \cdot \int_{0}^{p} x^2 dx + \frac{2}{1 - p} \cdot \int_{p}^{1} x - x^2 dx = \frac{2}{p} \cdot \frac{p^3}{3} + \frac{2}{1 - p} \cdot (\frac{1^2}{2} - \frac{1^3}{3} - \frac{p^2}{2} + \frac{p^3}{3}) = \frac{p + 1}{3}$$

Дисперсия:

$$E\xi^2 = 0 + \int_{0}^{p} x^2 \cdot \frac{2 \cdot x}{p} dx + \int_{p}^{1} x^2 \cdot \frac{2 \cdot (1 - x)}{1 - p} dx + 0 = \frac{2}{p} \cdot \int_{0}^{p} x^3 dx + \frac{2}{1 - p} \cdot \int_{p}^{1} x^2 - x^3 dx = \frac{2}{p} \cdot \frac{p^4}{4} + \frac{2}{1 - p} \cdot (\frac{1^3}{3} - \frac{1^4}{4} - \frac{p^3}{3} + \frac{p^4}{4}) = \frac{p^2 + p}{6}$$

$$D\xi = E\xi^2 - (E\xi)^2 = \frac{p^2 + p + 1}{6} - \frac{p^2 + 2 \cdot p + 1}{9} = \frac{p^2 - p + 1}{18}$$

Мода:

Проведём исследования функции плотности распределения. Для начала – ищем производную. Функция задана кусочно, поэтому найдём производную каждого из фрагментов:

$$f'(x) = \begin{cases} \frac{d}{dx} 0 & \text{если } x \in [-\infty, 0] \\ \frac{d}{dx} \frac{2 \cdot x}{p} & \text{если } x \in [0, p] \\ \frac{d}{dx} \frac{2 \cdot (1 - x)}{1 - p} & \text{если } x \in [p, 1] \\ \frac{d}{dx} 0 & \text{если } x \in [1, +\infty] \end{cases}$$

Вычисления производной:

$$f'(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ \frac{2}{p} & \text{если } x \in [0, p] \\ \frac{-2}{1 - p} & \text{если } x \in [p, 1] \\ 0 & \text{если } x \in [1, +\infty] \end{cases}$$

Производная на каждом фрагменте равна константе, следовательно, там моды быть не может. Поэтому все надежды на точки, где функция не дифференцируема – $0, p, 1$.

Итак, на промежутке $x \in [-\infty, 0]$ производная равна нулю – функция постоянна, а на промежутке $x \in [0, p]$ производная положительная – функция возрастает; значит, в точке 0 локального максимума нет. На промежутке $x \in [0, p]$ функция возрастает, а на $x \in [p, 1]$ убывает (производная отрицательна), значит,

точка p – локальный максимум. И, наконец, на промежутке $x \in [1, +\infty]$ функция постоянна, ведь там производная – ноль. Значит, в точке 1 локального максимума нет.

Точка p – единственный локальный максимум, и она является модой.

Медиана:

Найдём величину, при которой достигается равенство:

$$P(\xi \le \mu) = P(\xi \ge \mu) = \frac{1}{2}$$

$$\int_{-\infty}^{\mu} f(x)dx = \frac{1}{2}$$

Заметим, что μ может лежать только на двух фрагментах кусочно-заданной функции из четырёх – $\mu \in [0, p]$ и $\mu \in [p, 1]$. В ином случае вся площадь под графиком будет по одну сторону от μ. Рассмотрим два этих возможных случая.

$\mu \in [0, p]$:

$\int_0^{\mu} \frac{2 \cdot x}{p} dx = \frac{\mu^2}{p} = \frac{1}{2}$, тогда $\mu = \sqrt{\frac{p}{2}}$

$\mu \in [p, 1]$:

$\int_0^{p} \frac{2 \cdot x}{p} dx + \int_p^{\mu} \frac{2 \cdot (1-x)}{1-p} dx = \frac{(-\mu^2 + 2 \cdot \mu - p)}{1-p} = \frac{1}{2}$, получаем квадратное уравнение $\mu^2 - 2 \cdot \mu + (p+1) = 0$, его единственное, принадлежащее данному промежутку, решение: $1 - \sqrt{\frac{1-p}{2}}$

Заметим, что первый случай достигается, когда $p \le \frac{1}{2}$, а второй, когда $p \ge \frac{1}{2}$, тогда полное определение медианы в случае геометрического распределения:

$$\mu = \begin{cases} \sqrt{\frac{p}{2}} & \text{если } p \le \frac{1}{2} \\ 1 - \sqrt{\frac{1-p}{2}} & \text{если } p \ge \frac{1}{2} \end{cases}$$

Характеристическая функция:

Пусть в пределах этого и следующего подпунктов $f(t)$ – характеристическая функция, $p(x)$ – плотность распределения.

$f(t) = \int_R e^{i \cdot t \cdot x} p(x)dx = \int_{-\infty}^{0} e^{i \cdot t \cdot x} 0 dx + \int_0^{p} e^{i \cdot t \cdot x} \frac{2 \cdot x}{p} dx + \int_p^{1} e^{i \cdot t \cdot x} \frac{2 \cdot (1-x)}{1-p} dx + \int_1^{-\infty} e^{i \cdot t \cdot x} \cdot 0 dx = \frac{2}{p} \cdot \int_0^{p} e^{i \cdot t \cdot x} \cdot x dx + \frac{2}{1-p} \cdot \int_p^{1} e^{i \cdot t \cdot x} \cdot ($

$\backslash + 2 \cdot \frac{i \cdot t \cdot e^{i \cdot t \cdot 1} \cdot 1 - e^{i \cdot t \cdot 1} - i \cdot t \cdot e^{i \cdot t \cdot 1} - i \cdot t \cdot e^{i \cdot t \cdot p} \cdot p + e^{i \cdot t \cdot p} + i \cdot t \cdot e^{i \cdot t \cdot p}}{(1-p) \cdot t^2} = -2 \cdot \frac{(1-p) - e^{i \cdot t \cdot p} + p \cdot e^{i \cdot t}}{p \cdot (1-p) \cdot t^2}$

Функция распределния:

$$F(x) = \begin{cases} \int_{-\infty}^{x} 0\,dt & \text{если } x \in [-\infty, 0] \\ \int_{-\infty}^{0} 0\,dt + \int_{0}^{x} \frac{2 \cdot t}{p}\,dt & \text{если } x \in [0, p] \\ \int_{-\infty}^{0} 0\,dt + \int_{0}^{p} \frac{2 \cdot t}{p}\,dt + \int_{p}^{x} \frac{2 \cdot (1-t)}{1-p}\,dt & \text{если } x \in [p, 1] \\ \int_{-\infty}^{0} 0\,dt + \int_{0}^{p} \frac{2 \cdot t}{p}\,dt + \int_{p}^{1} \frac{2 \cdot (1-t)}{1-p}\,dt + \int_{1}^{x} 0\,dt & \text{если } x \in [1, +\infty] \end{cases}$$

Вычислим интегралы:

$$F(x) = \begin{cases} 0 \Big|_{-\infty}^{x} & \text{если } x \in [-\infty, 0] \\ 0 \Big|_{-\infty}^{0} + \frac{t^2}{p} \Big|_{0}^{x} & \text{если } x \in [0, p] \\ 0 \Big|_{-\infty}^{0} + \frac{t^2}{p} \Big|_{0}^{p} + \frac{(2 \cdot t - t^2)}{1-p} \Big|_{p}^{x} & \text{если } x \in [p, 1] \\ 0 \Big|_{-\infty}^{0} + \frac{t^2}{p} \Big|_{0}^{p} + \frac{(2 \cdot t - t^2)}{1-p} \Big|_{p}^{1} + 0 \Big|_{1}^{\infty} & \text{если } x \in [1, +\infty] \end{cases}$$

Подставляем x:

$$F(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ 0 + \frac{x^2}{p} & \text{если } x \in [0, p] \\ 0 + \frac{p^2}{p} + \frac{(2 \cdot x - x^2)}{1-p} - \frac{(2 \cdot p - p^2)}{1-p} & \text{если } x \in [p, 1] \\ 0 + \frac{p^2}{p} + \frac{(2 \cdot 1 - 1^2)}{1-p} - \frac{(2 \cdot p - p^2)}{1-p} + 0 & \text{если } x \in [1, +\infty] \end{cases}$$

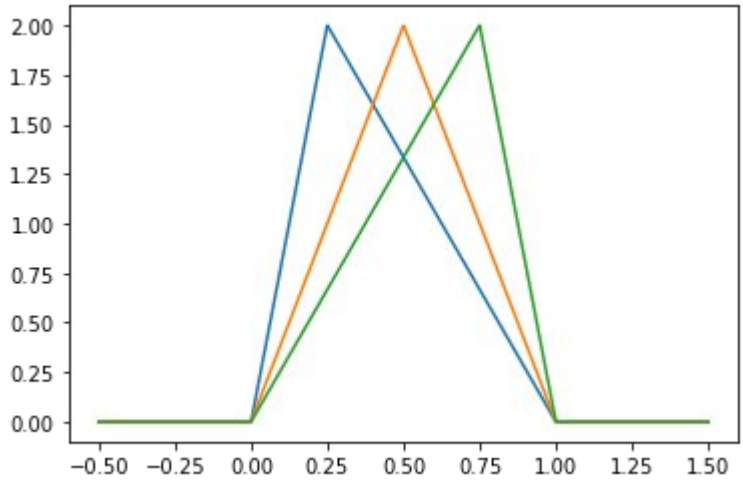
Сокращаем:

$$F(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ \frac{x^2}{p} & \text{если } x \in [0, p] \\ \frac{(-x^2 + 2 \cdot x - p)}{1-p} & \text{если } x \in [p, 1] \\ 1 & \text{если } x \in [1, +\infty] \end{cases}$$

Изображаем графически

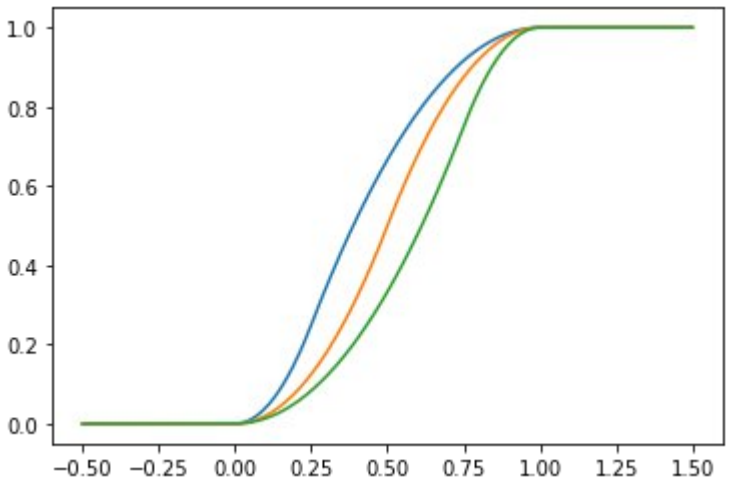
Плотность распределения для $p = 0.25, 0.5, 0.75$

In [60]:
```
x = np.linspace(-0.5,1.5,1000)

def drawTrigPDF(p):
    trig_rv = sts.triang(p)
    pdf = trig_rv.pdf(x)
    plt.plot(x, pdf)

drawTrigPDF(0.25)
drawTrigPDF(0.5)
drawTrigPDF(0.75)
```



Функции распределения для $p = 0.25, 0.5, 0.75$

In [102]:
```
x = np.linspace(-0.5,1.5,1000)

def drawTrigCDF(p):
    trig_rv = sts.triang(p)
    cdf = trig_rv.cdf(x)
    plt.plot(x, cdf)

drawTrigCDF(0.25)
drawTrigCDF(0.5)
drawTrigCDF(0.75)
```



Распределение Ципфа

Математическое ожидание:

$$E\xi = \sum_{i=1}^{N} i \cdot p(i) = \sum_{i=1}^{N} i \cdot \frac{i^{-s}}{H_{N,s}} = \sum_{i=1}^{N} \frac{i^{1-s}}{H_{N,s}} = \frac{\sum_{i=1}^{N} i^{1-s}}{H_{N,s}} = \frac{H_{N,s-1}}{H_{N,s}}$$

Дисперсия:

$$E\xi^2 = \sum_{i=1}^{N} i^2 \cdot p(i) = \sum_{i=1}^{N} i^2 \cdot \frac{i^{-s}}{H_{N,s}} = \frac{H_{N,s-2}}{H_{N,s}}$$

$$D\xi = E\xi^2 - (E\xi)^2 = \frac{H_{N,s-2}}{H_{N,s}} - \frac{{H_{N,s-1}}^2}{H_{N,s}^2} = \frac{H_{N,s-2} \cdot H_{N,s} - {H_{N,s-1}}^2}{H_{N,s}^2}$$

Мода:

$P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\};$

Вероятности значений случайно величины образуют убывающую геометрическую прогрессию. Значит, вероятность каждого последующего значения случайной величины меньше, чем текущего. Следовательно, нет ни одного значения, вероятность которого больше вероятностей двух соседних значений. Исключение – самое первое зачение, оно больше своего единственного соседа, стоящего справа. В данном случае самое первое значение равно 1, поэтому 1 – мода распределения.

Медиана:

$$P(\xi \le \mu) \ge \frac{1}{2}, P(\xi \ge \mu) \ge \frac{1}{2}$$

$$P(\xi \le \mu) \ge 0.5$$

$$P(\xi \le \mu) = \sum_{k=1}^{\mu} P(k) = \sum_{k=0}^{\mu} \frac{i^{-s}}{H_{N,s}} = \frac{H_{\mu,s}}{H_{N,s}} = 0.5$$

Из уравнения $\frac{H_{\mu,s}}{H_{N,s}} = 0.5$ находится μ – искомое значение медианы.

Производящая функция:

$$Mt^v = \sum_{k=1}^{N} t^k \cdot P(v = k) = \sum_{k=1}^{n} t^k \cdot \frac{k^{-s}}{H_{N,s}} = \frac{1}{H_{N,s}} \cdot \sum_{n=1}^{N} \frac{t^n}{n^s}$$

Характеристическая функция:

$$f(t) = \sum_{k} e^{i \cdot t \cdot x_k} \cdot P(\xi = x_k) = \sum_{k=1}^{N} e^{i \cdot t \cdot k} \cdot \frac{i^{-s}}{H_{N,s}} = \frac{1}{H_{N,s}} \cdot \sum_{n=1}^{N} \frac{e^{i \cdot t \cdot n}}{n^s}$$

Функция распределния:

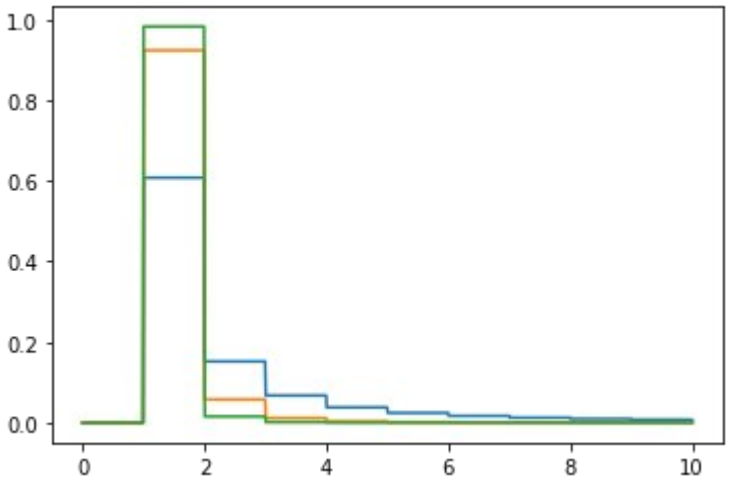
По определению, функцию распределения $F(x)$ можно посчитать как сумму вероятностей, при которых случайная величина меньше x:

$$F(x) = \sum_{k=0}^{x} P(k) = \sum_{i=1}^{x} \frac{i^{-s}}{H_{N,s}} = \frac{H_{x,s}}{H_{N,s}}$$

Изображаем графически

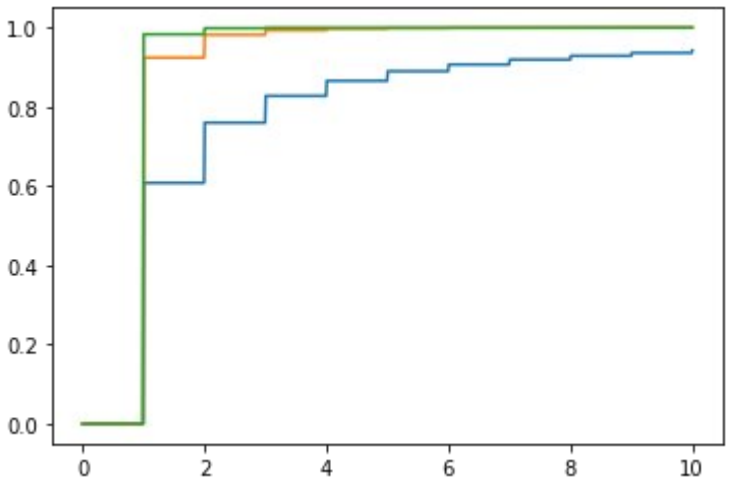
Гистограмма вероятностей для $s = 2, 4, 6$

In [146]:
```
x = np.linspace(0,10,1000)

def drawZipfPMF(s):
    zipf_rv = sts.zipf(s)
    pmf = zipf_rv.pmf(x//1) #нужно приводить к целому, так как в нецелых знач
ениях аргумента функция возвращает 0
    plt.plot(x, pmf)

drawZipfPMF(2)
drawZipfPMF(4)
drawZipfPMF(6)
```



Функции распределения:

In [83]:
```
x = np.linspace(0,10,1000)

def drawZipf(s):
    zipf_rv = sts.zipf(s)
    cdf = zipf_rv.cdf(x)
    plt.plot(x, cdf)

drawZipf(2)
drawZipf(4)
drawZipf(6)
```



1.3 Примеры и события, которые могут быть описаны выбранными случайными величинами

Геометрическое распределение

Всемирная паутина полна классических примеров, когда применяется геометрическое распределение. Наиболее банальный пример – мяч кидают в корзину, и он в неё может попасть с некоторой вероятностью p. В этом случае геометрическое распределение покажет вероятность, что произойдёт k неудач прежде, чем мяч окажется в корзине. Геометрическое распределение действительно можно применить к любой ситуации, когда проводится серия испытаний и подсчитывается вероятность количества неудач перед первым успехом. Постараемся быть чуть более оригинальными.

Пример 1.1. Авторская интерпретация.

Европа, спутник Юпитера, и Энцелад, спутник Сатурна, по мнению автора этой работы являются наиболее вероятными кандидатами среди астрономических объектов Солнечной системы, где может быть обнаружена жизнь внеземного происхождения (или хотя бы её задатки). Обнаружение жизни в этих мирах, пускай даже микроскопической, позволит сформировать более надёжное представление, какими бывают живые существа во Вселенной, и приведёт к прорывным открытиям в области биологии и других как-либо связанных с ней наук.

Предположим, что в обозримом будущем действительно будет организована подобная космическая миссия – например, космический зонд, желательно оснащённый мощным буром, прибудет на поверхность луны Юпитера с целью найти органические соединения, свидельствующие о задатках внеземной жизни. Допустим, зонд оказался на достаточно равномерной по своему составу поверхности.

Территорию (европаторию?) для изучения разделили на одинаковые по площади участки. На каждый участок будет отведено одинаковое количество времени и ресурсов. Стоит грамотно рассчитать топливо и электроэнергию, поэтому один из возникающих вопросов – сколько пройдёт времени и сколько надо потратить ресурсов, чтобы наконец достичь положительного результата?

Теперь сформулируем поставленную задачу на языке вероятностей. С какой вероятностью космическая станция успеет "обработать" x секторов прежде чем найти органические соединения в следующем секторе, если вероятность их обнаружить в любом секторе равна p?

Заметим, что количество затраченных времени и энергии прямо пропорционально количеству изученной площади. Пусть на изучение одного сектора расходуется одна условная единица времени и одна условная единица энергии.

Назовём испытанием процесс изучения очередного сектора космической станции. В таком случае успехом будет событие обнаружения органических соединений в определённом секторе, а неудачей – отведённое время на сектор вышло, и пора переходить к следующему.

Для начала посчитаем вероятность, что органические соединения были обнаружены в первом же секторе. Она равна вероятности найти органические соединения в любом из секторов, то есть p. Это же является вероятностью, что прошло 0 неудач, прежде чем был достигнут успех: $P(0) = p$

Теперь найдём вероятность, что произошла одна неудача до нахождения искомых веществ. Случилось два события: неудача, случившаяся с вероятностью $q = p - 1$, и успех, случившийся с вероятностью p. Будем считать эти события независимыми – вероятность успеха всегда остаётся прежней, неважно, какие были неудачи до этого. По утверждению из лекций, вероятность двух независимых событий равна

произведению их вероятностей: $P(1) = q \cdot p$

Вероятность, что произошло x неудач, считается аналогично. Произошло x независимых событий неудачи, затем произошло одно событие успеха. Так как все эти события независимы:
$P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\}$

Получено геометрическое распределение. Благодаря ему можно, например, посчитать, сколько в среднем уйдёт времени и ресурсов, чтобы найти органические вещества (с помощью математического ожидания). Конечно, в реальной жизни данная модель не целиком соответствует действительности. Геометрическое распределение предполагает бесконечное количество испытаний, а в данной интерпретации рано или поздно закончится вся площадь Европы. Ещё испытания нельзя назвать совсем уж независимыми – ведь если если где-то найдены органические вещества, больше вероятность, что рядом ещё где-то их часть. Но приближенно данная модель работает.

Пример 1.2. Авторская интерпретация, хотя очень сильно вряд ли, что раньше она нигде не возникала.

Теперь подберём что-то ближе к специальности. Для создания алгоритма расчёта контрольных сумм полезно знать, который по счёту бит изменил своё значение, повредив целостность данных. Вопрос на языке теории вероятности: каковая вероятность того, что перед повреждённым битом стоит x неповреждённых, если вероятность повреждения бита в течение некоторого времени равна p?

Определим испытание как воздействие среды на данные в течение некоторого времени. Неудача – бит не повредился. Успех – бит повредился. Как-то по-злодейски, но это лишь в данной модели.

Учтём, что события воздействия на определённый бит являются независимыми. Пусть перед повреждённым битом находится x не повреждённых. Среда воздействовала на каждый из $x + 1$ бит, то есть испытание было проведено со всеми ними. Среди испытаний x раз случилась неудача с вероятностью $q = p - 1$ и один раз случился успех с вероятностью p. Так как каждое из этих событий независимо друг от друга, найдём искомую вероятность как произведение вероятностей всех этих независимых событий: $P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\}$

Получено геометрическое распределение.

Пример 1.3.

В работе Angelos D. Keromytis "Financial Cryptography and Data Security", 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers, геометрическое распределение используется рассчёта и гарантии вычислительной мощности дифференциальной приватности для Бинарного протокола.

Пример 1.4.

В статье Frank C. Kaminsky, James C. Benneyan, Robert D. Davis and Richard J. Burke "Statistical Control Charts Based on a Geometric Distribution", https://doi.org/10.1080/00224065.1992.12015229 (https://doi.org/10.1080/00224065.1992.12015229), в смоделированном примере построение контрольной диаграммы производственных и административных процессов на основе геометрического распределения снижает частоту ложных тревог с 9,6\% до 0,9\%.

Связь геометрического распределения с другими распределениями

Биномиальное

– как бы обрезанная версия, так как отметаются вероятности большинства событий. Событие успеха всегда на последнем месте и всегда одно, а биномиальный кэфициент равен единице – ведь биномальный коэфициент описывает число расстановок успеха и неудач, а здесь расстановка единственная – n неудач подряд и на последнем месте успех.

Равномерное

– пересекаются в вырожденном случае, когда успех должен быть в первый же раз и его вероятность равна 1/2. Тогда что успех, что неудача равновероятны, и это равномерное распредлеение.

Отрицательное биномиальное

– геометрическое есть частный случай его, когда число успехов равно единице.

Пуассона

– пусть x = 1, так как успех у нас должен быть ровно один раз. Но ещё успех должен быть ровно на последнем месте из n испытаний, так что делим на n.

Гипергеометрическое

– по названию должны быть связаны, но связи найдено не было.

Треугольное распределение

Пример 2.1. Авторская интерпретация.

Попытка связать текущий пример с первым, ведь так по-литераторному красивее.

Допустим, что гипотетическая станция при подлёте обнаружила органические вещества в некотором секторе. При этом источников этих веществ оказалось два, однако точную их широту определить невозможно. Известно, что оба источника располагаются на широте в диапазоне $[0, p]$ условных единиц, $p \in (0, 1)$. Космическая станция намеревается спуститься на поверхность и изучить этот диапазон для более детального исследования и нахождения источников. Этот сектор равномерный – вероятсность обнаружить каждый из источников в любой точке одинаково, расположение источников независимо друг от друга, и станция намеревается равномерно проехать весь диапазон широты $[0, p]$ условных единиц с постоянной скоростью. Можно поставить вопрос – проехав какую часть пути, станция наткнётся на хотя бы один из источников?

Сначала следует рассудить отдельно с одним из источников. Так как поверхность однородная, то в диапазоне $[0, p]$ условных единиц, проезжая одинаковые промежутки пути, вероятность обнаружить источник будет одинакова. Данная модель соответствует равномерному распределению. На лекциях оно было изучено, и пример интерпретанции ранвомерного распределения является отдельным заданием, поэтому сразу же приведём плотность данного распределения:

$$f_1(x) = \begin{cases} \frac{1}{p} & \text{если } x \in [0, p] \\ 0 & \text{иначе} \end{cases}$$

Нахождение второго источника можно рассмотреть по этой же модели – плотность распределения будет той же: $f_1(x) = f_2(x)$. Теперь находим плотность распределения для обнаружения двух источников сразу. Так как обнаружение каждого из них – события независимые, воспользуемся формулой свёртки

распределений:

$$f(x) = \int_{-\infty}^{\infty} f_2(t) \cdot f_1(x-t)dt$$

Анализируем, когда функции принимают ненулевые значения:

$f_2(t): 0 < t < p$

$f_1(t): 0 < x - t < p$, тогда $x - p < t < x$

В зависимости от того, чему равен x, два полученных промежутка $[0, p]$ и $[x - p, x]$ накладываются по-разному друг на друга. Возможны четыре возможных случая:

а) первый промежуток идёт до второго, пересечения нет; б) первый промежуток до второго, пересечение есть;

в) первый промежуток после второго, пересечение есть;

г) первый промежуток после второго, пересечения нет.

В зависимости от случая плотности функций принимают разные значения. Когда $x \in [0, p]$, то $f_2(t) = 0$, получаем, что $f(x) = 0$. Аналогично, подставляя значения функций, получаем треугольное распределение:

$$f(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ \frac{2 \cdot x}{p} & \text{если } x \in [0, p] \\ \frac{2 \cdot (1-x)}{1-p} & \text{если } x \in [p, 1] \\ 0 & \text{если } x \in [1 + \infty] \end{cases}$$

Пример 2.2.

Треугольное распределение можно использовать при недостатке информации о случайной величине. Пусть распределение случайной величины неравномерное, а плотность с какой-то точки начинает заметно возрастать, а пройдя максимум – заметно убывать, где после какой-то точки она постепенно близится к нулю. Если посмотреть на график плотности, то отметив точку, где он начинает резко возрастать, как 0, а точку максимума как p, и точку, где функция перестаёт резко убывать, как 1, то получится треугольное распределение, описывающее распреление приближенно. Подобных распределений много, напрмиер, нормальное, экспоненциальное. А такая округлённая оценка позволит получить более точные сведения.

Связь треугольного распределения с другими распределениями

Нормальное

– треугольное распределение можно использовать, чтобы грубо и приблизительно оценить некоторое нормальное распределение случайной величины. На рисунке приведён пример, как можно это реализовать.

Равномерное

– из примера 2.1. видно, что сумма двух независимых случайных величин, распределённых равномерно, образует случайную величину, распределённую треугольно.

Лапласа, Коши, Экспоненциальное, Гамма, Максвелла

– аналогично нормальному, можем так же их точку максимума обозначить за p, а точки, где функция почти выровнялась к нулю, – за 0 и 1 соответственно, стоит отметить, в этом случае треугольник близок к равнобедренному, а p стремится к значению 0.5 за счёт симметричности функций.

Эрланга, Рэлея, Логнормальное, Парето

– тоже можно грубо округлить, но треугольник будет далёк от равнобедренного. В случае Парето $p = 1$.

Распределение Ципфа

Пример 3.1.

Распределение населения по городам подчиняется закону Ципфа. Однако точного соответствия закону нет.

Пример 3.2.

Если упорядочить все слова по частоте использования, то их распределение будет соответствовать закону Ципфа.

1.4. Моделирование выбранных случайных величин

Геометрическое распределние

Моделирование данной случайной величины будет происходить на основе непрерывного равномерного распределения на отрезке $[0, 1]$. Примем, что функция random.random() возвращает значение такой случайной величины. $P(x) = P(\xi = x) = p \cdot q^x, x \in \mathrm{N} \cup \{0\}$.

Представим себе отрезок $[0, 1]$ с точкой p на нём. Вспомним, что вероятность попасть в конкретную точку на отрезке равна нулю, но вероятность попасть в некоторую часть отрезка длиной l равна l. Следовательно, на данном отрезке вероятность попасть в промежуток $[0, p]$ равна p. Идём далее. Поставим следующую точку на отрезке – $p + p \cdot (p - 1) = p + p \cdot q$. Посчитаем длину отрезка $[p, p + p \cdot q]$. Она равна $p + p \cdot q - p = p \cdot q$. Значит, вероятность попасть в этот промежуток равна $p \cdot q$. Дальше ставим на отрезке точки $p + p \cdot q + p \cdot q^2, p + p \cdot q + p \cdot q^2 + p \cdot q^3, \ldots, \sum_{i=0}^{x} p \cdot q^i, \ldots$ и так до бесконечности. То есть разбили отрезок на бесконечное количество промежутков вида $[\sum_{i=0}^{x-1} p \cdot q^i, \sum_{i=0}^{x} p \cdot q^i]$. Тогда вероятность того, что непрерывная равномерная случайная величина, распределённая на данном отрезке, попадёт в промежуток $[\sum_{i=0}^{x-1} p \cdot q^i, \sum_{i=0}^{x} p \cdot q^i]$, равна $\sum_{i=0}^{x} p \cdot q^i - \sum_{i=0}^{x-1} p \cdot q^i = p \cdot q^x$. Иными словами,

$P(x) = p \cdot q^x$

Следовательно, получен способ моделирования геометрически распределённой случайной величины. Конкретное её значние как бы должно лежать на соответстсующем ему интервале $[\sum_{i=0}^{x-1} p \cdot q^i, \sum_{i=0}^{x} p \cdot q^i]$.
Применим свойство суммы геометрической прогрессии и запишем промежуток эквивалентно:
$[1 - q^x - p \cdot q^x, 1 - q^x]$.

Тогда предоставим код, генерируеющий значение случайной величины непрерывного равномерного распределения и выдающий конкретное значение на основе того, в каком промежутке лежит полученная случайна величина. Идея алгоритма проста – мы получили некоторое наблюдение n (равномерное непрерывное распределние), и нам следует проверить, в каком конекренто промежутке оно лежит. Не будем гнаться за производительностью, так что позволим себе последовательно проверить каждый из промежутков на предмет того, находится ли там n. Перейдя в очередной i-ый промежуток, проверим, лежит ли n в нём. Если нет, то переходим к следующему. И так пока не найдём нужный промежуток. Заметим, что по заданному распределению функция должна возвращать и 0 с определённой вероятностью. И так как цикл *while* отработает хотя бы раз всегда, в конце возвращаем значение на единицу меньше.

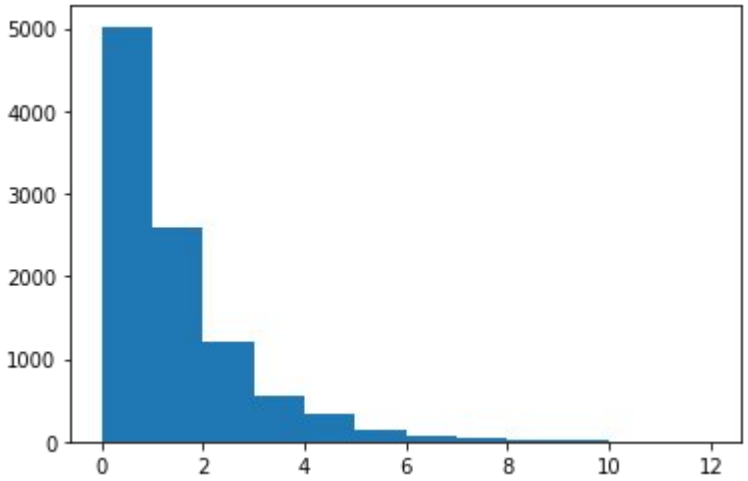
Заметим, что теоретически функция может обрабатывать результат бесконечное время, так как у нас бесконечное количество промежутков, каждый из которых, возможно, придётся проверить. Но из-за ограниченного количества памяти количество промежутков всегда конечно, поэтому функция всегда завершится.

In [84]:
```
def GeomCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    s = 0
    i = 0
    q = 1 - p
    n = random()
    while s <= n:
        s += p * q**i
        i += 1
    return i-1
```

Можем построить гистограмму по получаемым значениям от этой функции:

In [85]:
```
l = [GeomCustom(0.5) for i in range (10000)]
plt.hist(l, bins = max(l))
```

Out[85]:
```
(array([5.021e+03, 2.597e+03, 1.199e+03, 5.600e+02, 3.290e+02, 1.440e+02,
        7.600e+01, 3.200e+01, 2.200e+01, 1.200e+01, 2.000e+00, 6.000e+00]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.]),
 <a list of 12 Patch objects>)
```



Ещё можно воспользоваться готовой реализацией *geom* из модуля *scipy. stats*. К сожалению, автору работы не удалось найти её исходный код, так что пользоваться ей следует с некоторым опасениями.

Треугольное распределение

Воспользуемся методом обратного преобразования для моделирования данной случайной величины.

Пусть a – некоторое число на вещественной прямой.

Из определения функции распределения, $P(x_i \leq a) = F(a)$

y_i – реализации случайной величины. Тогда

$x_i = F^{-1}(y_i)$

$P(x_i \leq a) = F(a)$

$P(x_i \leq a) = P(y_i \in [0; F(a)])$

Находим обратную функцию треугольного распределения

$$F^{-1}(x) = \begin{cases} \sqrt{p \cdot x} & \text{если } x \in [0, p] \\ 1 - \sqrt{(1-p) \cdot (1-x)} & \text{если } x \in [p, 1] \end{cases}$$

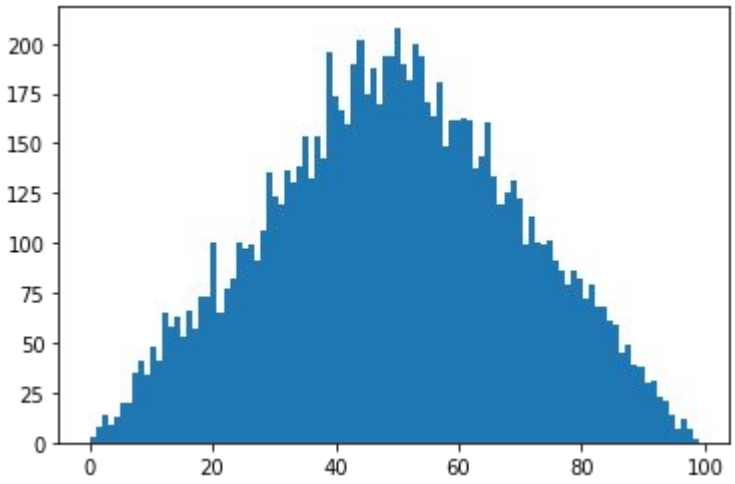
Реализуем её

In [76]:
```
def TrigCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    n = random()
    if n < p:
        return sqrt(p * n)
    else:
        return 1 - sqrt((1-p)*(1-n))
```

Построим гистограмму по значениям, получаемым от функции. Так как метод *hist* работает только с целыми числами, будем полученные зачения умножать на сто и округлять до целых (например, в меньшую сторону). Таким образом мы будем получать числа от 0 до 99, имеющие треугольное распределние.

In [138]:
```
l = [int(TrigCustom(0.5)*100//1) for i in range (10000)]
plt.hist(l, bins = 100)
```

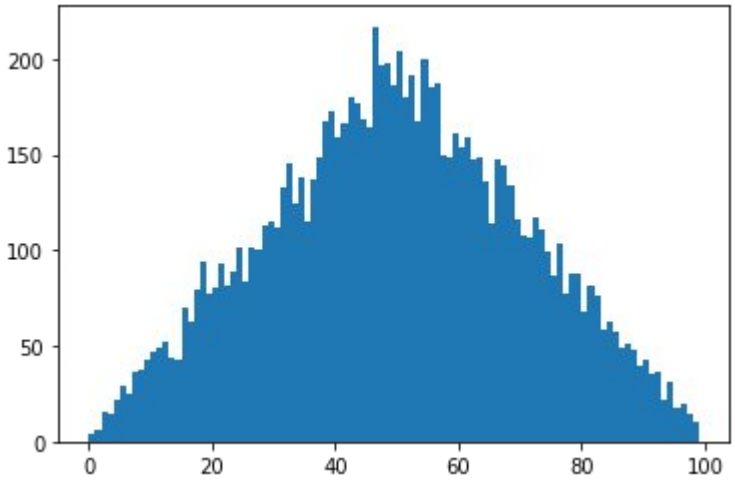
Out[138]:
```
(array([  3.,   8.,  14.,   9.,  13.,  20.,  20.,  35.,  41.,  34.,  48.,
         41.,  65.,  58.,  63.,  53.,  66.,  57.,  73.,  73., 100.,  65.,
         77.,  82., 100.,  97.,  99.,  91., 106., 135., 123., 119., 136.,
        130., 138., 153., 132., 153., 142., 196., 174., 167., 159., 190.,
        202., 175., 188., 170., 194., 194., 208., 190., 182., 200., 194.,
        171., 164., 181., 148., 162., 162., 163., 162., 137., 143., 161.,
        133., 119., 125., 131., 122.,  99., 113., 100.,  99., 101.,  91.,
         86.,  79.,  86.,  82.,  72.,  79.,  68.,  68.,  61.,  59.,  45.,
         49.,  39.,  38.,  30.,  31.,  23.,  21.,  14.,   7.,  12.,   7.,
          2.]),
 array([ 0.  ,  0.99,  1.98,  2.97,  3.96,  4.95,  5.94,  6.93,  7.92,
         8.91,  9.9 , 10.89, 11.88, 12.87, 13.86, 14.85, 15.84, 16.83,
        17.82, 18.81, 19.8 , 20.79, 21.78, 22.77, 23.76, 24.75, 25.74,
        26.73, 27.72, 28.71, 29.7 , 30.69, 31.68, 32.67, 33.66, 34.65,
        35.64, 36.63, 37.62, 38.61, 39.6 , 40.59, 41.58, 42.57, 43.56,
        44.55, 45.54, 46.53, 47.52, 48.51, 49.5 , 50.49, 51.48, 52.47,
        53.46, 54.45, 55.44, 56.43, 57.42, 58.41, 59.4 , 60.39, 61.38,
        62.37, 63.36, 64.35, 65.34, 66.33, 67.32, 68.31, 69.3 , 70.29,
        71.28, 72.27, 73.26, 74.25, 75.24, 76.23, 77.22, 78.21, 79.2 ,
        80.19, 81.18, 82.17, 83.16, 84.15, 85.14, 86.13, 87.12, 88.11,
        89.1 , 90.09, 91.08, 92.07, 93.06, 94.05, 95.04, 96.03, 97.02,
        98.01, 99.  ]),
 <a list of 100 Patch objects>)
```



В пункте 1.3 (Пример 2.1) рассматривалась авторская интерпретация геометрического распределения, где было выведено, что сумма двух непрерывно равномерно распределённых случайных величин образует треугольно распределённую случайную величину. Мы можем воспользоваться этим для соответствующего способа моделирования случайной величины.

In [118]:
```
def TrigCustomR():
        return 0.5*(random()+random())

l = [int(TrigCustomR()*100//1) for i in range (10000)]
plt.hist(l, bins = max(l))
```

Out[118]:
```
(array([  4.,   6.,  16.,  14.,  22.,  29.,  25.,  36.,  38.,  43.,  47.,
         49.,  52.,  44.,  43.,  70.,  63.,  79.,  94.,  77.,  80.,  93.,
         82.,  89., 101.,  84., 101., 100., 113., 115., 112., 133., 145.,
        125., 138., 115., 137., 149., 167., 173., 159., 166., 180., 177.,
        169., 164., 217., 197., 198., 186., 204., 180., 192., 167., 200.,
        185., 187., 150., 149., 161., 154., 159., 148., 149., 136., 114.,
        148., 144., 134., 116., 108., 107., 117., 111.,  99.,  87., 104.,
         77.,  88.,  88.,  68.,  82.,  76.,  58.,  63.,  57.,  49.,  51.,
         48.,  40.,  43.,  35.,  36.,  22.,  31.,  18.,  20.,  14.,  10.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
        13., 14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25.,
        26., 27., 28., 29., 30., 31., 32., 33., 34., 35., 36., 37., 38.,
        39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50., 51.,
        52., 53., 54., 55., 56., 57., 58., 59., 60., 61., 62., 63., 64.,
        65., 66., 67., 68., 69., 70., 71., 72., 73., 74., 75., 76., 77.,
        78., 79., 80., 81., 82., 83., 84., 85., 86., 87., 88., 89., 90.,
        91., 92., 93., 94., 95., 96., 97., 98., 99.]),
 <a list of 99 Patch objects>)
```



Так же в модуле *stats* есть готовый метод *triang*, который так же можно использоваться для работы с этим распределением.

Мы можем сравнить все три метода по производительности. Сгенерируем выборки из 100 000 значений каждым из трёх способов и выясним, сколько времени на это нужно каждому из методов.

In [141]:
```
import time

start_time = time.time()
[int(TrigCustom(0.5)*100//1) for i in range (100000)]
print("По методу обратного преобразования:\n %s секунды\n" % (time.time() - s
tart_time))

start_time = time.time()
[int(TrigCustomR()*100//1) for i in range (100000)]
print("По методу суммы двух случайных величин:\n %s секунды\n" % (time.time()
- start_time))

start_time = time.time()
[int(sts.triang.rvs(0.5)*100//1) for i in range (100000)]
print("По готовому методу:\n %s секунд" % (time.time() - start_time))
```

```
По методу обратного преобразования:
 0.09458780288696289 секунды

По методу суммы двух случайных величин:
 0.06643819808959961 секунды

По готовому методу:
 5.817661762237549 секунд
```

Результат показал, что быстрее всех работает метод моделирования с помощью суммы двух случайных величин. Чуть медленнее работает метод обратного преобразования, но это незначительно. А использование готового метода sts.triang.rvs(0.5) замедляет скорость программы почти в 100 раз.

Распределение Ципфа

$$P(x) = \frac{x^{-s}}{H_{N,s}}, x \in \{1, 2...N\}, H_{N,s} = \sum_{n=1}^{N} n^{-s}$$

Аналогично со случаем моделирования геометрически распределённой случайной величины, будем опираться на равномерное непрерывное распределение на отрезке $[0, 1]$. Идейно будем использовать тот же подход. Случайная величина может принимать N различных целочисленных значений – от 1 до N. Так что если мы отрезок $[0, 1]$ разобьём на N промежутков, каждый из которых будет иметь длину $\frac{x^{-s}}{H_{N,s}}, x \in \{1, 2...N\}$, то веротность попасть в соответствующий промежуток будет $\frac{x^{-s}}{H_{N,s}}$. Реализуем это.

Первый промежуток – $[0, \frac{1^{-s}}{H_{N,s}}]$, второй – $(\frac{1^{-s}}{H_{N,s}}, \frac{1^{-s}}{H_{N,s}} + \frac{2^{-s}}{H_{N,s}}]$, i-ый – $(\sum_{k=0}^{i-1} \frac{(k)^{-s}}{H_{N,s}}, \sum_{k=0}^{i} \frac{(k)^{-s}}{H_{N,s}}]$.

Соответственно длина x-ого промежутка: $\sum_{k=0}^{x} \frac{(k)^{-s}}{H_{N,s}} - \sum_{k=0}^{x-1} \frac{(k)^{-s}}{H_{N,s}} = \frac{x^{-s}}{H_{N,s}} = \frac{x^{-s}}{\sum_{n=1}^{N} n^{-s}}$. Как утверждалось ранее, вероятность попасть в промежуток длиной l равна l. Значит, мы добились искомого распределения.

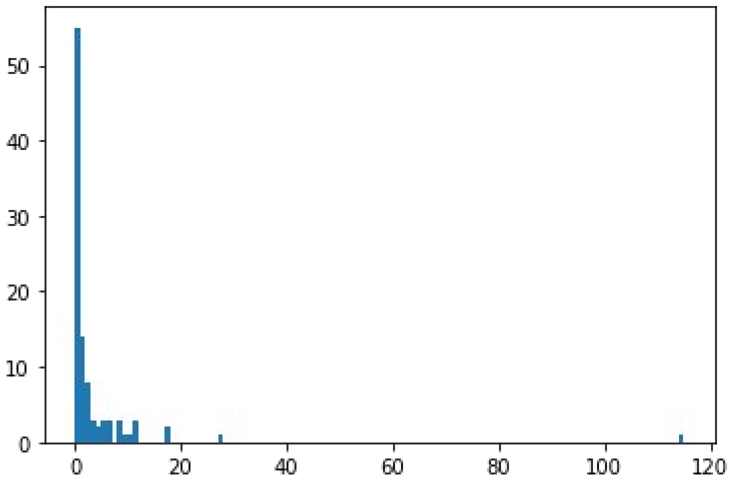
Алгоритм реализуем следующим образом. Пусть заданы N и s. Получим значение случайной величины, распределённой непрерывно равномерно на $[0, 1]$. Будем последовательно проверять каждый из N промежутков на случай того, находится ли в нём полученное значение. Как только оно будет получено, функция вернёт номер промежутка.

In [2]:
```
def ZipfCustom(s, N):
    if (N != (N//1) or N<0):
        return 0 #Проверка на корректность ввода параметра
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    sum = 0
    i = 0
    n = random()
    while sum <= n:
        i += 1
        sum += (i**(-s))/H
    return i-1
```

Можем посмотреть на гистограмму по полученным значениям

In [17]:
```
l = [ZipfCustom(2,500) for i in range (100)]
plt.hist(l, bins = max(l))
```

Out[17]:
```
(array([55., 14.,  8.,  3.,  2.,  3.,  3.,  0.,  3.,  1.,  1.,  3.,  0.,
         0.,  0.,  0.,  0.,  2.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]),
 array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
         11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
         22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
         33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
         44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
         55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
         66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
         77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
         88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
         99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
        110., 111., 112., 113., 114., 115.]),
 <a list of 115 Patch objects>)
```



Однако это уже больше относится ко второму домашнему заданию, к нему мы и перейдём!

Домашнее задание 2

In [1]:
```
# импортирование библиотек

from random import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
from math import sqrt

%matplotlib inline

# Определяем свою эмперическую функцию распределения ниже
```

Исправлено: ЭФР определена по-другому

In [2]:
```
def ECDF(x):
    counts = []
    xset = np.sort(list(set(x)))
    x = np.sort(x)
    for el in xset:
        counts.append(np.count_nonzero(x == el))
    def result(v):
        s = 0
        i = 0
        for i in range(len(xset)):
            if (v < xset[i]):
                break
            s += counts[i]
        return s / x.size
    return result
```

2.1. Моделирование выбранных случайных величин

Выберем коэффициенты для каждого из распределений.

Геометрическое: $p = 0.2$;

Треугольное: $p = 0.2$;

Ципфа: $s = 2, N = 500$.

Моделирование геометрического распределения

Создадим указанные выборки:

In [4]:
```
def GeomCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    s = 0
    i = 1
    q = 1 - p
    n = random()
    while s <= n:
        s += p * q**(i-1)
        i += 1
    return i-1

volumes = (5, 10, 100, 1000, 100000)
SampleGeom = [[[GeomCustom(0.2) for i in range(N)] for i in range(5)] for N i
n volumes]
```

Выведем выборки из 5 и 10 элементов:

In [5]:
```
print(*SampleGeom[0],'\n', sep='\n')
print(*SampleGeom[1], sep='\n')
```

```
[2, 6, 1, 12, 1]
[4, 3, 9, 8, 1]
[5, 17, 11, 4, 1]
[1, 7, 4, 3, 5]
[5, 9, 1, 1, 6]


[10, 15, 9, 4, 1, 15, 3, 1, 9, 1]
[3, 9, 2, 5, 11, 5, 2, 3, 2, 3]
[6, 14, 7, 25, 4, 5, 2, 8, 3, 7]
[2, 1, 5, 3, 10, 1, 7, 1, 5, 1]
[15, 5, 5, 4, 2, 4, 7, 1, 6, 8]
```

Моделирование треугольного распределения

Создадим указанные выборки и выведем выборки из 5 и 10 элементов:

In [6]:
```
def TrigCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    n = random()
    if n < p:
        return sqrt(p * n)
    else:
        return 1 - sqrt((1-p)*(1-n))

volumes = (5, 10, 100, 1000, 100000)
SampleTrig = [[[TrigCustom(0.2) for i in range(N)] for i in range(5)] for N i
n volumes]

for i in range(5):
    for j in range(5):
        print("%.2f" % SampleTrig[0][i][j], end = '\n' if j == 4 else ' ')

print()

for i in range(5):
    for j in range(10):
        print("%.2f" % SampleTrig[1][i][j], end = '\n' if j == 9 else ' ')
```

```
0.33 0.31 0.61 0.24 0.74
0.10 0.59 0.66 0.38 0.55
0.51 0.42 0.32 0.52 0.11
0.42 0.26 0.92 0.47 0.11
0.13 0.40 0.21 0.22 0.28

0.49 0.34 0.25 0.09 0.71 0.78 0.21 0.07 0.23 0.65
0.45 0.22 0.77 0.48 0.50 0.20 0.14 0.46 0.46 0.92
0.84 0.37 0.72 0.59 0.36 0.50 0.95 0.37 0.23 0.48
0.72 0.08 0.26 0.38 0.11 0.20 0.27 0.07 0.76 0.05
0.31 0.16 0.59 0.23 0.20 0.28 0.06 0.25 0.14 0.57
```

Моделирование распределения Ципфа

Создадим указанные выборки и выведем выборки из 5 и 10 элементов:

In [7]:
```
def ZipfCustom(s, N):
    if (N != (N//1) or N<0):
        return 0 #Проверка на корректность ввода параметра
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    sum = 0
    i = 1
    n = random()
    while sum <= n:
        i += 1
        sum += ((i-1)**(-s))/H
    return i-1

volumes = (5, 10, 100, 1000, 100000)
SampleZipf = [[[ZipfCustom(2, 500) for i in range(N)] for i in range(5)] for
N in volumes]

print(*SampleZipf[0],'\n', sep='\n')
print(*SampleZipf[1], sep='\n')
```

```
[1, 24, 4, 2, 2]
[1, 1, 1, 4, 2]
[2, 2, 2, 1, 1]
[1, 1, 1, 1, 1]
[2, 1, 1, 1, 1]


[9, 1, 1, 3, 19, 1, 29, 1, 1, 4]
[7, 2, 1, 6, 1, 1, 2, 1, 1, 2]
[1, 2, 1, 1, 1, 6, 1, 2, 9, 3]
[1, 1, 4, 2, 1, 1, 21, 4, 1, 1]
[2, 1, 1, 1, 1, 2, 2, 1, 6, 3]
```

2.2. Построение эмпирической функции распределениявеличин

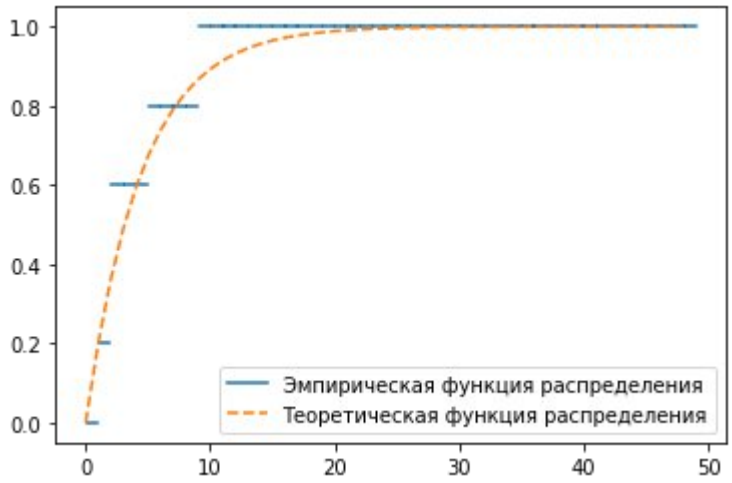
Для геометрического распределения

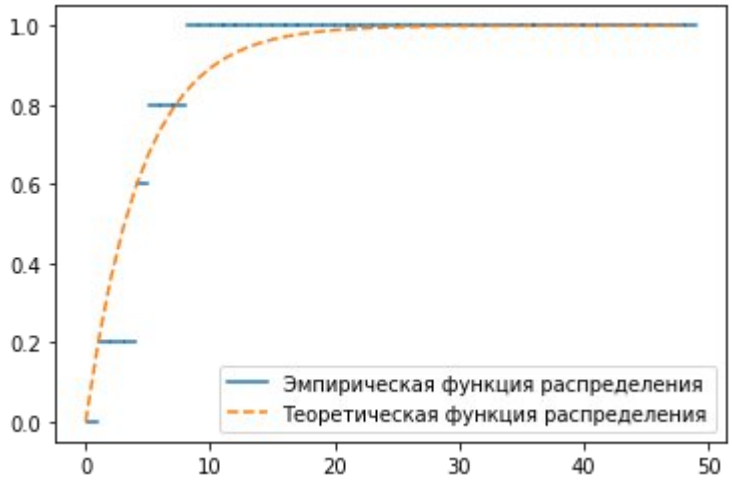
Для сравнения с теоретической функцией распределения воспользуемся готовым методом geom из модуля scipy.stats Заметим, что готовый метод использует вероятность $p(k) = p \cdot q^{k-1}, k \in [1, +\infty)$, поэтому наша "самодельная" функция сгенерировала выборку, соответствующую данному распределению

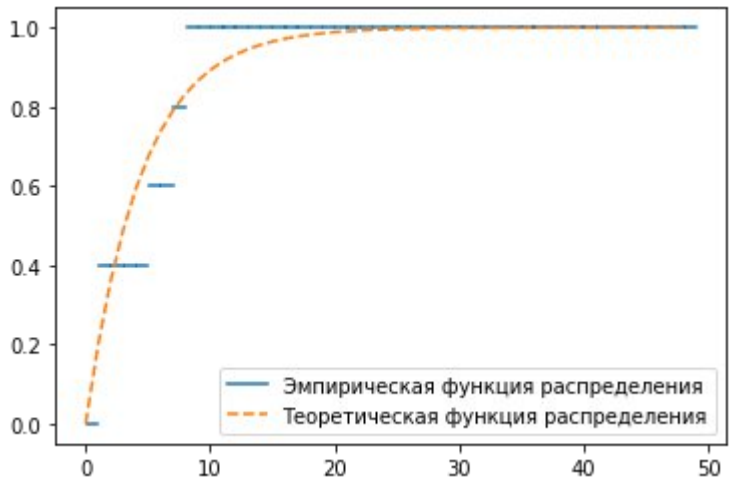
Исправлено: перестроены графики с новым определением ЭФР

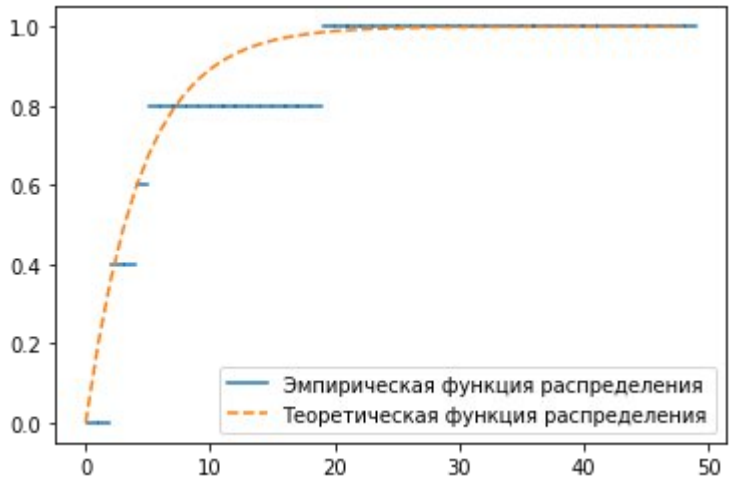
In [101]:

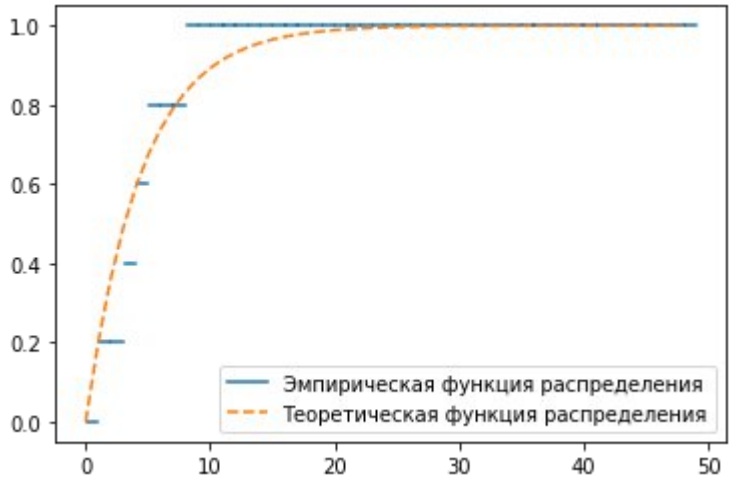
```
ECDFgeom = []
for samples in SampleGeom:
    for sample in samples:
        ecdf = ECDF(sample)
        x1 = range(0, 49)
        x = np.linspace(0,49,1000)
        l = []
        x[x % 1 < 0.05] = np.nan
        for i in x:
            l.append(ecdf(i))
        plt.plot(x, l, label='Эмпирическая функция распределения')
        plt.plot(x1, sts.geom.cdf(x1, 0.2), label='Теоретическая функция расп
ределения', ls='--')
        plt.legend(loc='lower right')
        plt.show()
```

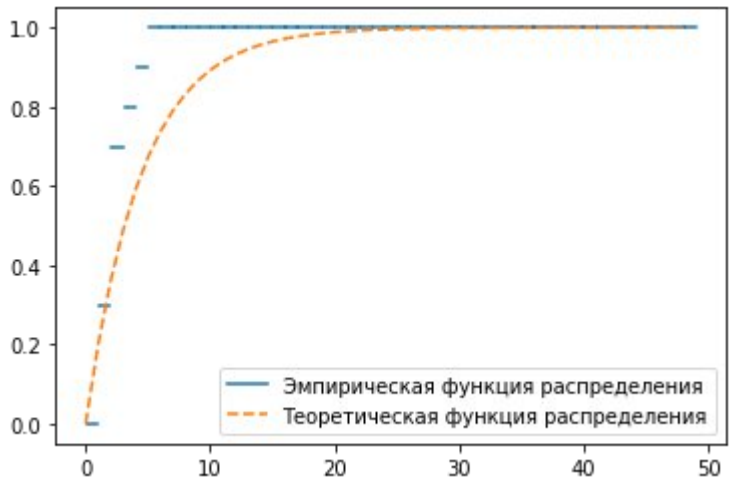

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

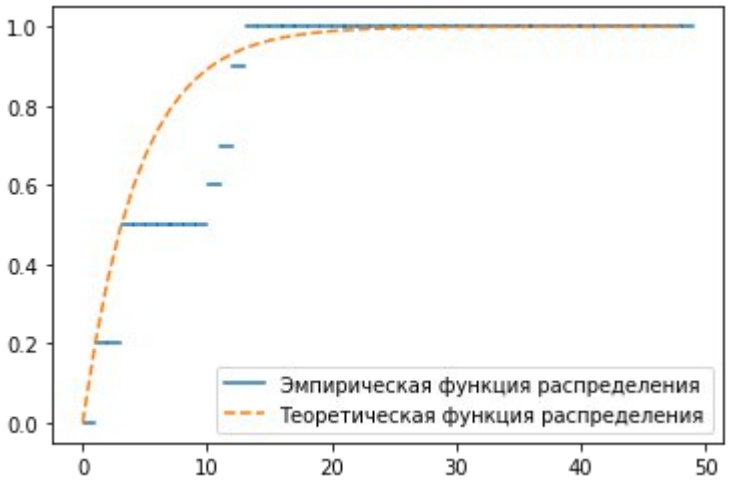

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

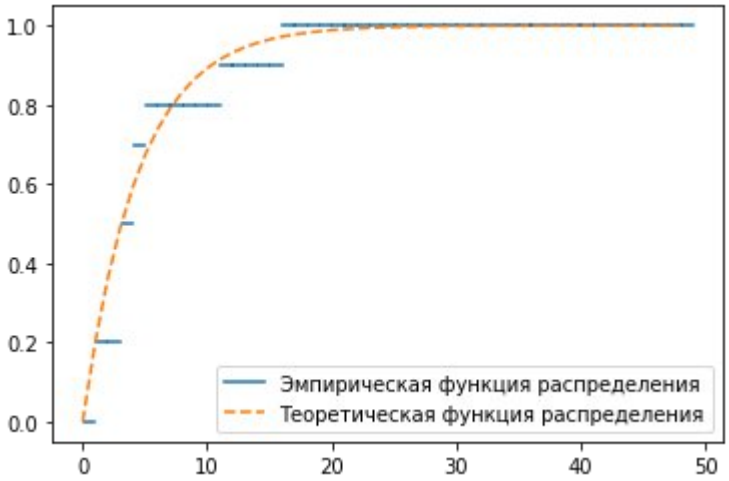

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

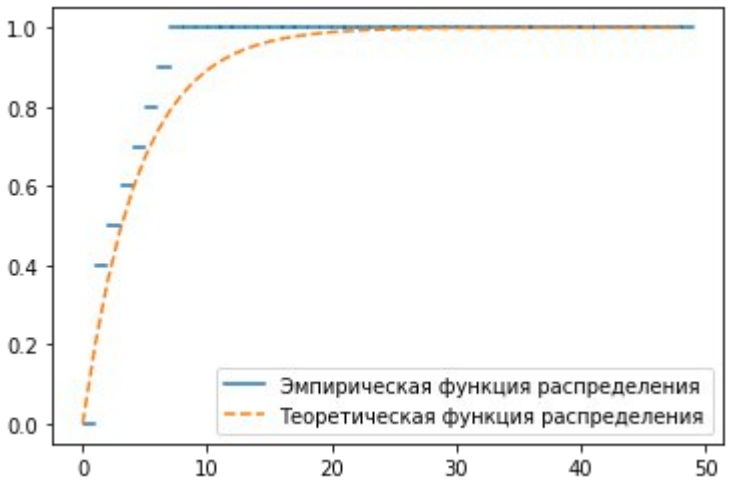

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

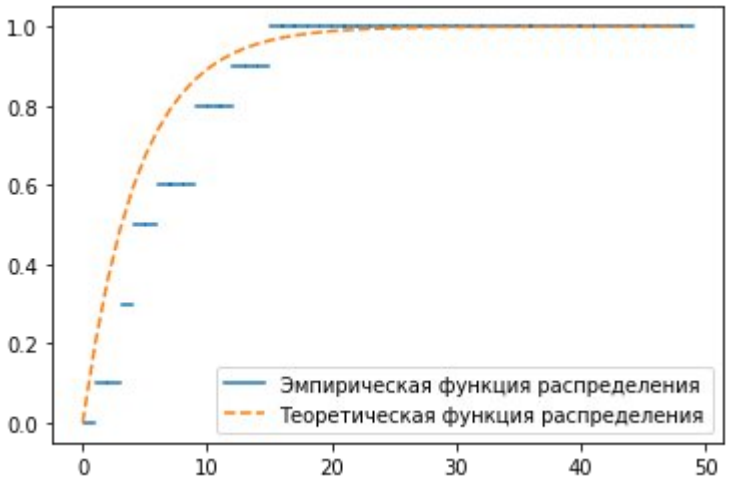

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

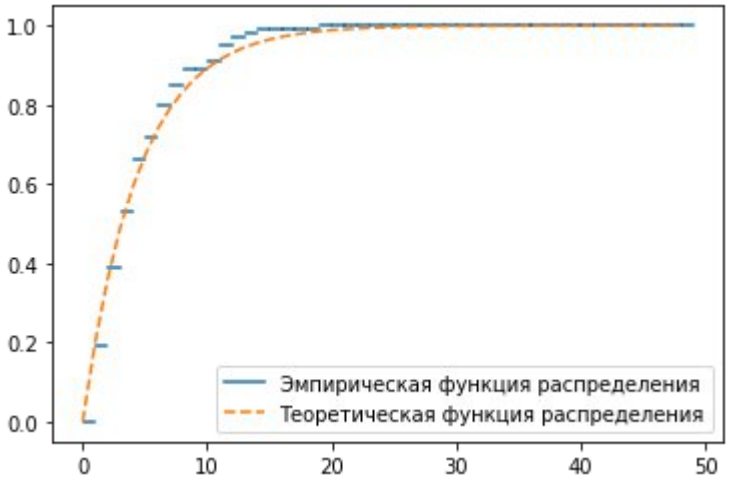

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

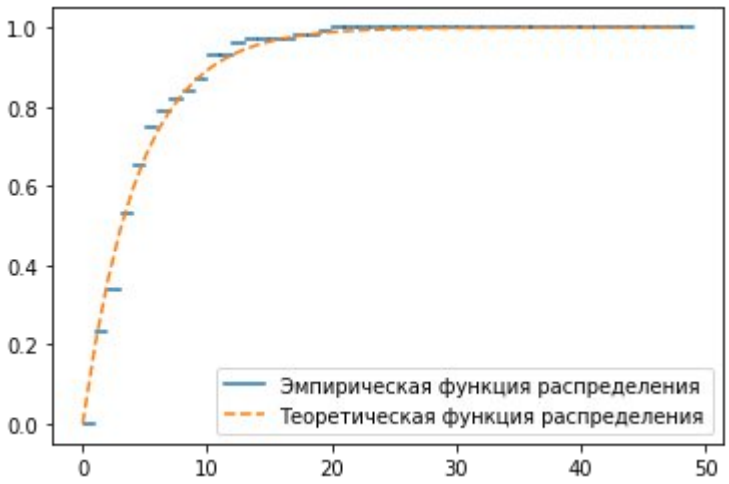

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

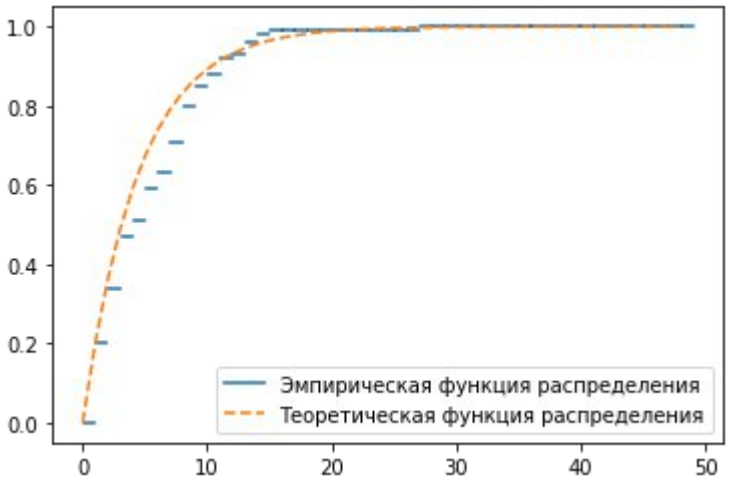

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

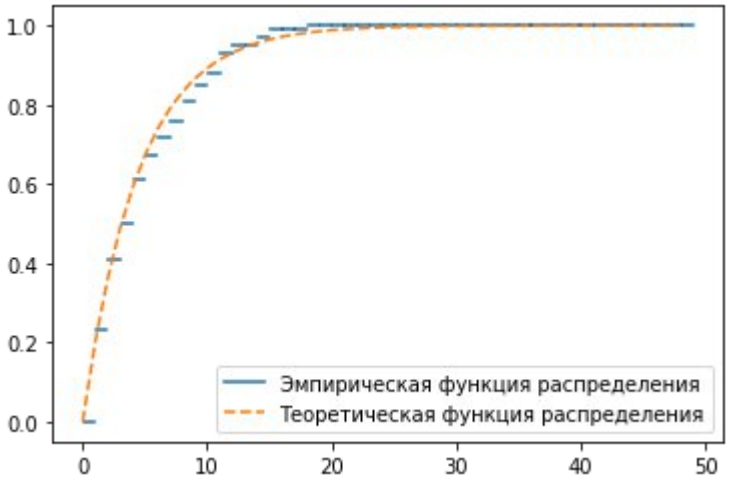

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

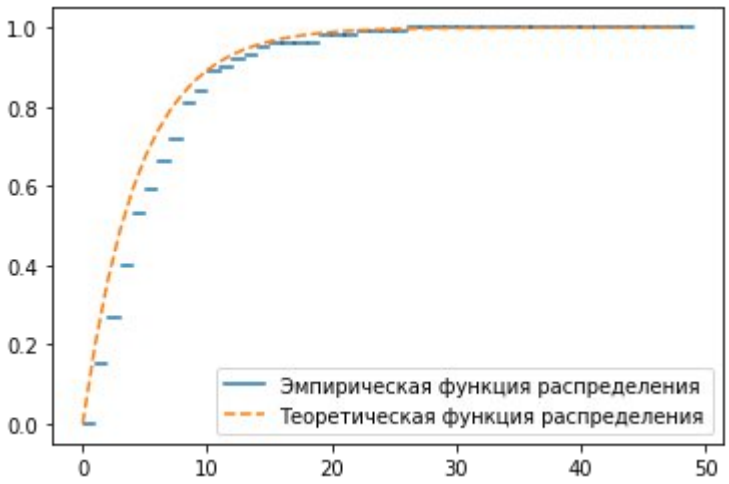

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

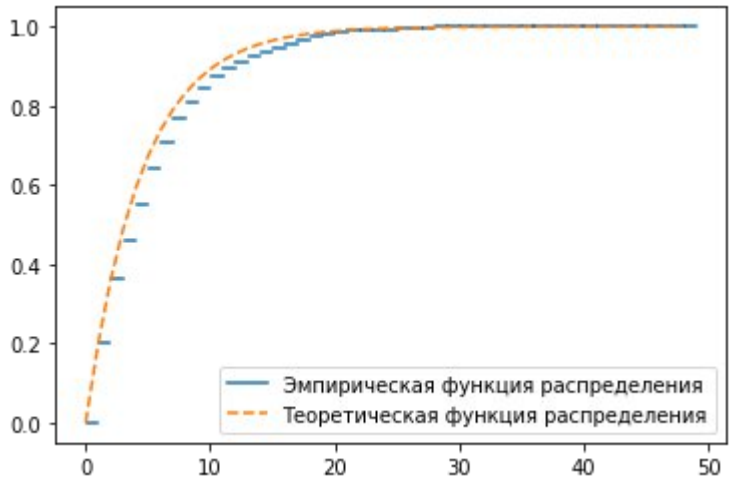

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

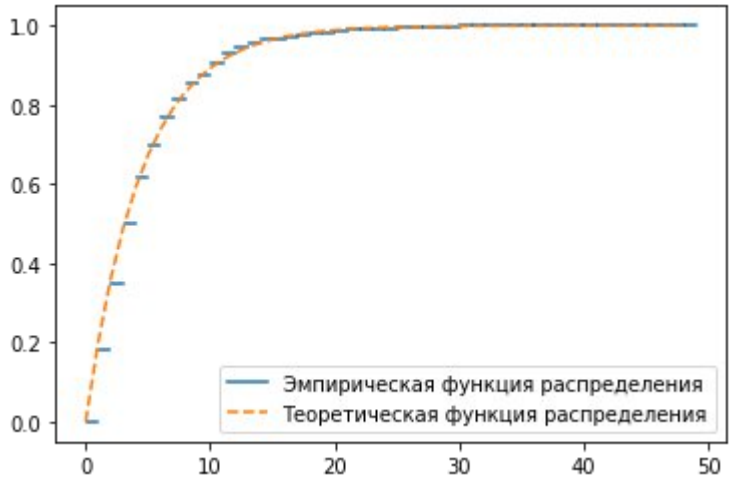

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

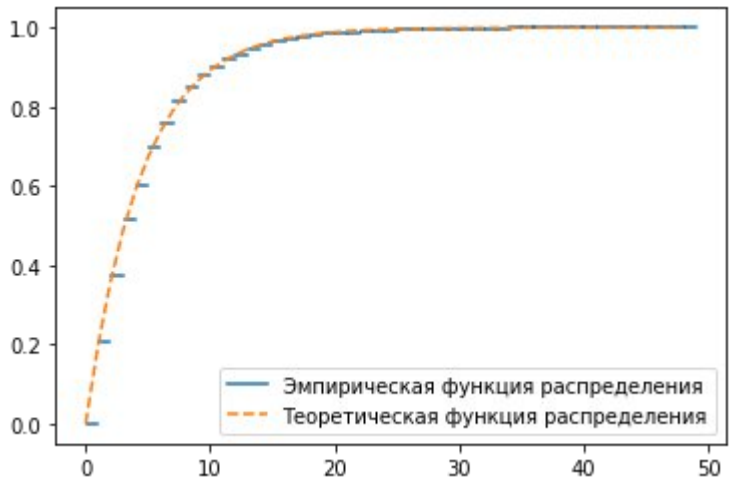

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

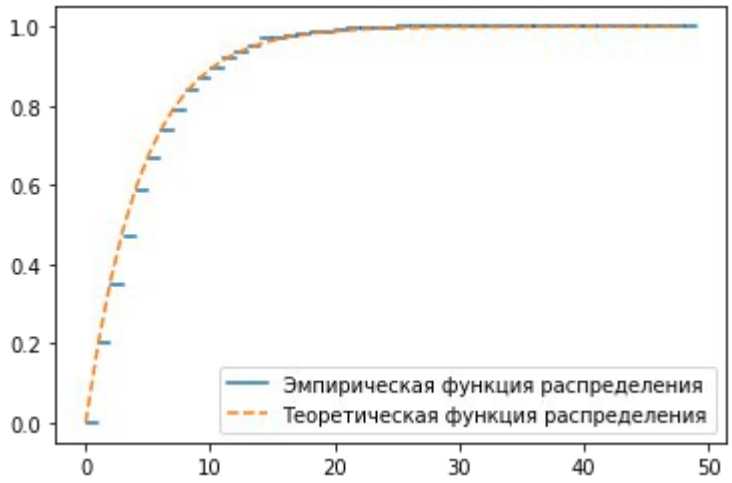

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

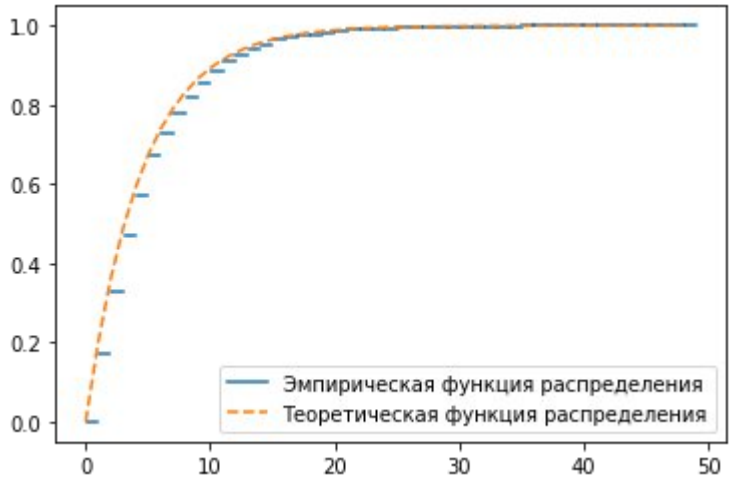

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

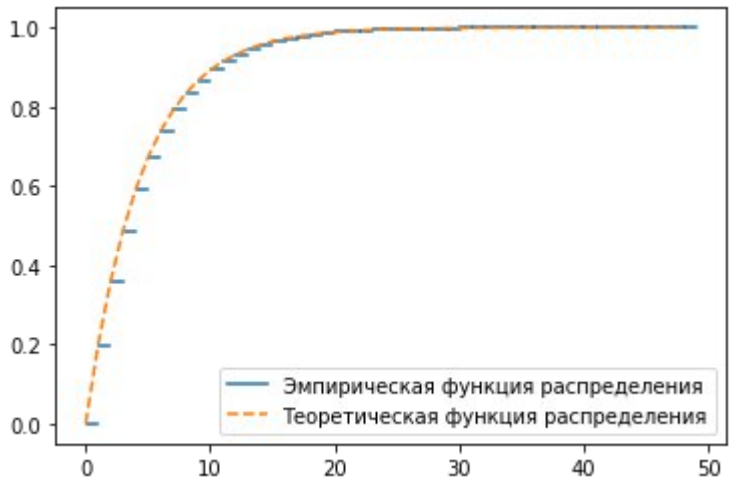

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

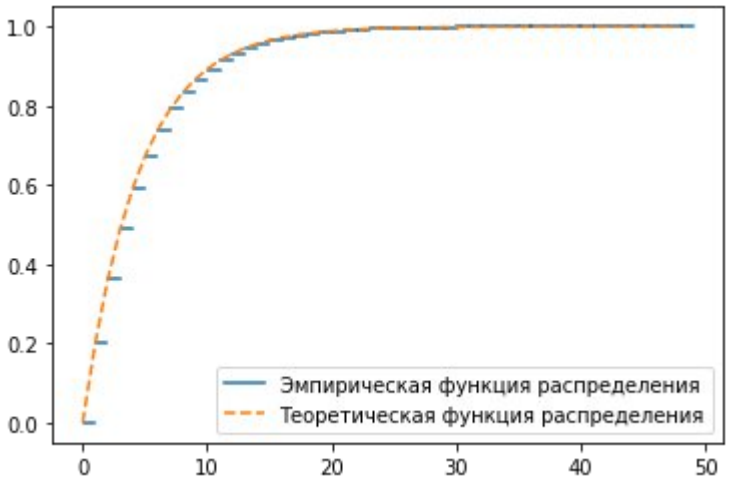

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

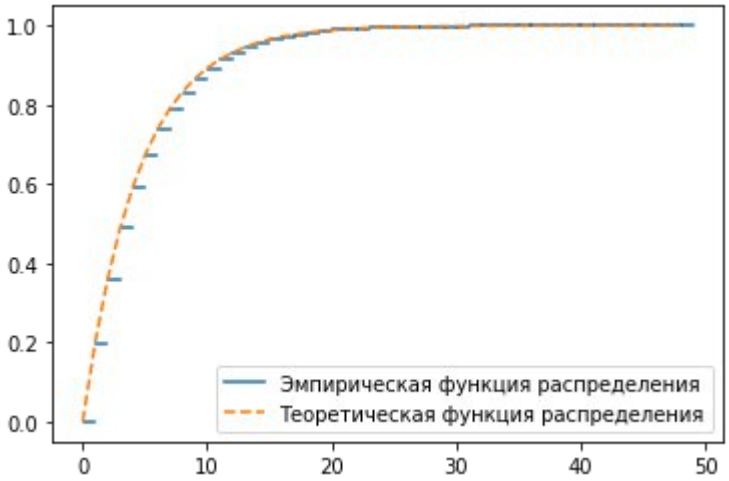

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

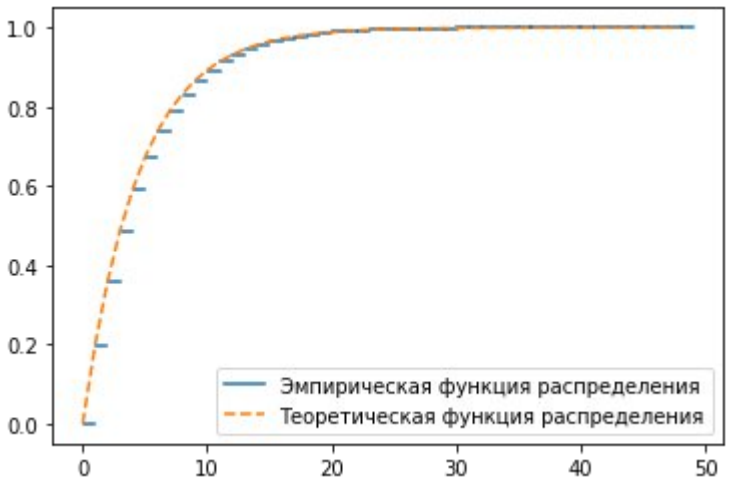

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

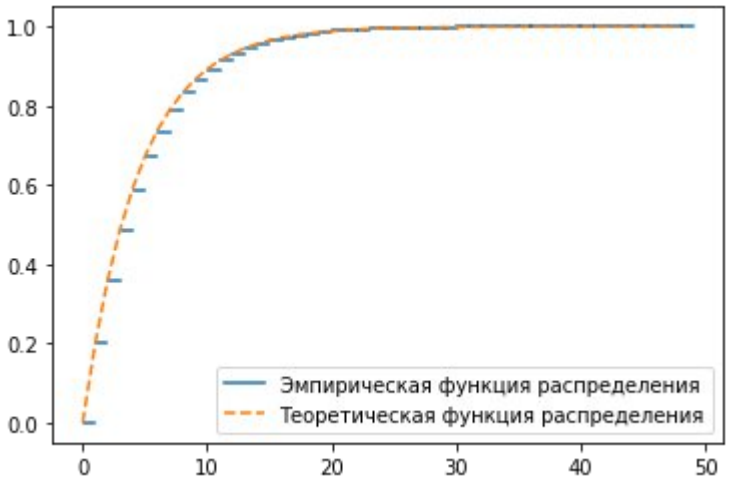

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения



Чудесно, мы добились ожидаемого результата! На графиках явно прослеживается, что с увеличением объёма выборки эмпирическая функция распределения начинает всё больше напоминать теоретическую функцию распределения.

Верхние границы разностей каждой пары эмпирических функций распределения

In [8]:
```
ECDFgeom = np.array(ECDFgeom).reshape(5, 5, 50)

for n in ECDFgeom:
    for i in range(len(n)-1):
        for j in range(i+1, len(n)):
            print('%.2f'%max(abs(n[i] - n[j])), end = '\n' if i == len(n)-2 e
lse ' ')
```

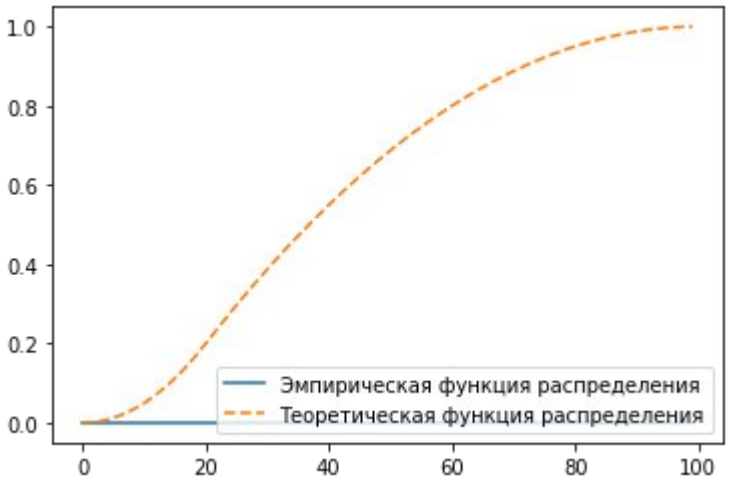
```
0.20 0.20 0.00 0.40 0.40 0.20 0.40 0.20 0.20 0.40
0.30 0.40 0.20 0.30 0.40 0.40 0.40 0.30 0.40 0.20
0.11 0.16 0.12 0.08 0.13 0.04 0.05 0.13 0.10 0.05
0.02 0.02 0.02 0.03 0.02 0.02 0.03 0.03 0.01 0.03
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

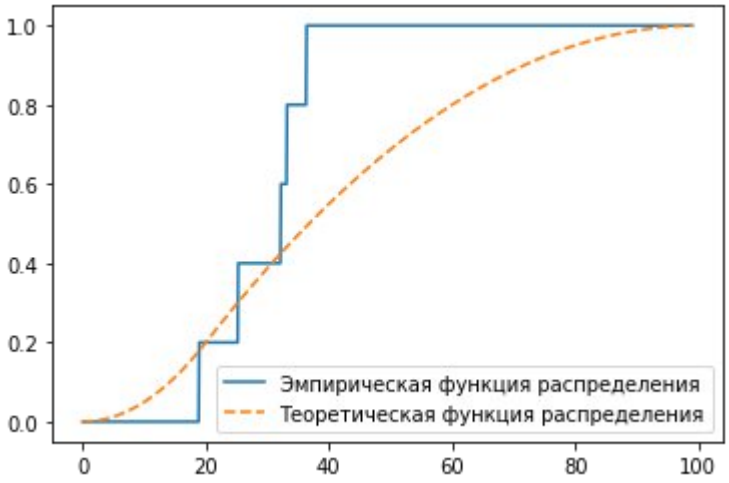
Отсюда явно видим, что с увеличением объёма выборки отклонение от теоретической функции распределения стремится к нулю. Это соответствует теореме Гливенко-Кантелли.

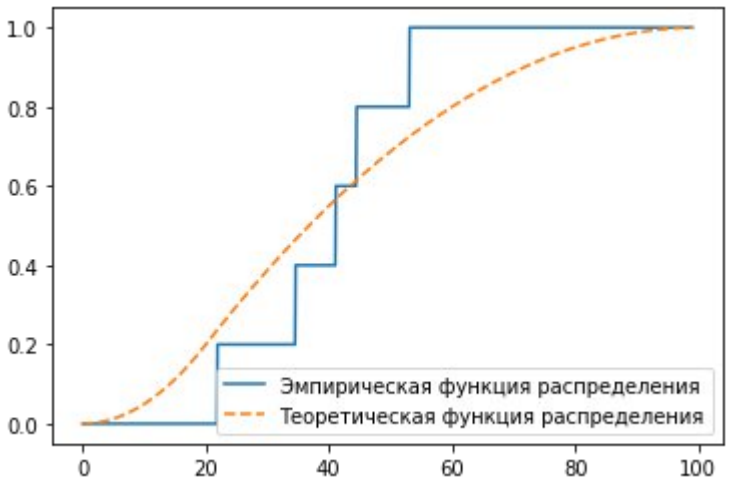
Для треугольного распределения

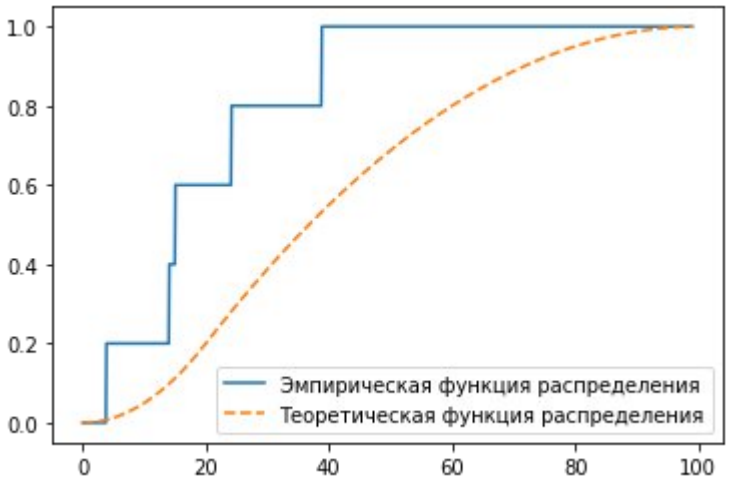
Аналогично с предыдущим распределением, эмпирическую функцию будем сравнивать с готовым методом triang из scipy.stats

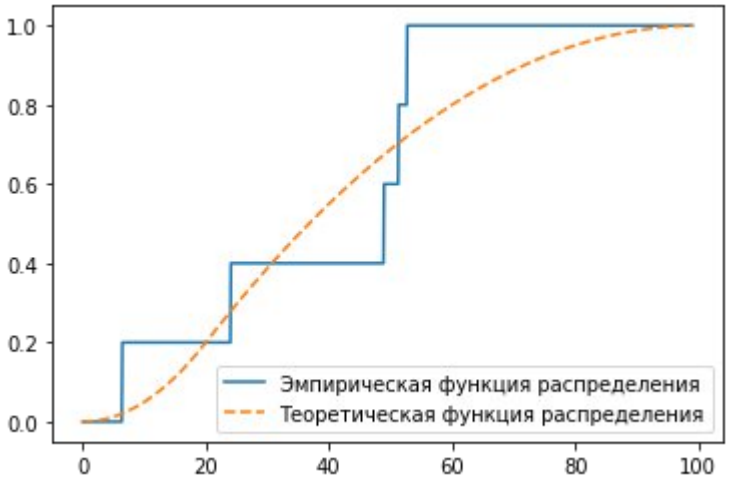
In [106]:
```
#y = lambda x: np.power(x,2)/0.2 if (x < 0.2) else -np.power(x,2)/0.8

ECDFtrig = []
xrange = 100
for samples in SampleTrig:
    for sample in samples:



        x1 = range(xrange)
        for i in range(len(sample)):
            sample[i] *= 100
        ecdf = ECDF(sample)
        x = np.linspace(0, xrange-1, 1000)
        l = []
        for i in x:
            l.append(ecdf(i))
        plt.plot(x, l, label='Эмпирическая функция распределения')
        plt.plot(x1, sts.triang.cdf(x1,0.2,0,100), label='Теоретическая функц
ия распределения', ls='--')
        plt.legend(loc='lower right')
        plt.show()
        for i in range(len(sample)):
            sample[i] /= 100
```

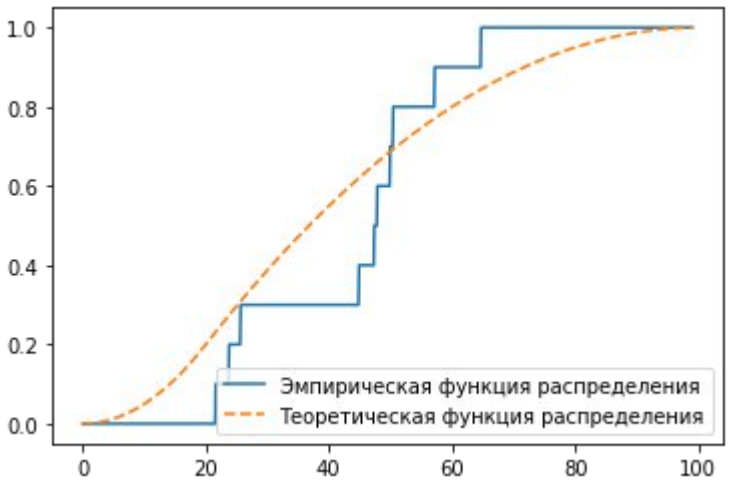

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

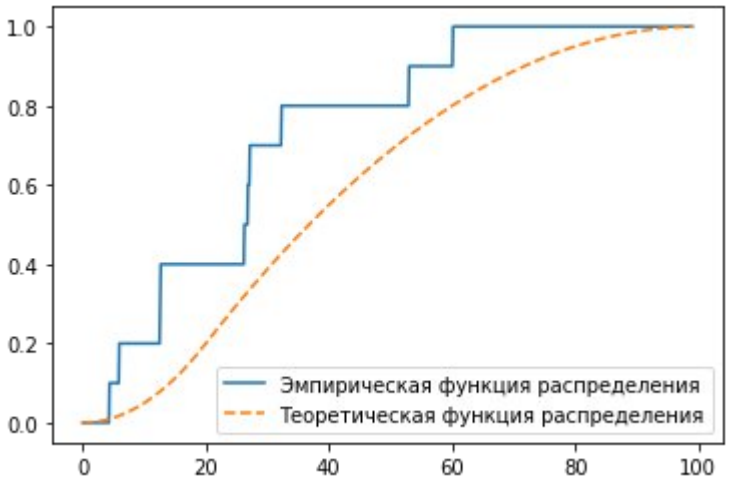

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

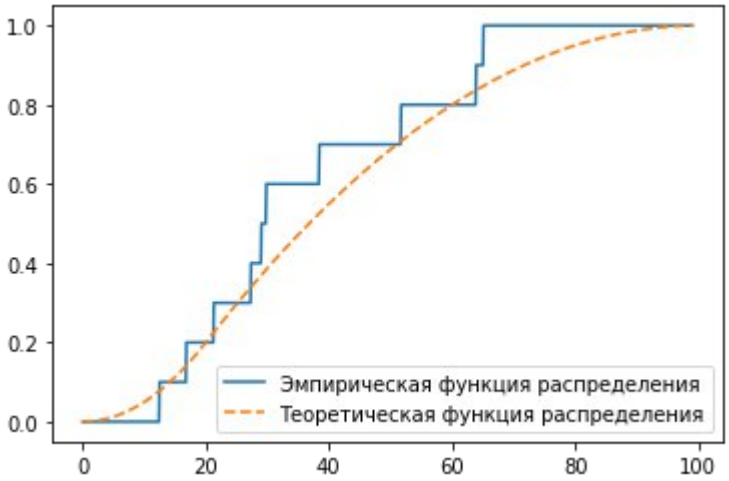

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

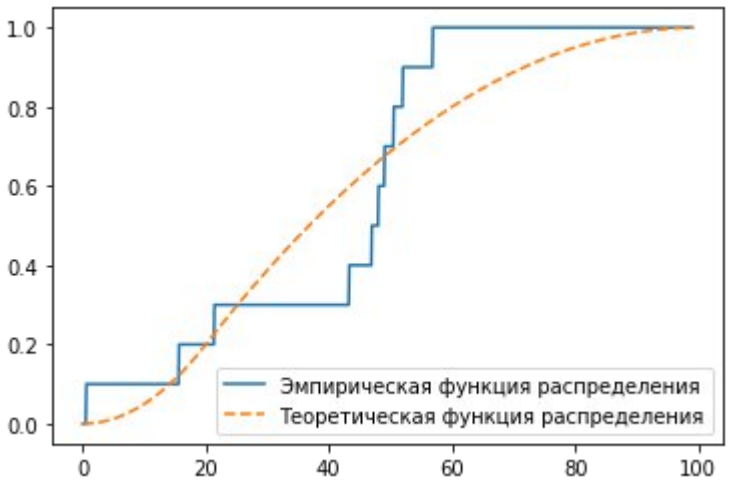

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

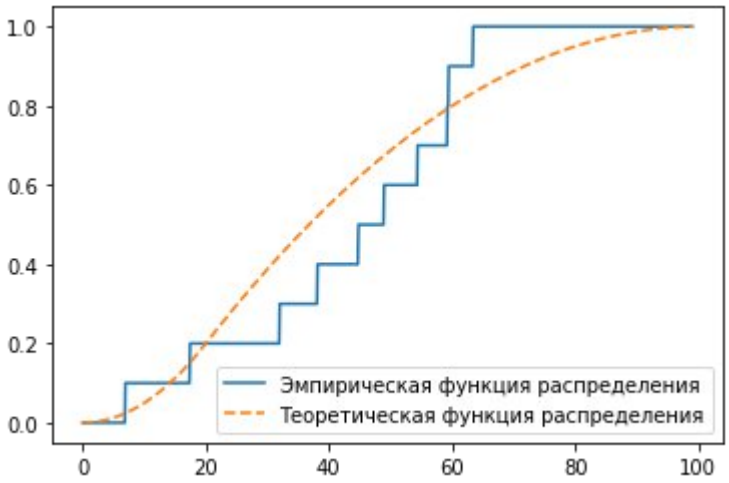

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

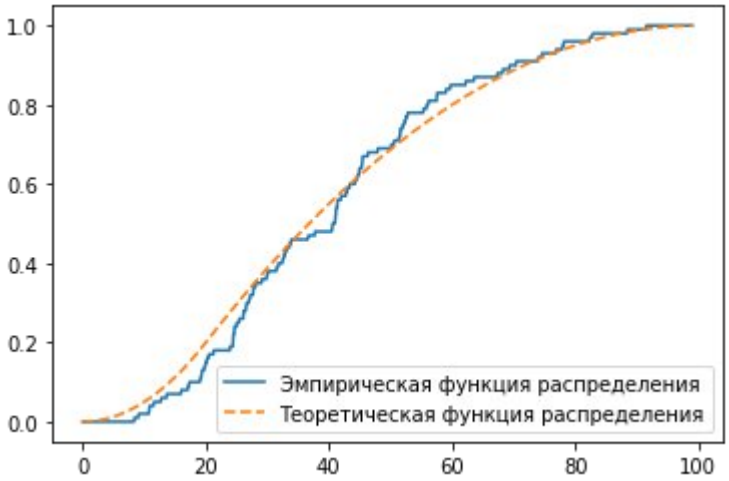

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

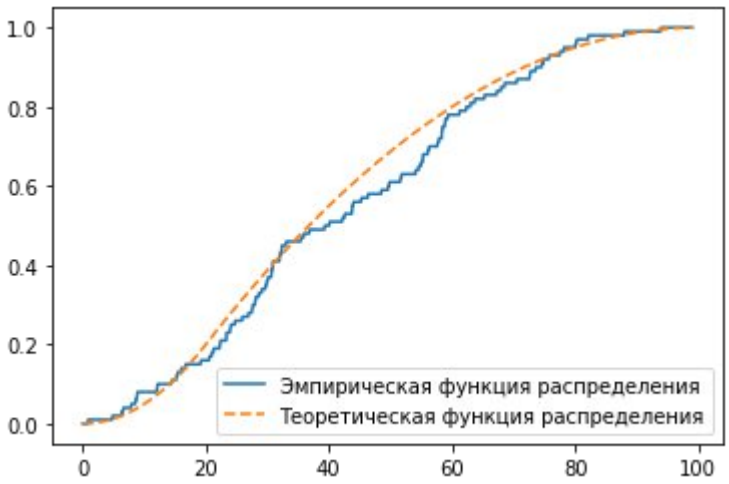

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

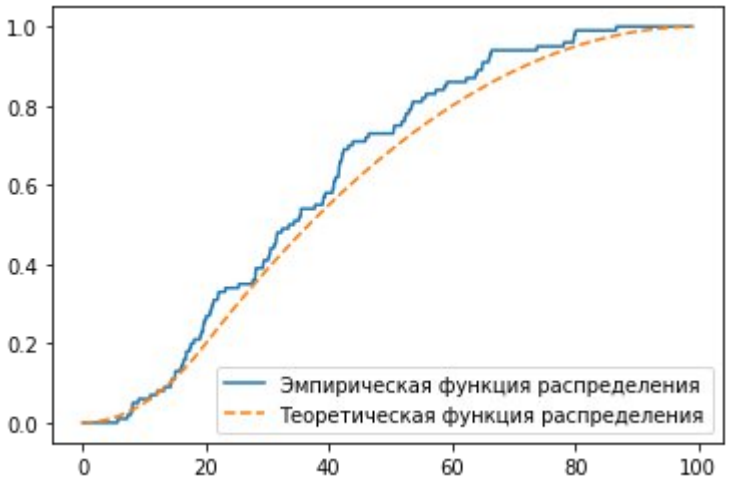

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

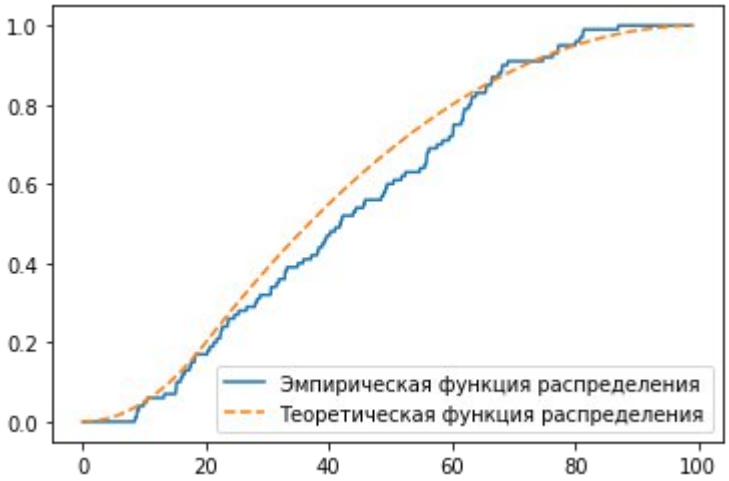

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

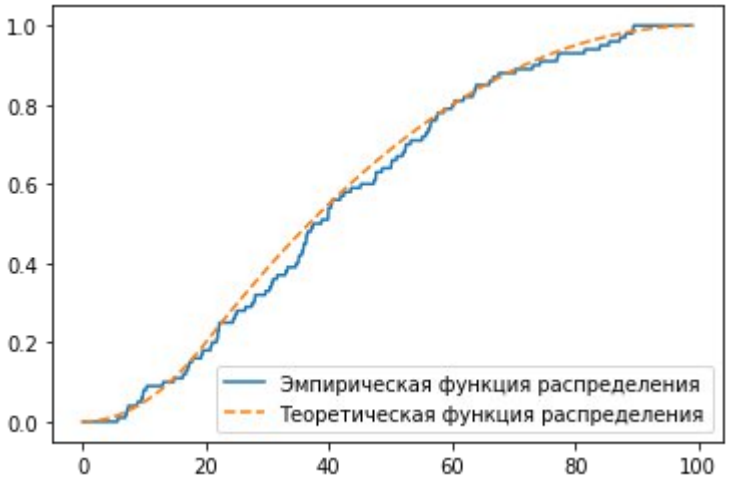

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

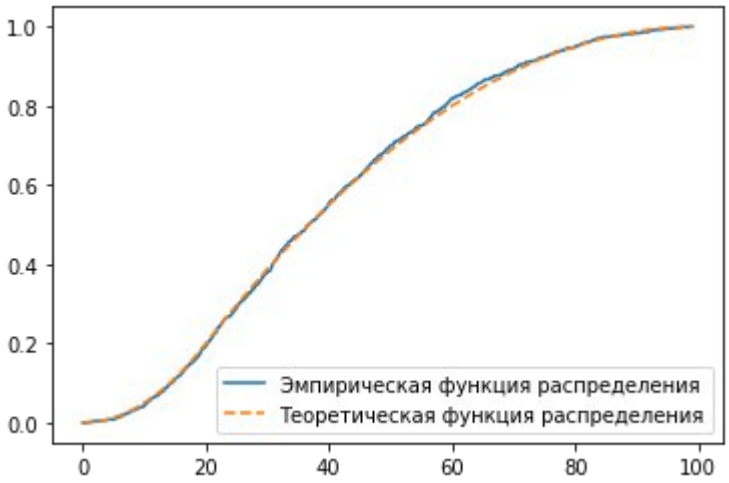

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

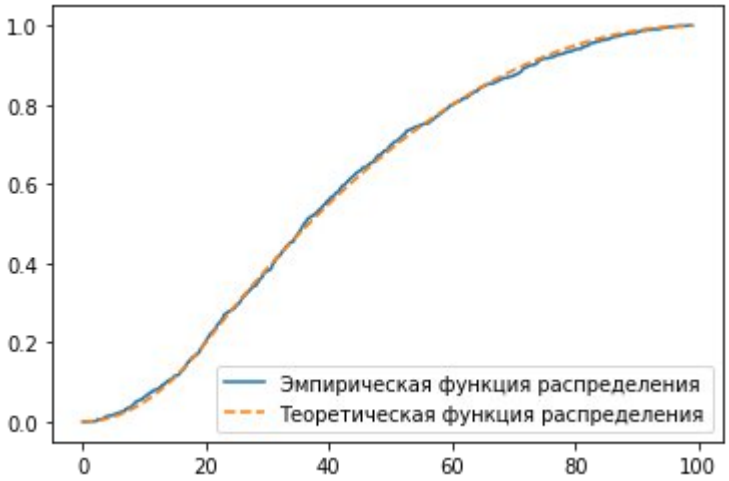

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

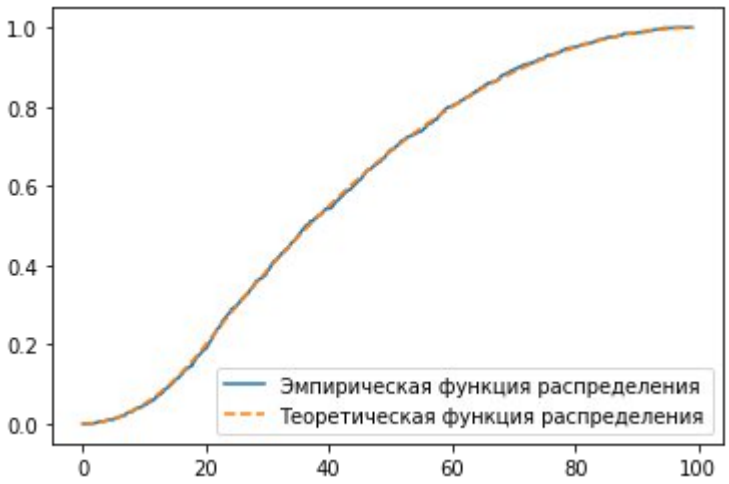

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

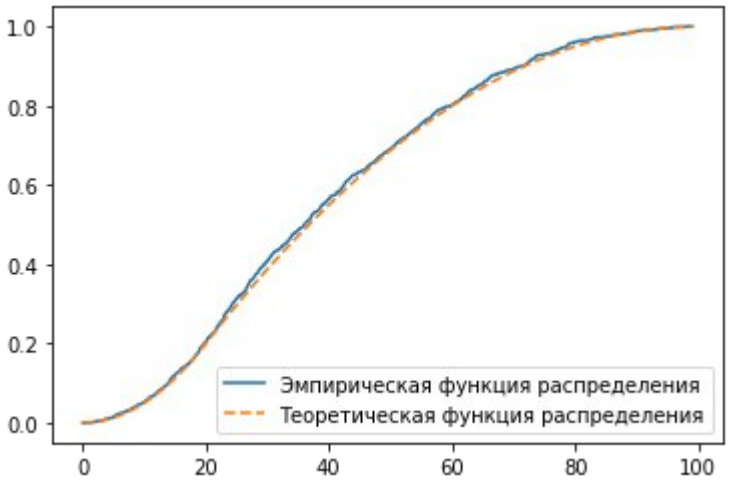

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

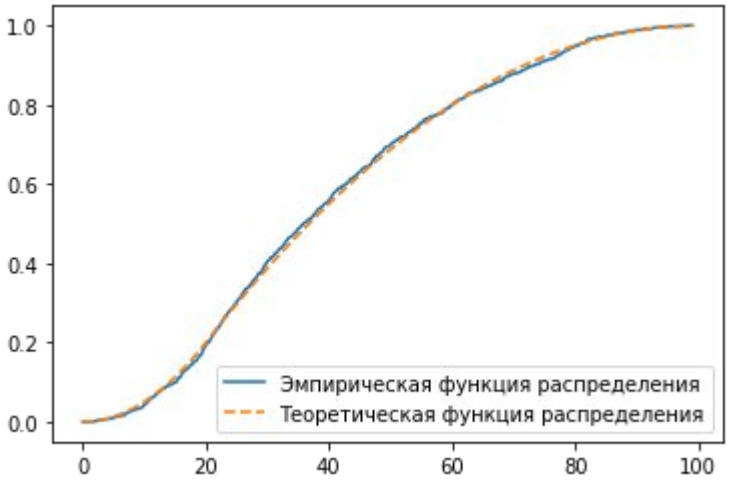

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

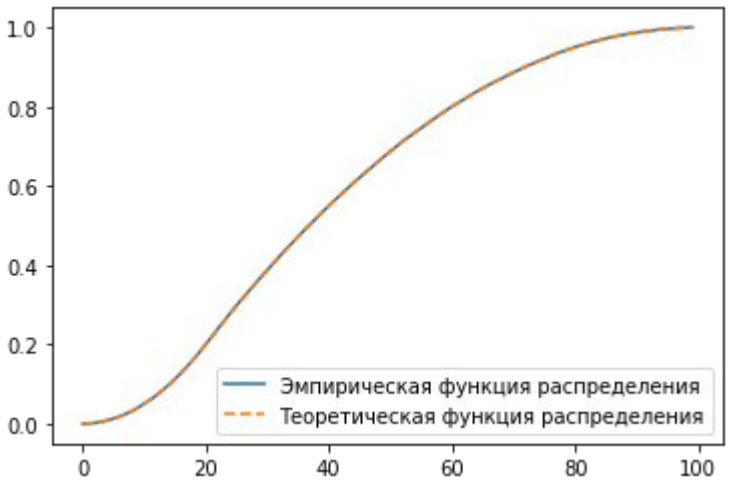

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

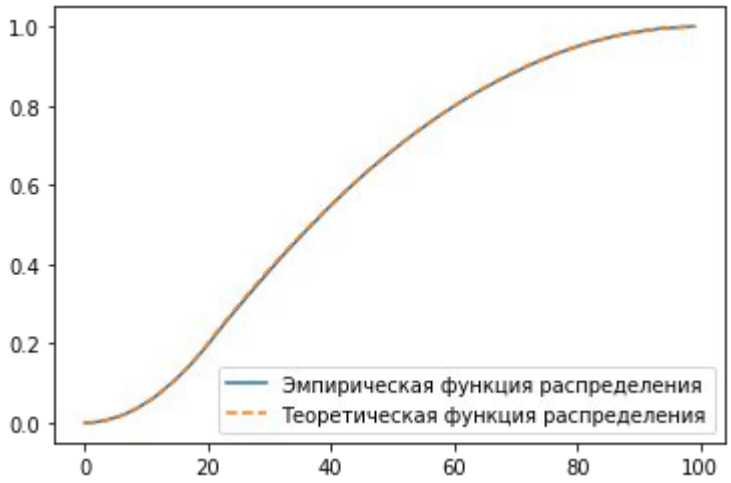

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

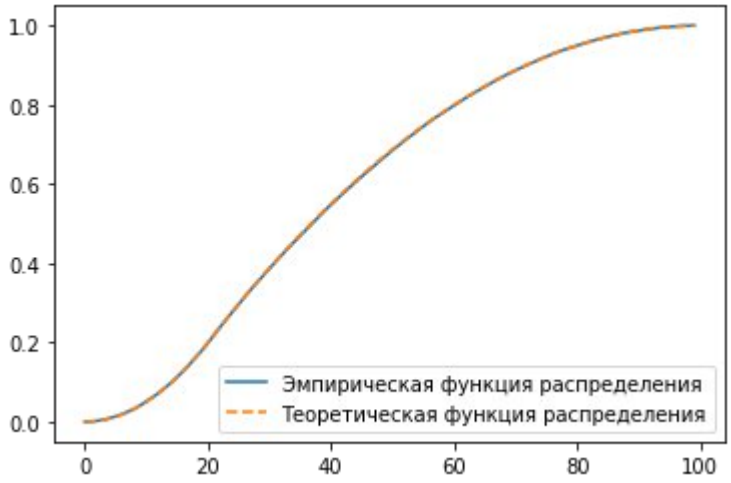

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

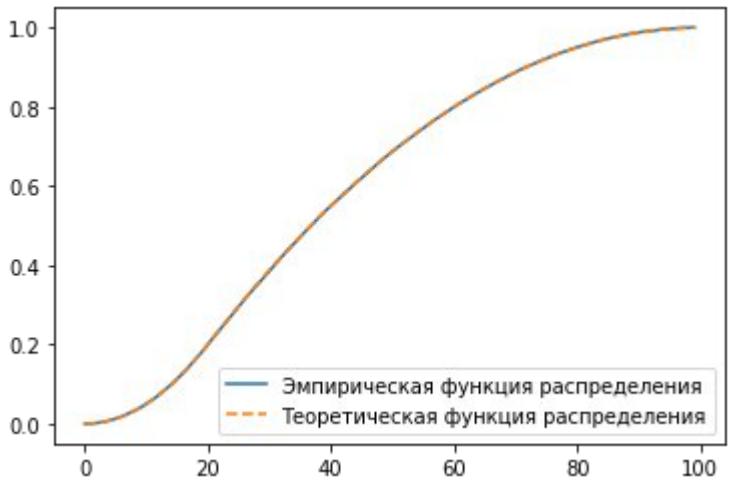

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения

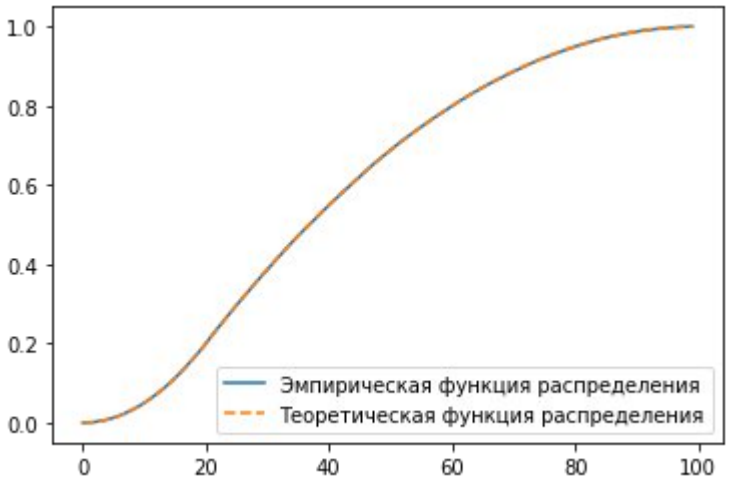

1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Эмпирическая функция распределения
Теоретическая функция распределения



Как и с предудущим распределением, с увеличением объёма выборки эмпирическая функция всё больше напоминает теоретическую.

Верхние границы разностей каждой пары эмпирических функций распределения

In [93]:
```
ECDFtrig = np.array(ECDFtrig).reshape(5, 5, 100)

for n in ECDFtrig:
    for i in range(len(n)-1):
        for j in range(i+1, len(n)):
            print('%.2f'%max(abs(n[i] - n[j])), end = '\n' if i == len(n)-2 e
lse ' ')
```

```
0.40 0.40 0.60 0.40 0.40 0.60 0.60 0.60 0.40 0.20
0.50 0.30 0.60 0.50 0.30 0.50 0.20 0.60 0.30 0.50
0.11 0.12 0.08 0.10 0.12 0.09 0.15 0.11 0.06 0.11
0.05 0.02 0.03 0.04 0.06 0.07 0.08 0.02 0.02 0.03
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

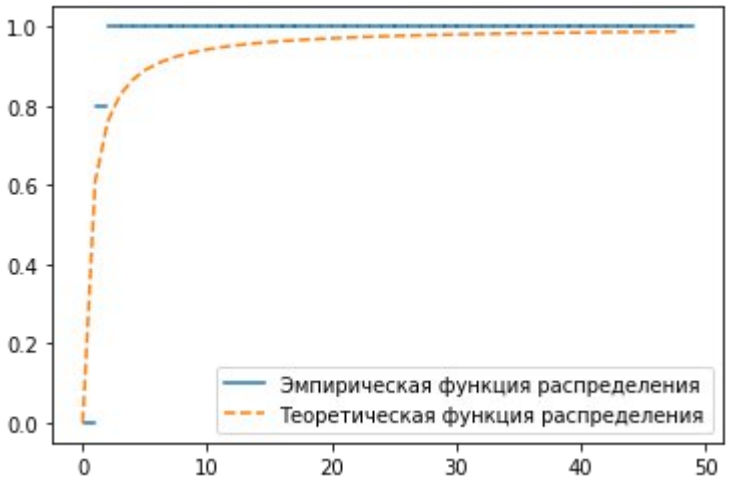
Отклонения от своего теоретического аналога становятся минимальны при увеличении объёма выборки.

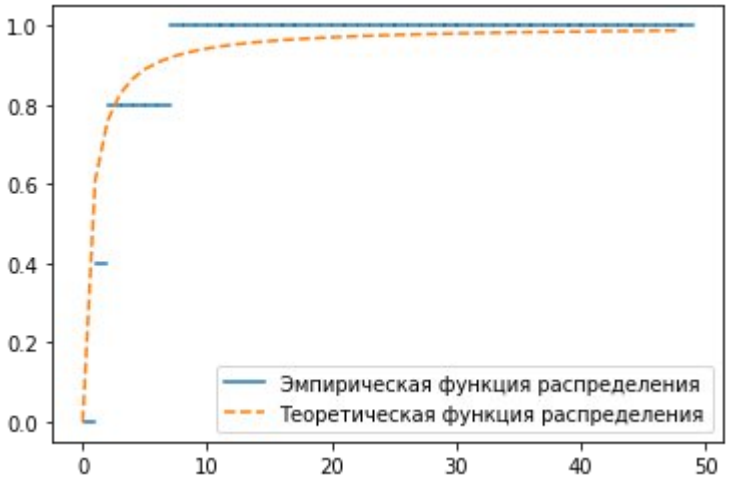
Для распределения Ципфа

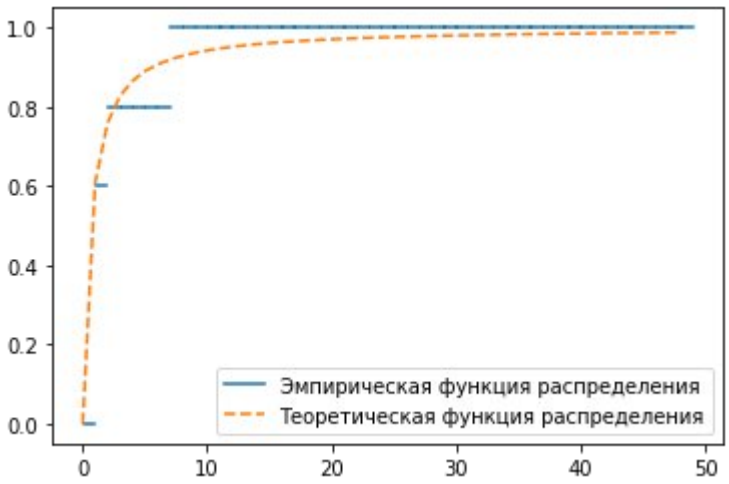
Для сравнения с теоретической функцией распределения воспользуемся готовым методом zipf из модуля scipy.stats

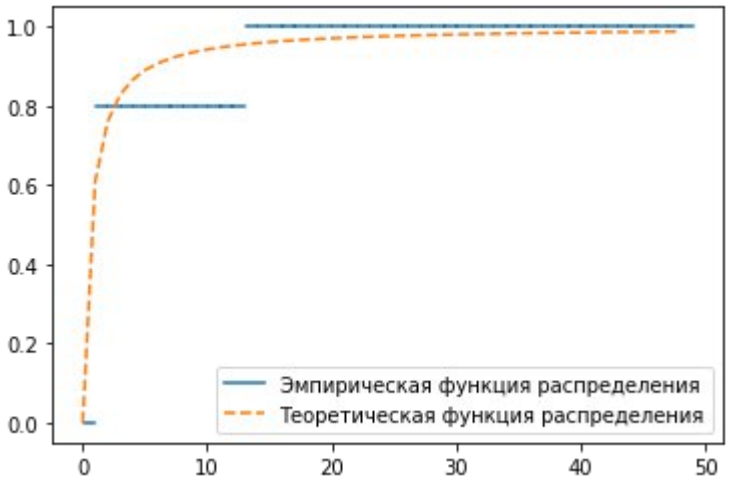
In [107]:

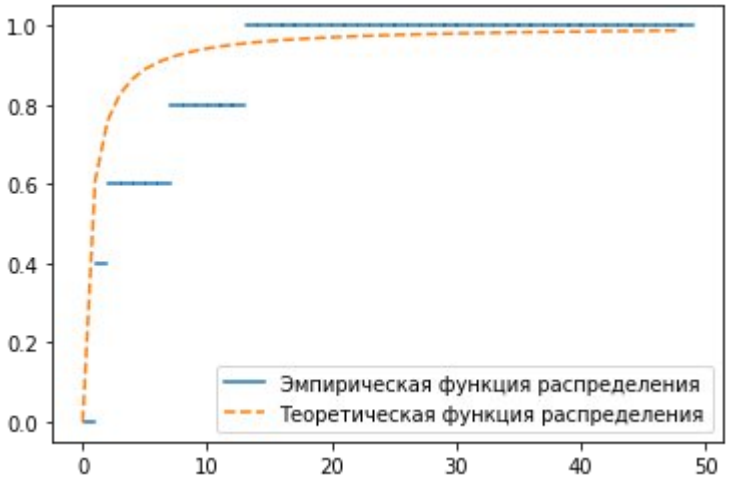
```
ECDFzipf = []
for samples in SampleZipf:
    for sample in samples:
        ecdf = ECDF(sample)
        x1 = range(0, 49)
        x = np.linspace(0,49,1000)
        l = []
        x[x % 1 < 0.05] = np.nan
        for i in x:
            l.append(ecdf(i))
        plt.plot(x, l, label='Эмпирическая функция распределения')
        plt.plot(x1, sts.zipf.cdf(x1, 2), label='Теоретическая функция распре
деления', ls='--')
        plt.legend(loc='lower right')
        plt.show()
```

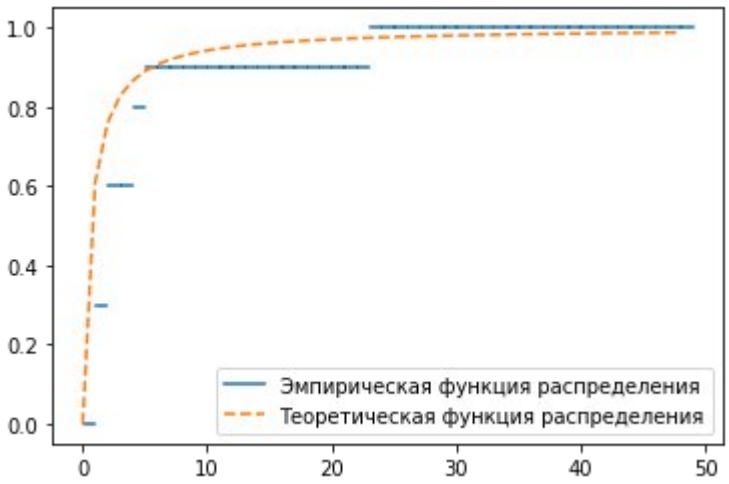

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

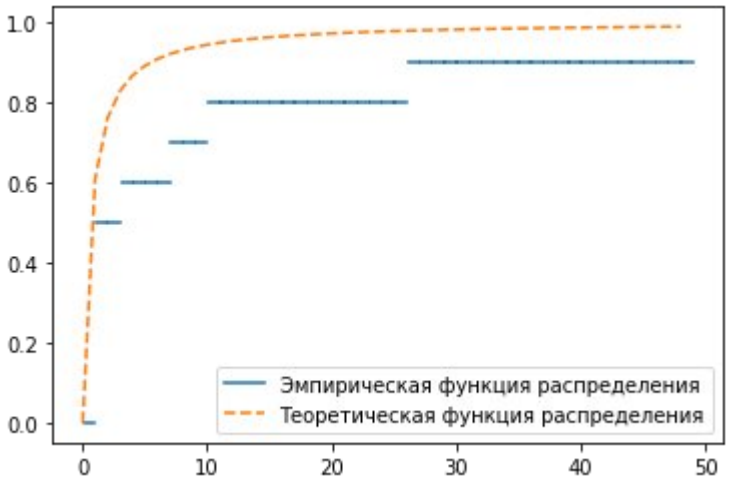

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

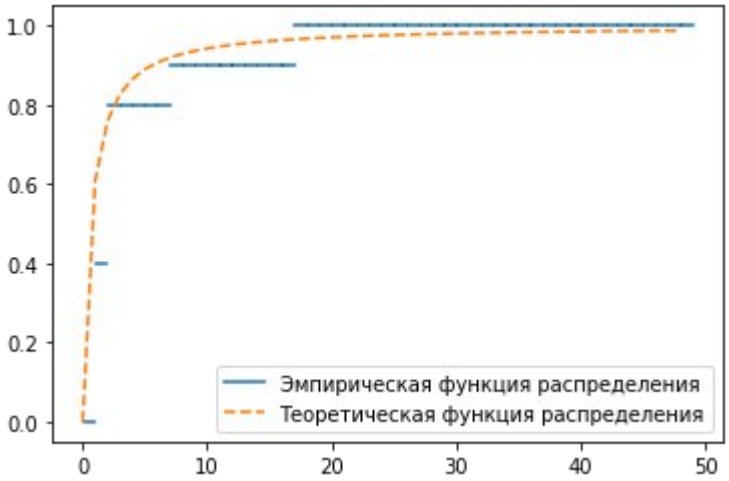

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

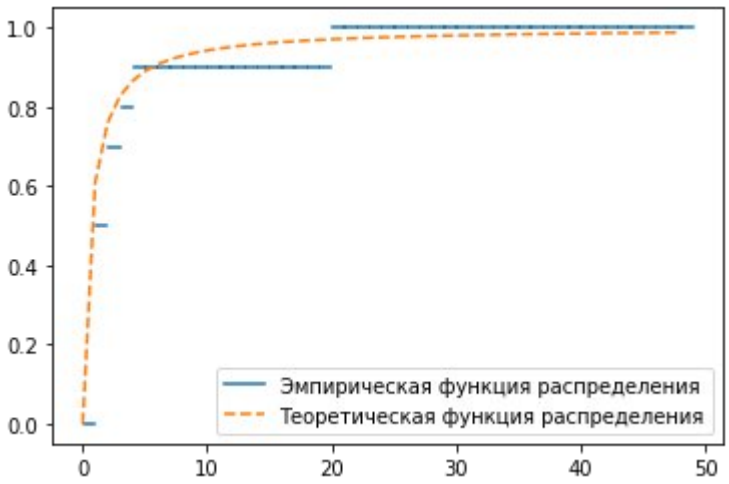

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

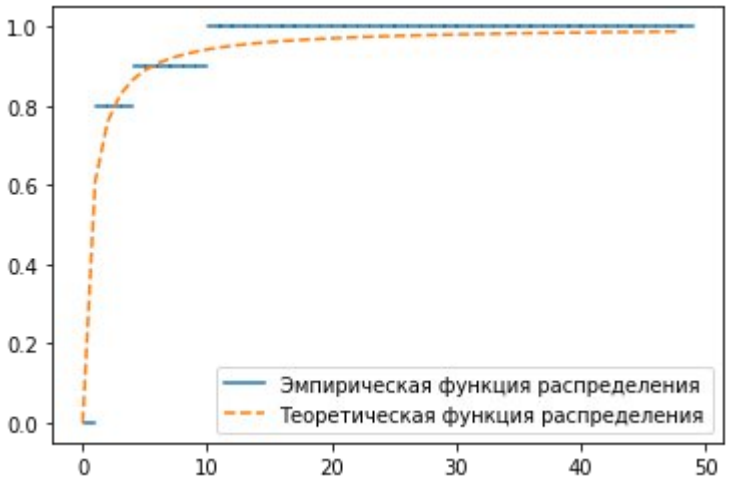

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

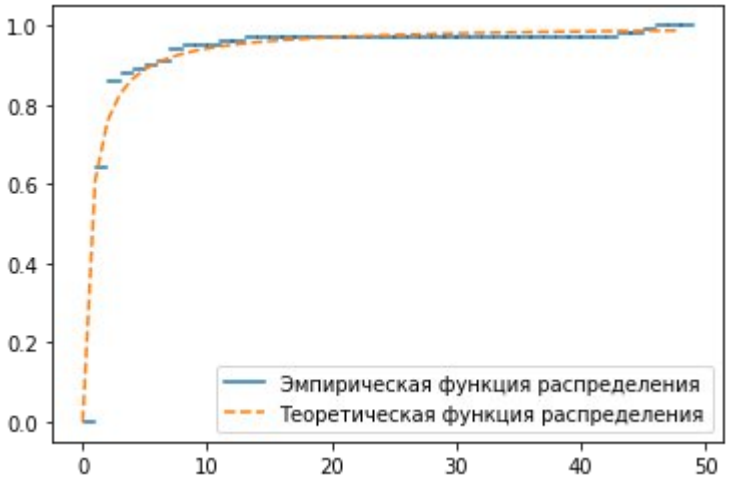

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

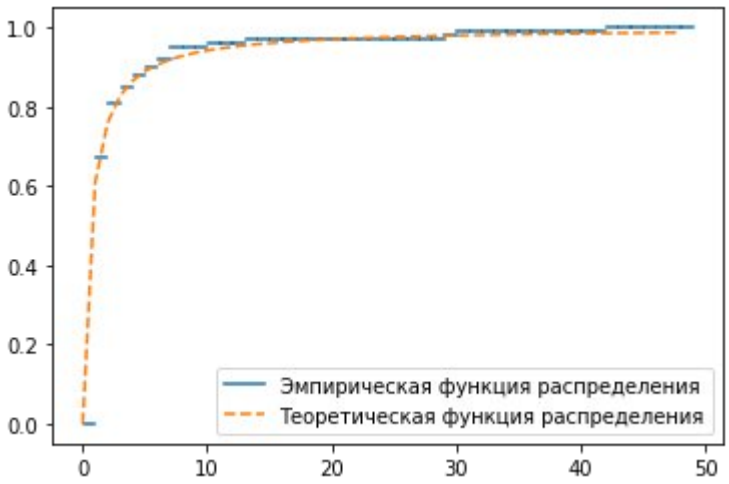

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

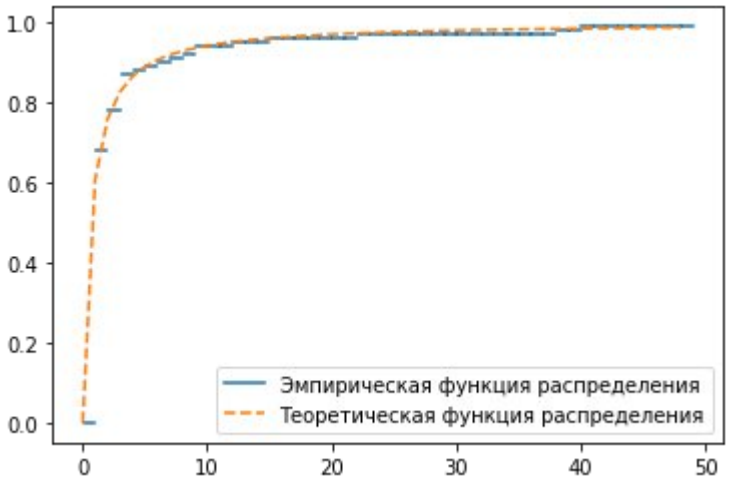

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

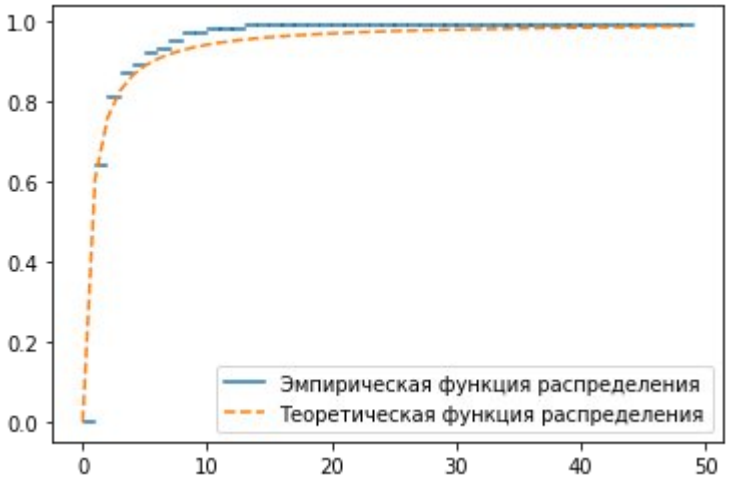

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

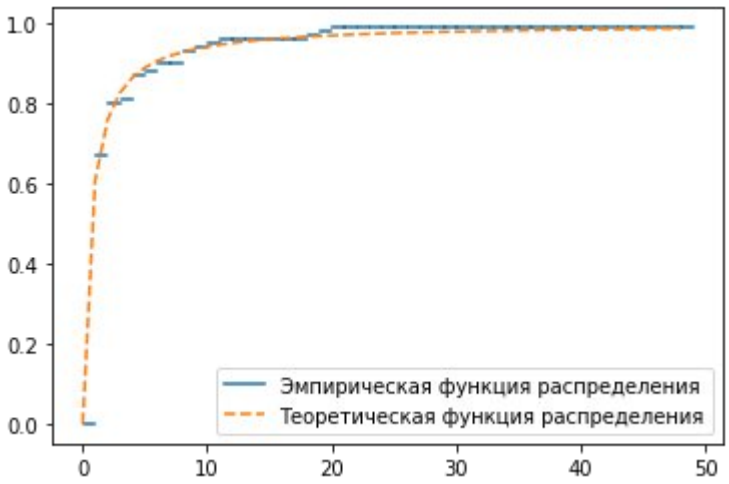

Эмпирическая функция распределения
Теоретическая функция распределения

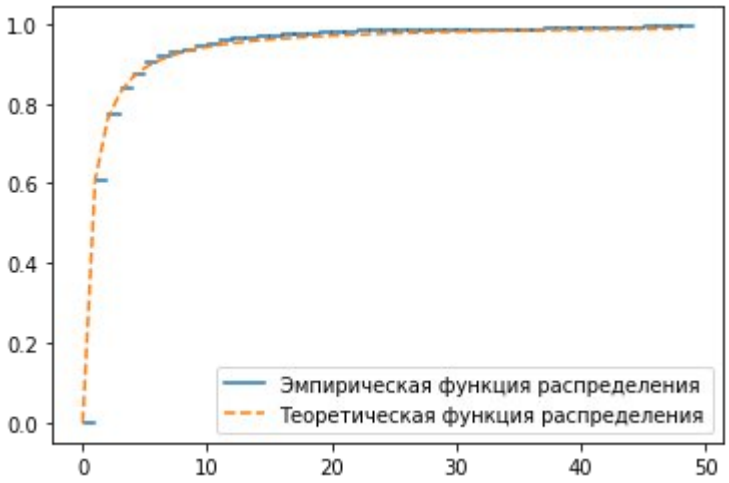

Эмпирическая функция распределения
Теоретическая функция распределения

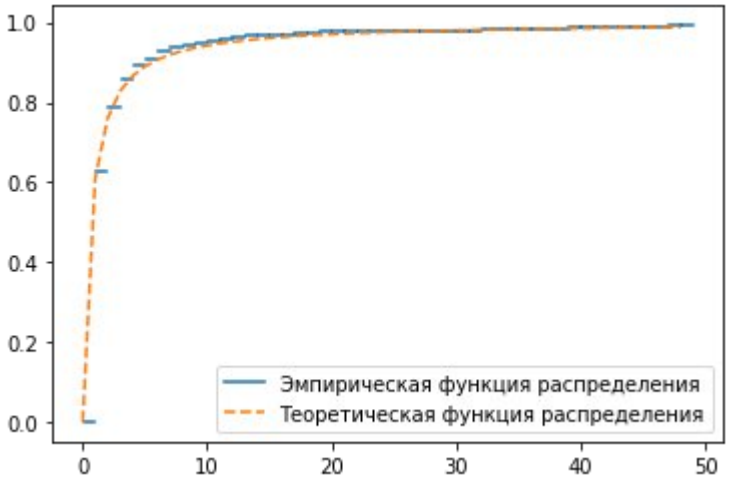

Эмпирическая функция распределения
Теоретическая функция распределения

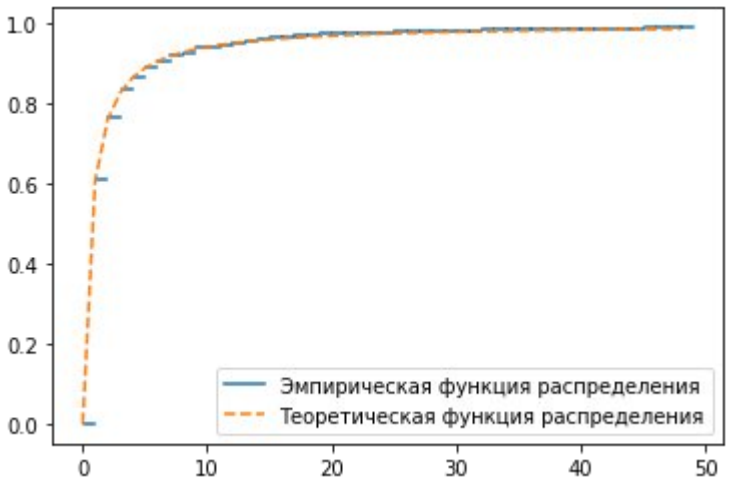

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

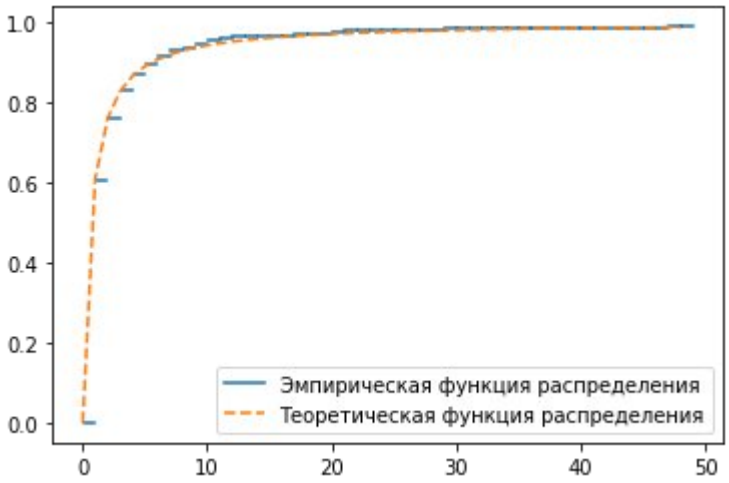

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

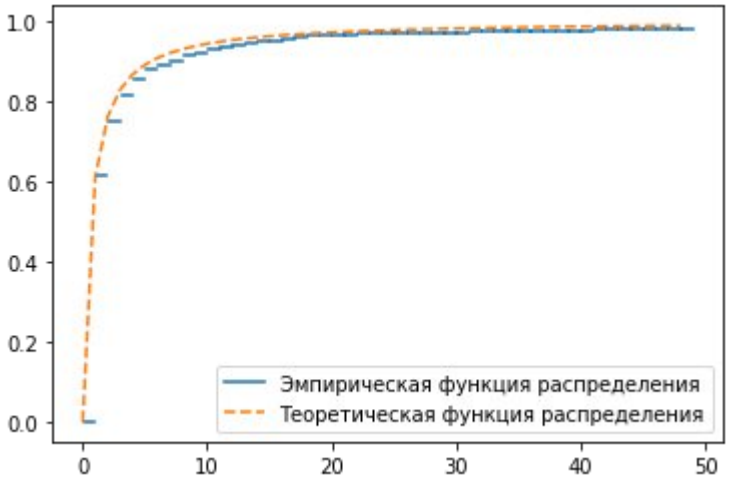

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

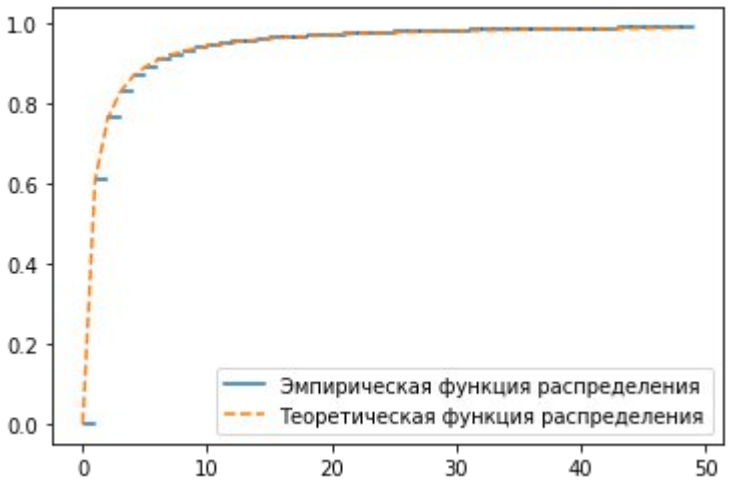

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

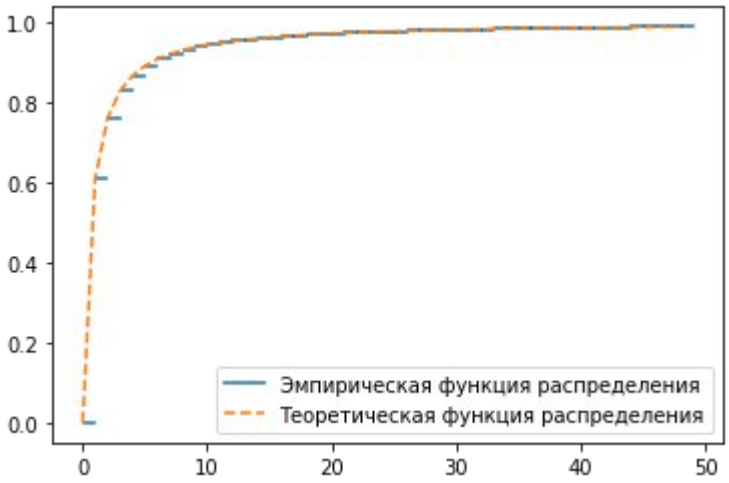

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

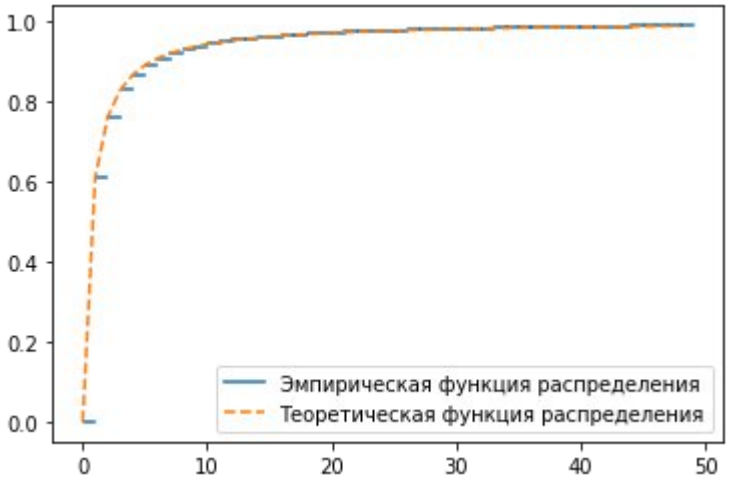

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

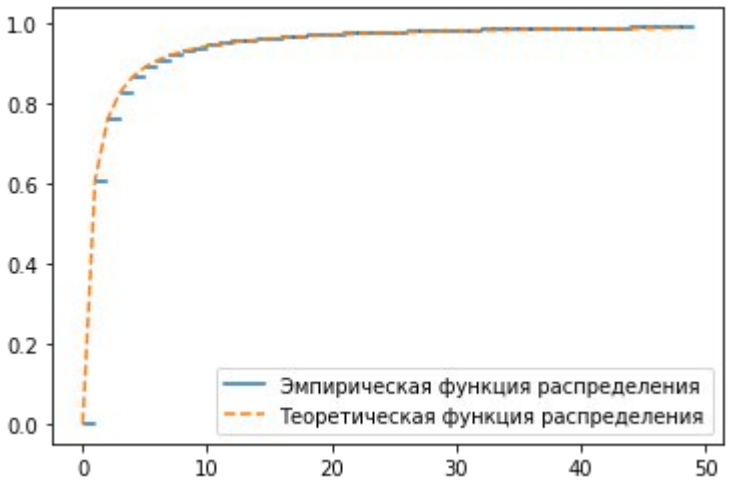

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения

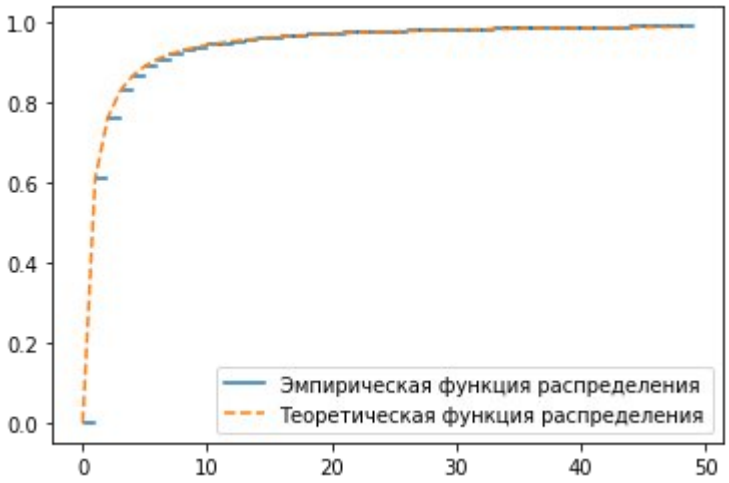

1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Эмпирическая функция распределения
Теоретическая функция распределения



Верхние границы разностей каждой пары эмпирических функций распределения

In [12]:
```
ECDFzipf = np.array(ECDFzipf).reshape(5, 5, 50)

for n in ECDFzipf:
    for i in range(len(n)-1):
        for j in range(i+1, len(n)):
            print('%.2f'%max(abs(n[i] - n[j])), end = '\n' if i == len(n)-2 e
lse ' ')
```

```
0.20 0.40 0.40 0.20 0.20 0.20 0.40 0.20 0.60 0.60
0.40 0.40 0.30 0.30 0.30 0.30 0.10 0.20 0.30 0.30
0.07 0.09 0.08 0.13 0.09 0.06 0.10 0.05 0.04 0.07
0.02 0.03 0.02 0.02 0.05 0.03 0.04 0.03 0.03 0.02
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

На данном примере мы убедились, что эмпирическая функция по мере увеличения объёма выборки начинает всё больше сходиться к своему теоретическому аналогу. Это же подтверждается теоремой Гливенко-Кантелли.

2.3. Построение вариационного ряда выборки

Для геометрического распределения

In [5]:
```
volumes = (5, 10, 100, 1000, 100000)
VarRangeGeom = [[[np.sort(SampleGeom[N][i])] for i in range(5)] for N in rang
e(len(volumes))]
```

In [7]:
```
print(*VarRangeGeom[0][0])
print(*VarRangeGeom[1][1])
```

```
[ 1  2  3  3 10]
[ 4  4  5  6  7  7  9 10 11 23]
```

Выборочные квантили

In [6]:
```
for samples in SampleGeom:
    print(*np.quantile(samples, [0.1,0.5,0.7], axis=1).T)
```

```
[1.  1.  3.4] [1.  1.  2.6] [1.4 4.  4. ] [1.  1.  4.2] [1. 3. 3.]
[1.  3.5 4. ] [1. 4. 6.] [1.  3.  6.3] [1.  3.  4.3] [2.  4.5 6. ]
[1. 4. 6.] [1.  4.  5.3] [1. 4. 6.] [1.  3.  5.3] [1. 4. 6.]
[1.  4.  6.3] [1. 4. 6.] [1. 4. 6.] [1. 4. 6.] [1. 4. 6.]
[1. 4. 6.] [1. 4. 6.] [1. 4. 6.] [1. 4. 6.] [1. 4. 6.]
```

Для подсчёта теоретических квантилей воспользуемся готовым методом, с которым мы уже сталкивались ранее

In [12]:
```
print(sts.geom.ppf([0.1,0.5,0.7], 0.2))
```

```
[1. 4. 6.]
```

Как видно из получаемых значений, с увеличением объёма выборки квантили стремятся к своим теоретическим значениям.

Для треугольного распределения

Аналогично создаём выборочный ряд и квантили

In [17]:
```
volumes = (5, 10, 100, 1000, 100000)
VarRangeTrig = [[[np.sort(SampleTrig[N][i])] for i in range(5)] for N in rang
e(len(volumes))]

print('Примеры вариационных рядов:\n', *VarRangeTrig[0][0])
print(*VarRangeTrig[1][1], end = '\n\n\n')

print('Выборочные квантили:')
for samples in SampleTrig:
    print(*np.quantile(samples, [0.1,0.5,0.7], axis=1).T)

print()

print('Теоретические квантили:\n', sts.triang.ppf([0.1,0.5,0.7], 0.2))
```

```
Примеры вариационных рядов:
 [0.24958479 0.25051306 0.27190341 0.47876349 0.83106308]
[0.14042063 0.22019568 0.28353523 0.30875667 0.54063035 0.55920782
 0.64075965 0.65878326 0.70081295 0.92725918]


Выборочные квантили:
[0.2499561  0.27190341 0.43739147] [0.21794306 0.52513512 0.53915355] [0.3273
0417 0.53282045 0.7686411 ] [0.18531851 0.25461221 0.50600873] [0.24825121 0.
56837799 0.64636347]
[0.12117926 0.22701568 0.23850311] [0.21221818 0.54991908 0.64616673] [0.1585
736  0.50567717 0.61800657] [0.1910522  0.37180251 0.43853409] [0.25982527 0.
39538844 0.49437051]
[0.12833939 0.37182302 0.46520037] [0.13330847 0.34665228 0.44807169] [0.1659
2833 0.4093677  0.54729337] [0.14735819 0.38341805 0.53118823] [0.1545365  0.
37334054 0.51629293]
[0.14358836 0.35150633 0.49725568] [0.14099519 0.35356297 0.50069622] [0.1434
6164 0.38702622 0.52474898] [0.1396086  0.34873516 0.50569464] [0.13473996 0.
36741813 0.49836587]
[0.14118705 0.36627153 0.50903416] [0.14118728 0.36686175 0.51077375] [0.1416
5962 0.36770507 0.50929987] [0.14059934 0.36657939 0.50924759] [0.1421467  0.
36865147 0.51104503]

Теоретические квантили:
 [0.14142136 0.36754447 0.51010205]
```

Аналогично с предыдущим распределением, с увеличением объёма выборки квантили стремятся к своим теоретическим значениям.

Для распределения Ципфа

И вновь создаём выборочный ряд и квантили

In [18]:
```
volumes = (5, 10, 100, 1000, 100000)
VarRangeZipf = [[[np.sort(SampleZipf[N][i])] for i in range(5)] for N in rang
e(len(volumes))]

print('Примеры вариационных рядов:\n', *VarRangeZipf[0][0])
print(*VarRangeZipf[1][1], end = '\n\n\n')

print('Выборочные квантили:')
for samples in SampleZipf:
    print(*np.quantile(samples, [0.1,0.5,0.7], axis=1).T)

print()

print('Теоретические квантили:\n', sts.zipf.ppf([0.1,0.5,0.7], 2))
```

```
Примеры вариационных рядов:
 [ 1  1  1  1 22]
[ 1  1  1  1  1  1  1  2  3 30]


Выборочные квантили:
[1. 1. 1.] [1.  1.  2.6] [1.  2.  2.8] [1. 2. 2.] [1. 1. 1.]
[1. 3. 4.] [1.  1.  1.3] [1.  2.  2.3] [1. 2. 2.] [1.  1.  1.3]
[1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.]
[1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.]
[1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.] [1. 1. 2.]

Теоретические квантили:
 [1. 1. 2.]
```

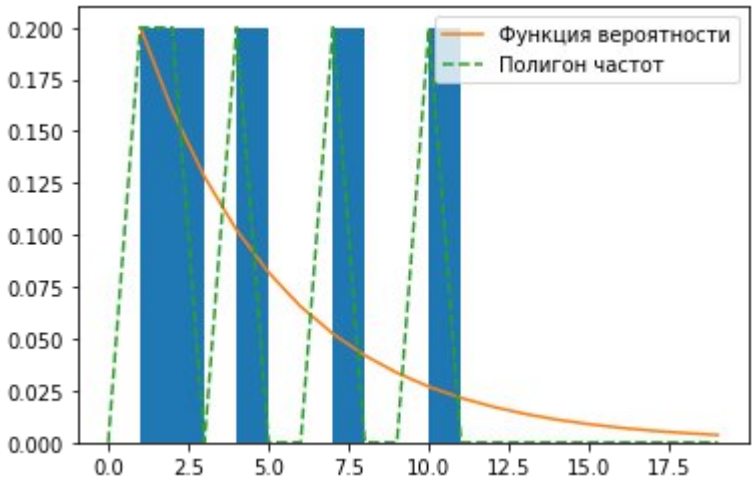
Ура, здесь тоже без сюрпризов. Увеличение объёма выборки дало точное приближение к заданным квантилям

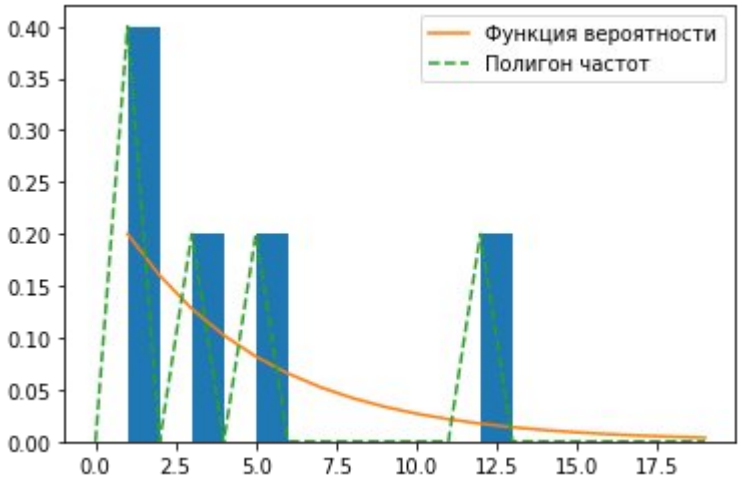
2.4. Построение гистограммы и полигон частот

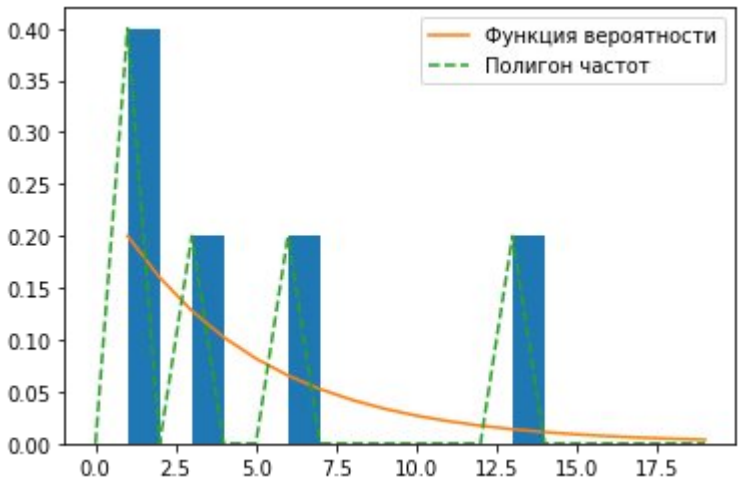
Геометрическое распределение

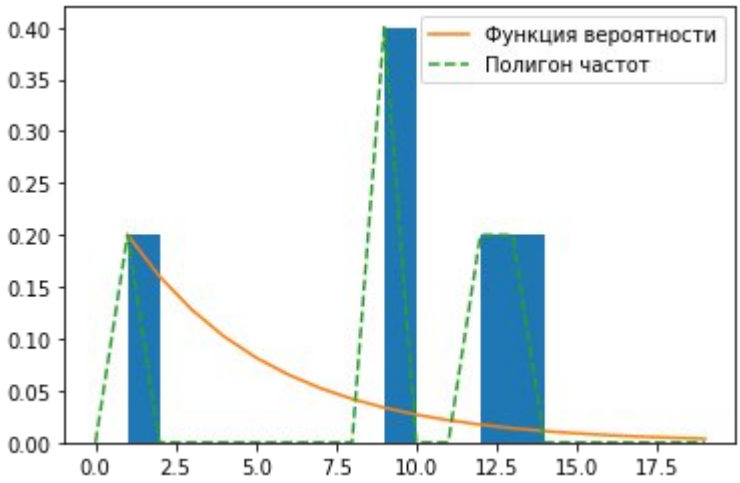
In [34]:

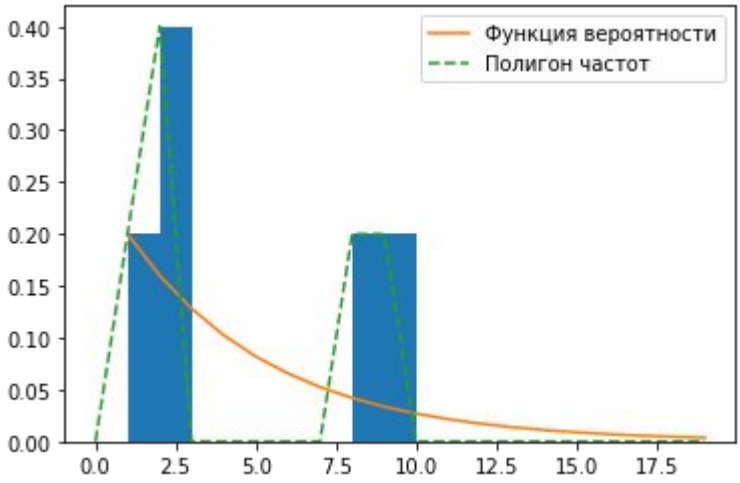
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        plt.hist(SampleGeom[i][j], bins = range(1, 20), density=True)

        GeomFreqs = []
        for k in range(20): # в иделе должно быть range(len(SampleGeom[i]
[j]))
                            # но это слишком большое число
                            # и на графиках становится ничего не видно
                            # поэтому на графике показаны частоты лишь первых
20 значений
            GeomFreqs.append(SampleGeom[i][j].count(k)/len(SampleGeom[i][j]))

        plt.plot(range(1, 20), sts.geom.pmf(range(1, 20), 0.2), label='Функци
я вероятности')
        plt.plot(GeomFreqs, '--', label='Полигон частот')

        plt.legend(loc='upper right')
        plt.show()
```

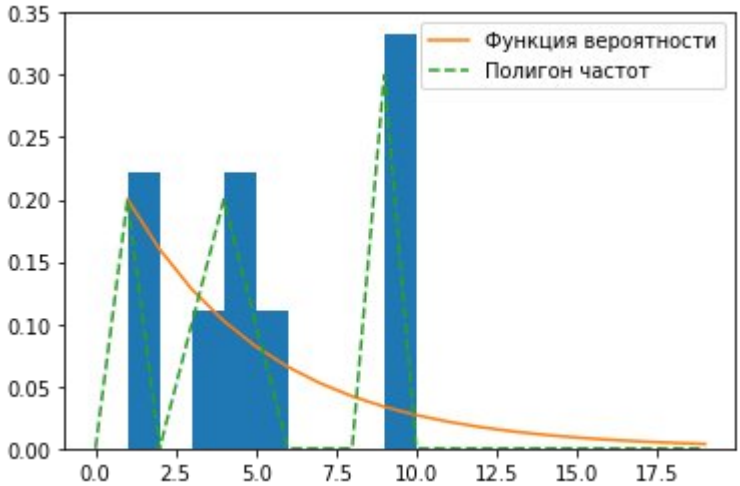

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

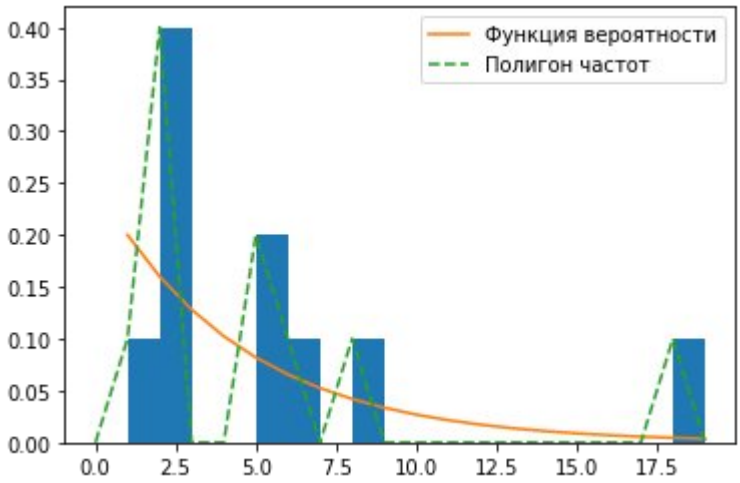

Функция вероятности
Полигон частот
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

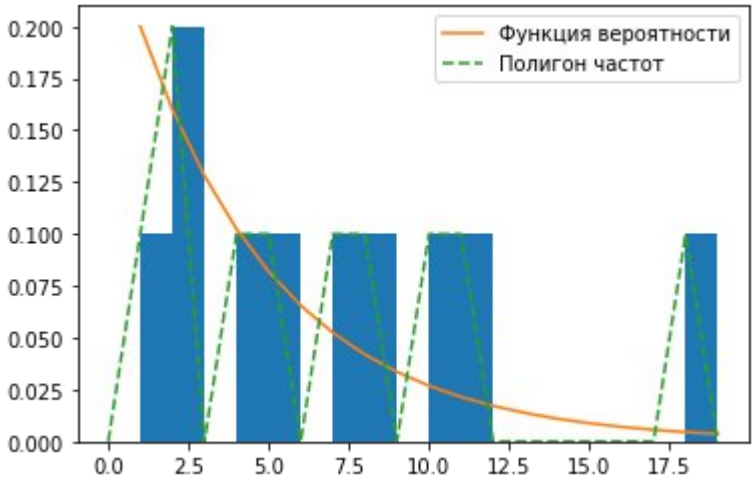

Функция вероятности
Полигон частот
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

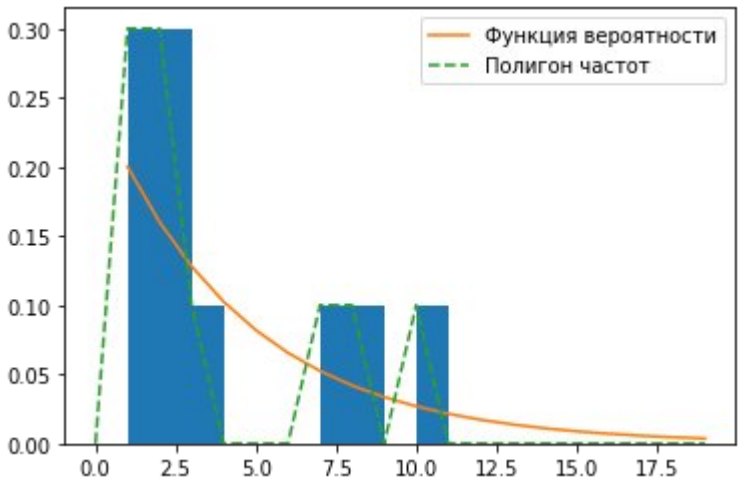

Функция вероятности
Полигон частот
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

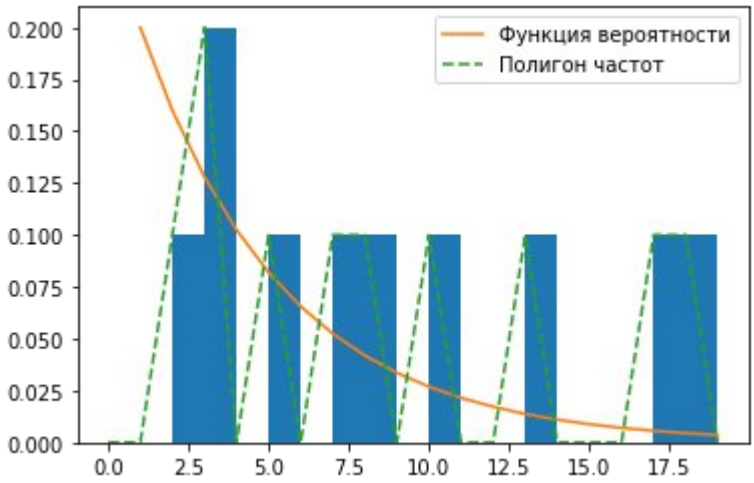

Функция вероятности
Полигон частот
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

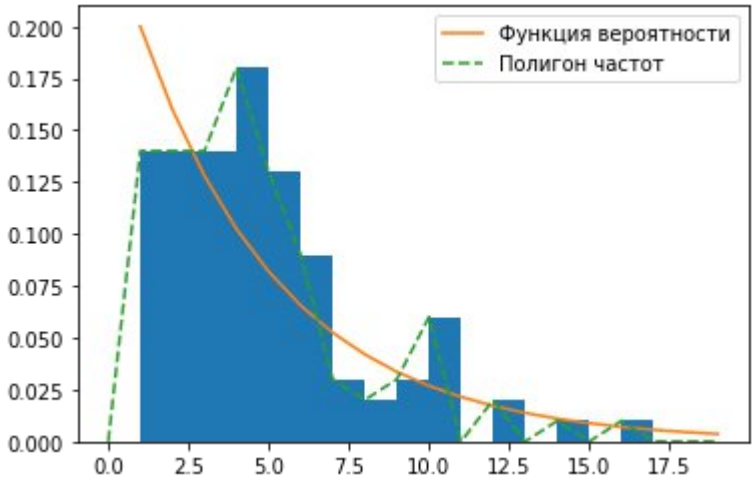

Функция вероятности
Полигон частот
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

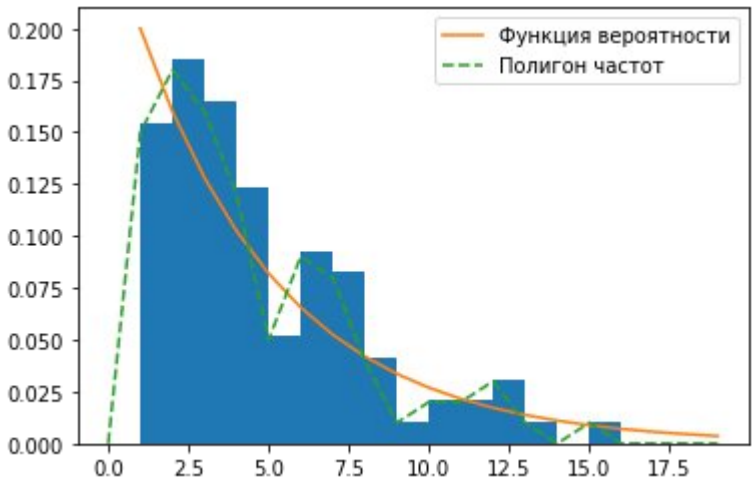

Функция вероятности
Полигон частот
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

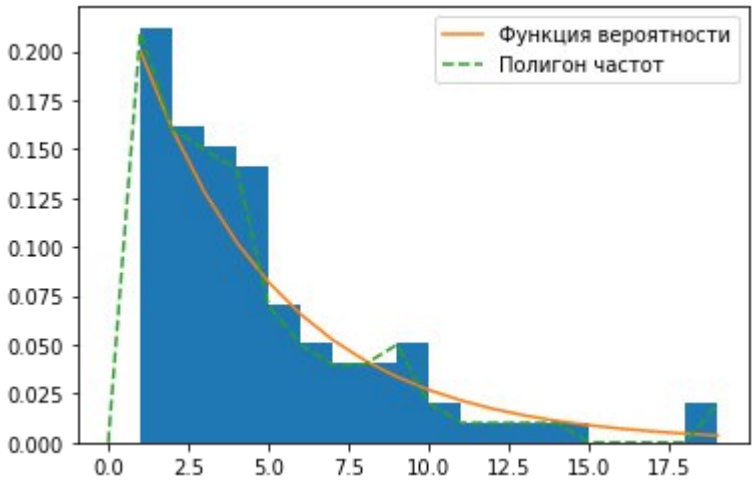

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

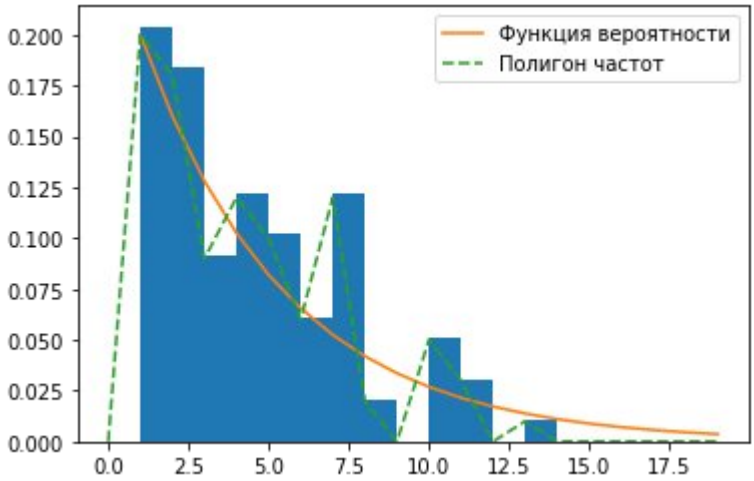

Функция вероятности
Полигон частот
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

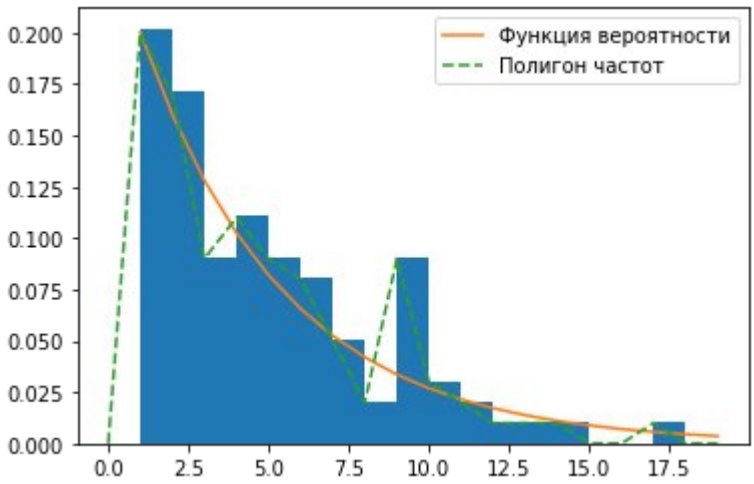

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

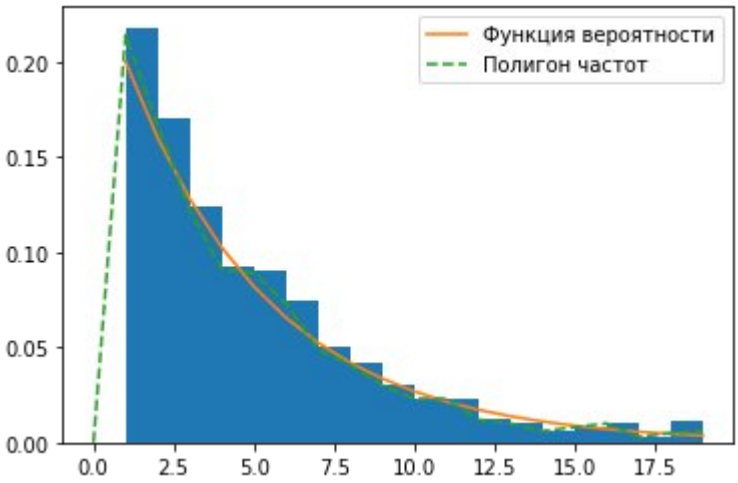

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

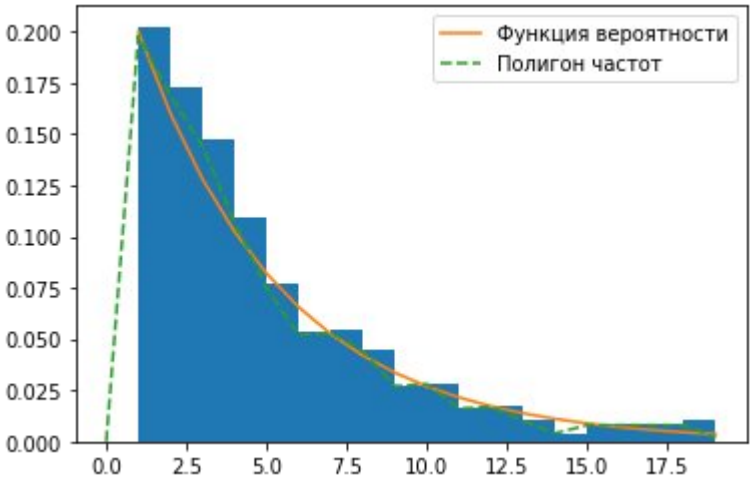

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

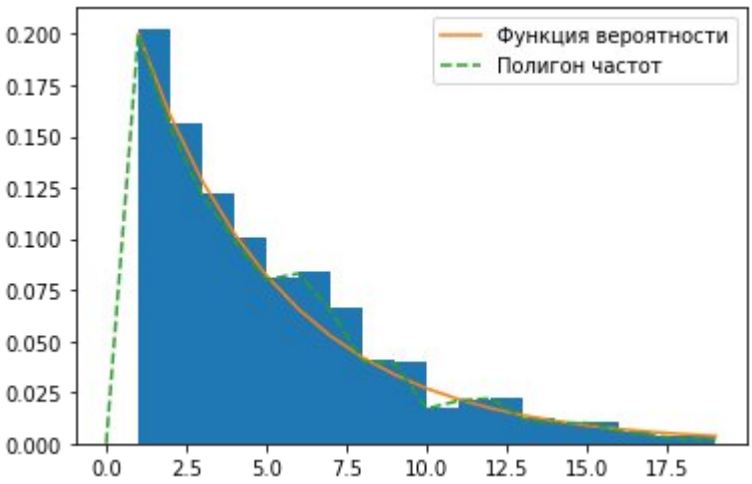

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

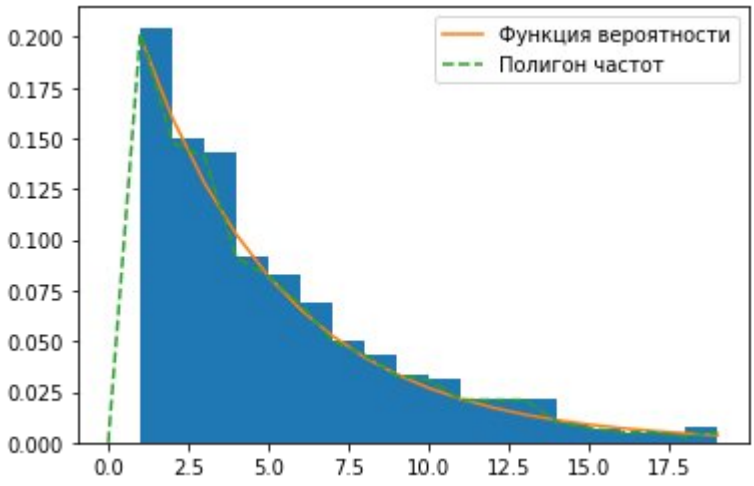

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

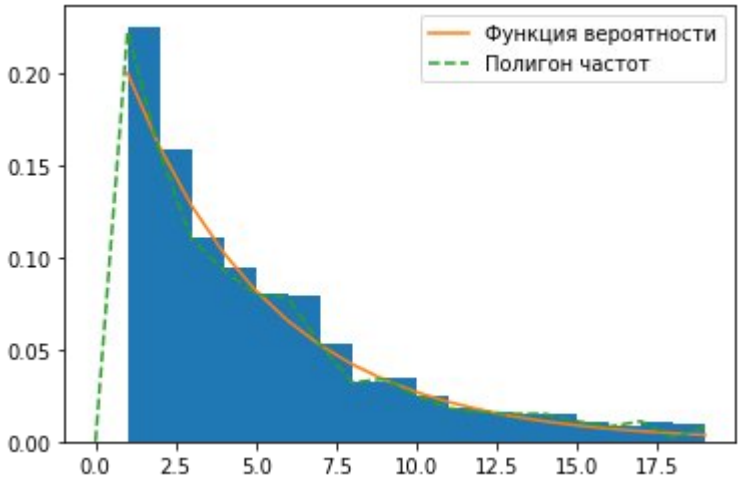

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

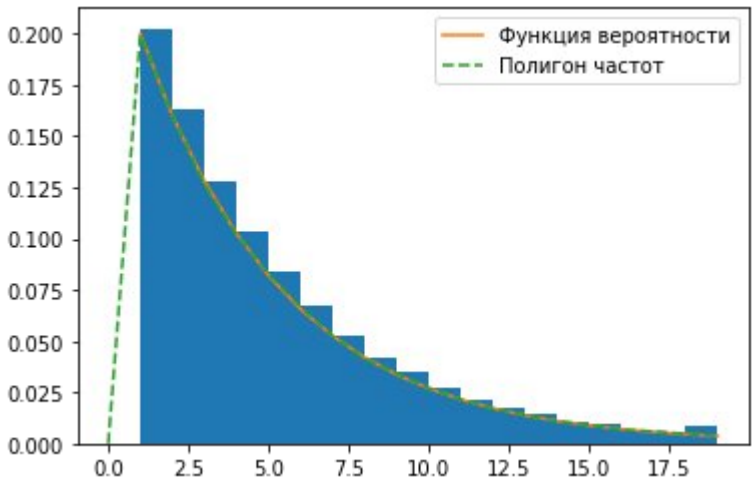

Функция вероятности
Полигон частот
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

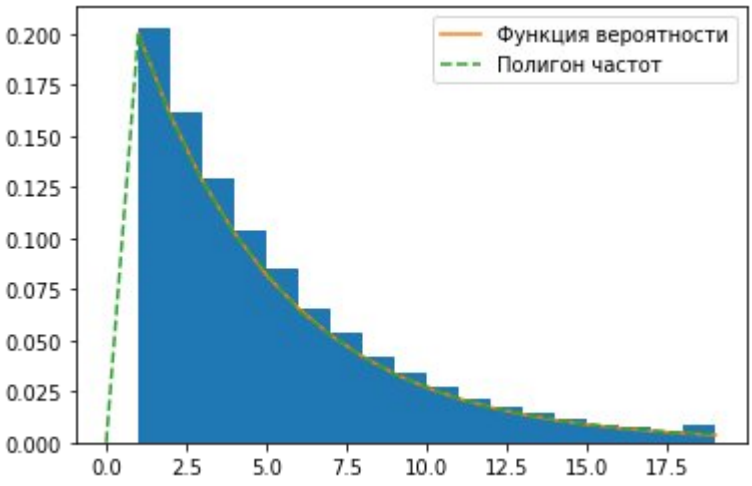

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

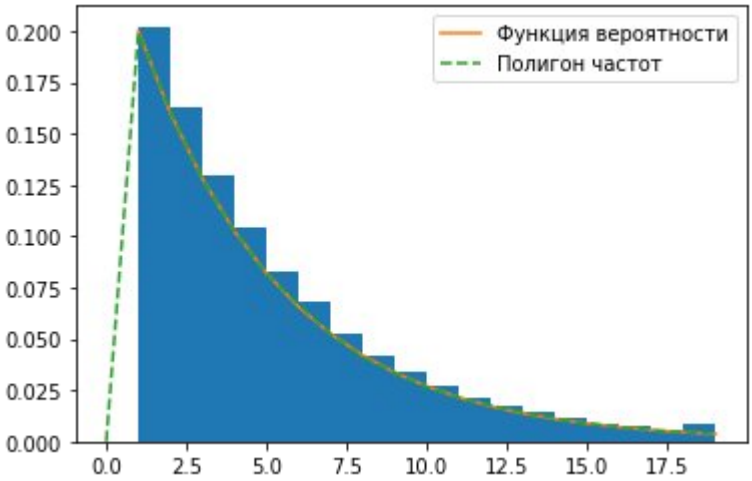

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

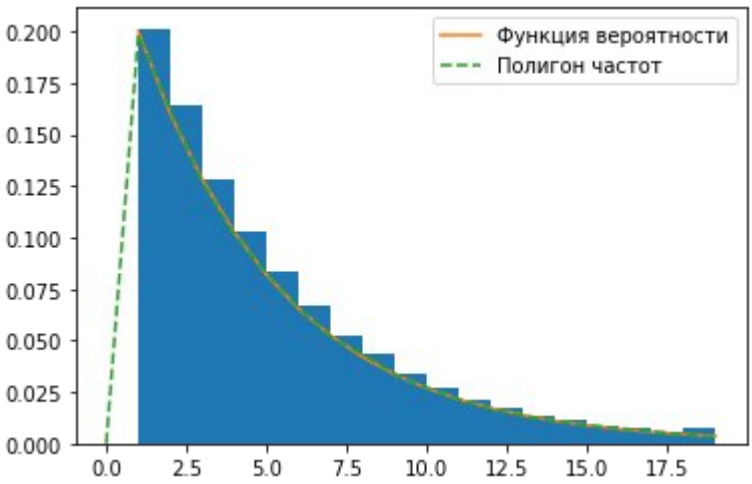

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

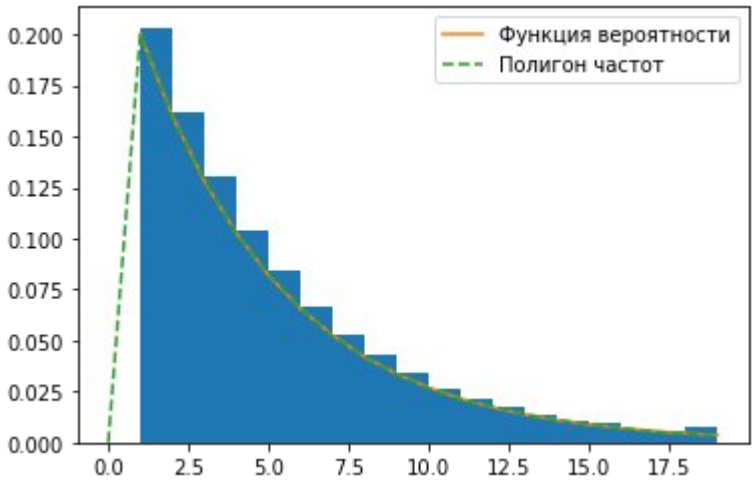

Функция вероятности
Полигон частот
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

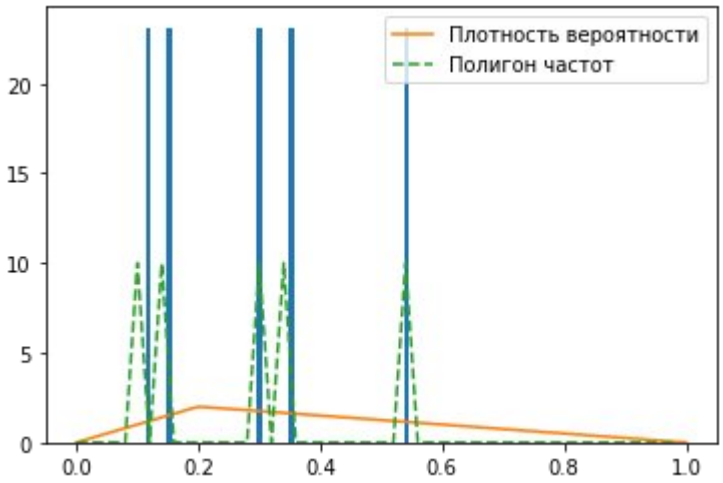

Функция вероятности
Полигон частот
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

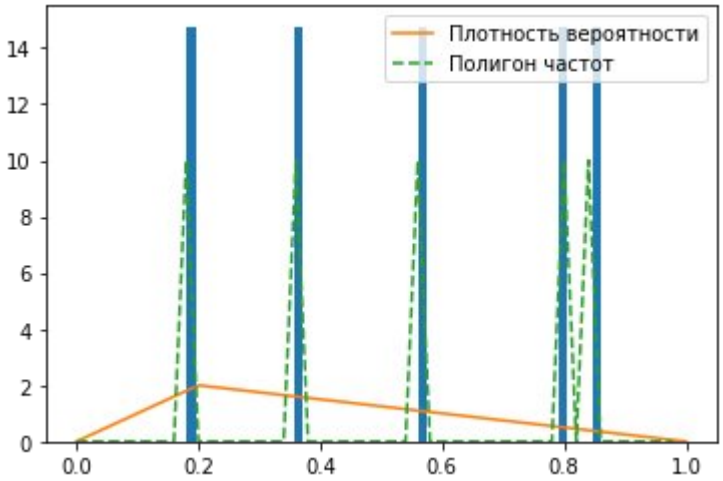


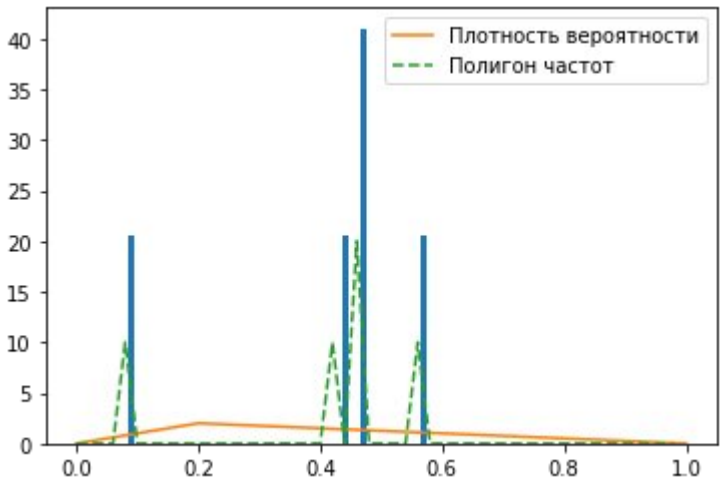
В результате наблюдаем, что полигон частот стремится к функции вероятности случайной величины, это чудесно

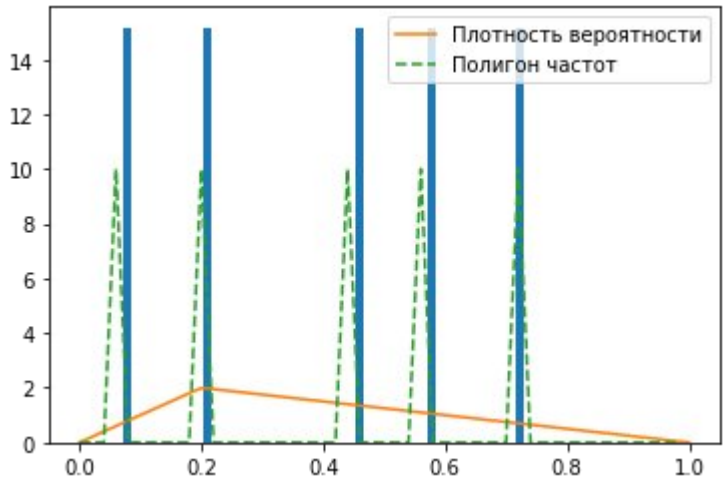
Треугольное распределение

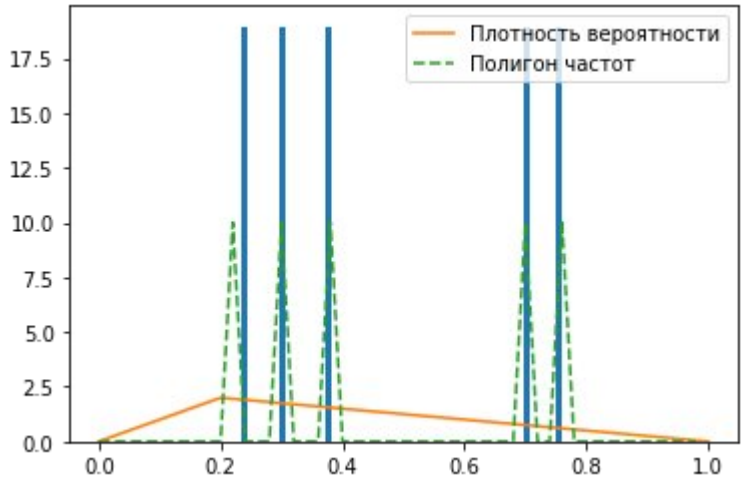
In [28]:
```
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        plt.hist(SampleTrig[i][j], bins = 50, density=True)

        TrigX = []
        TrigFreqs = []
        for a in range(50):
            count = 0
            for k in range(len(SampleTrig[i][j])):
                if (SampleTrig[i][j][k] >= a/50 and SampleTrig[i][j][k] < (a+
1)/50):
                    count += 1
            TrigFreqs.append(count*50/len(SampleTrig[i][j]))
            TrigX.append(a/50)

        x = np.linspace(sts.triang.ppf(0, 0.2), sts.triang.ppf(1, 0.2), 100)
        plt.plot(x, sts.triang.pdf(x, 0.2), label='Плотность вероятности')
        plt.plot(TrigX, TrigFreqs, '--', label='Полигон частот')

        plt.legend(loc='upper right')
        plt.show()
```

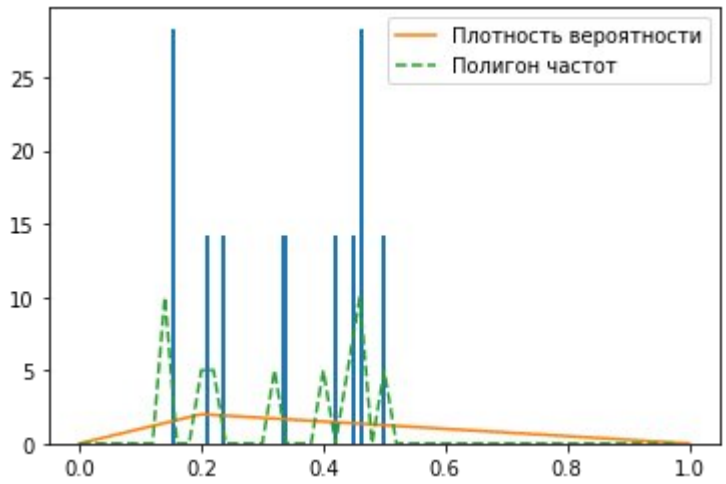

Плотность вероятности
Полигон частот
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

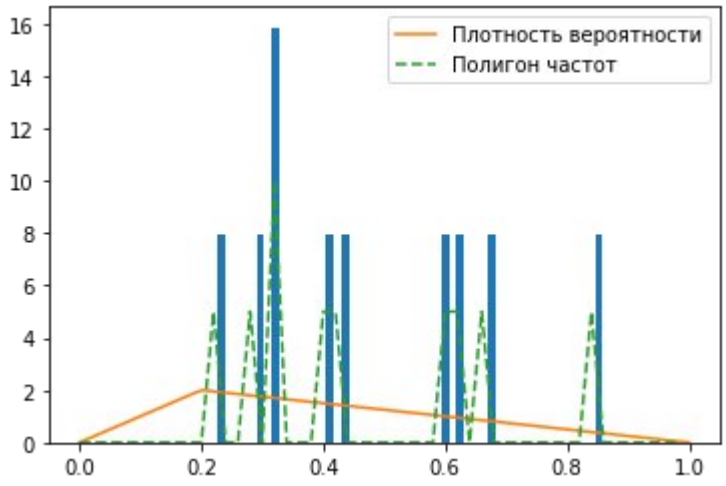

Плотность вероятности
Полигон частот
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

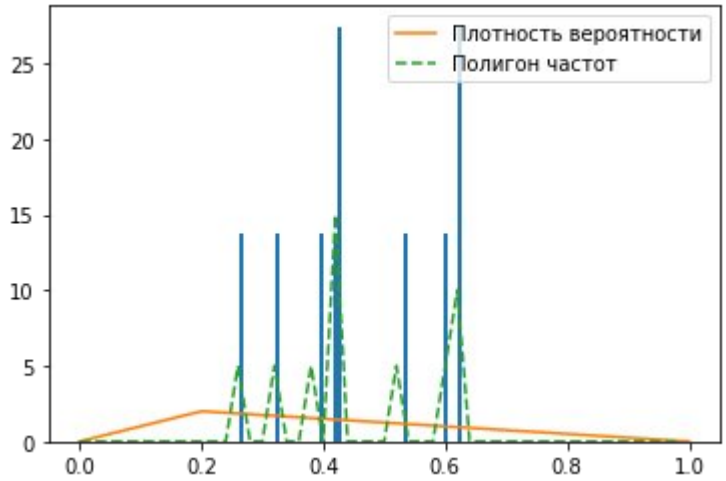

Плотность вероятности
Полигон частот
40
35
30
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

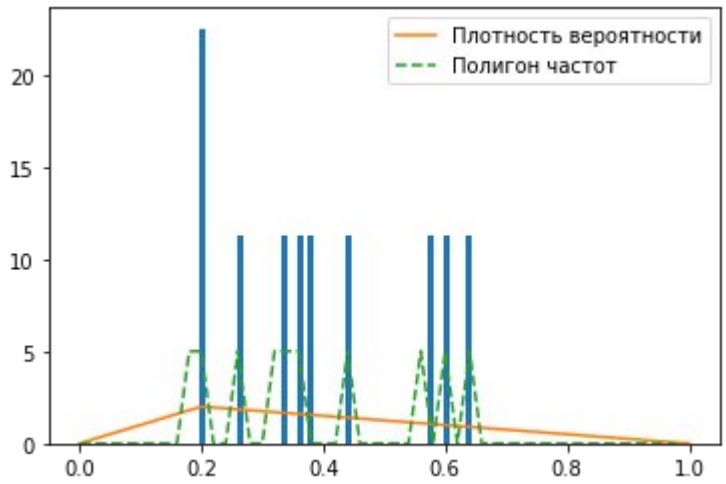

Плотность вероятности
Полигон частот
0
2
4
6
8
10
12
14
0.0
0.2
0.4
0.6
0.8
1.0

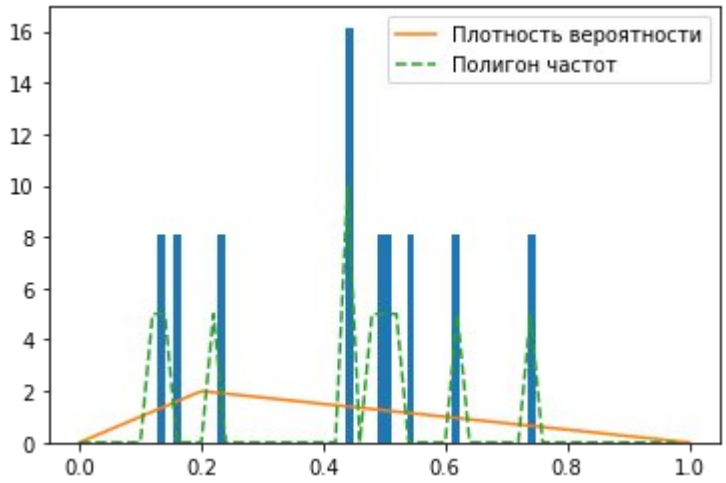

Плотность вероятности
Полигон частот
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
0.2
0.4
0.6
0.8
1.0

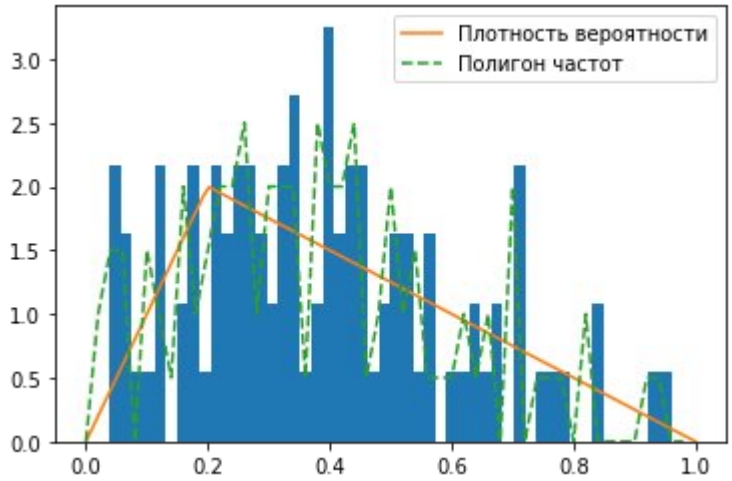

Плотность вероятности
Полигон частот
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0

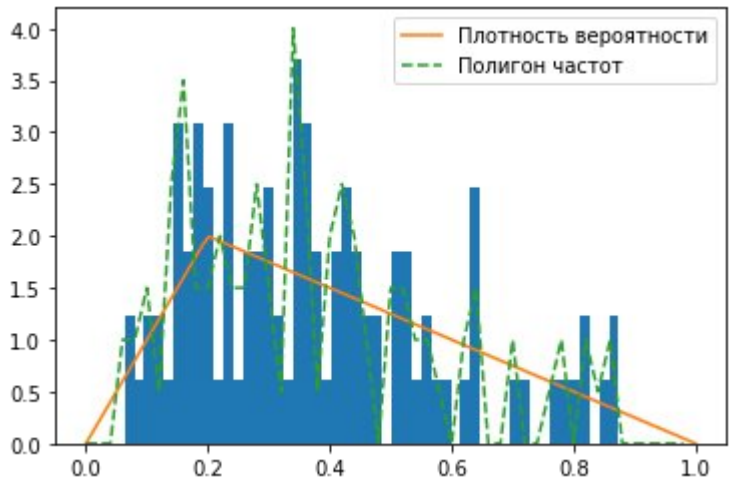

Плотность вероятности
Полигон частот
16
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

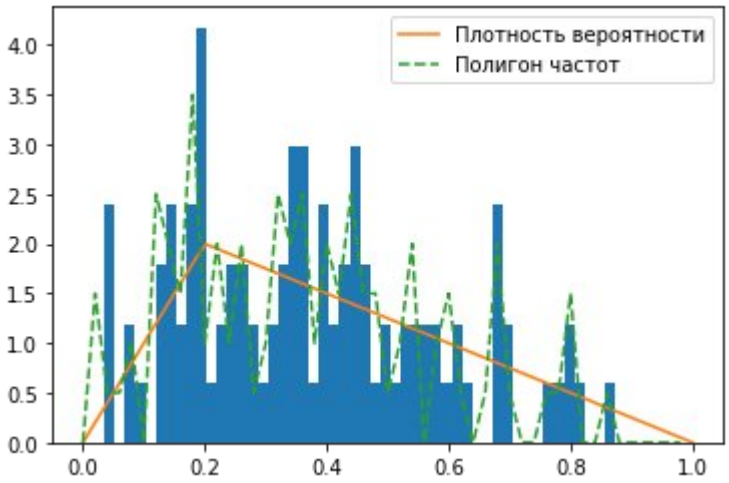

Плотность вероятности
Полигон частот
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

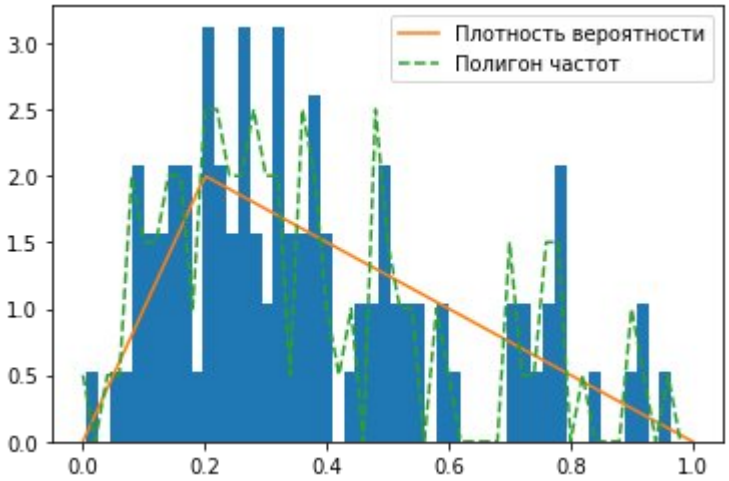

Плотность вероятности
Полигон частот
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

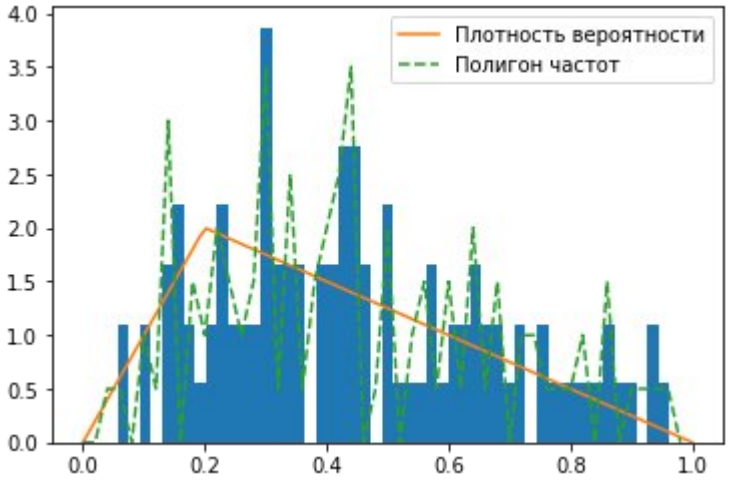

Плотность вероятности
Полигон частот
16
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

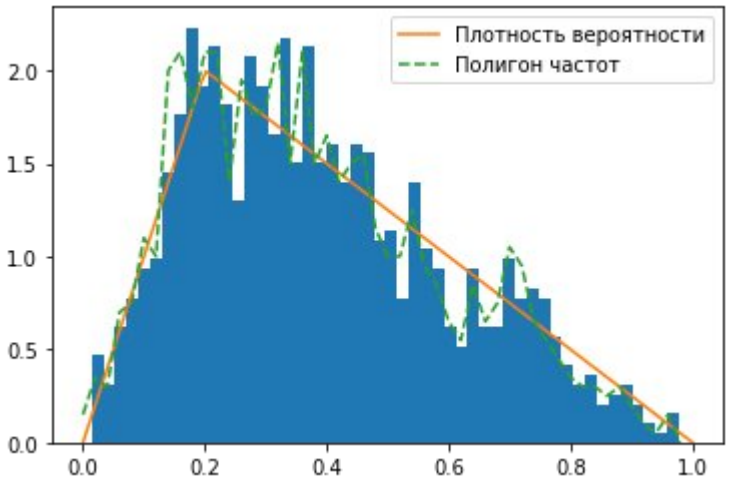

Плотность вероятности
Полигон частот
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Плотность вероятности
Полигон частот
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

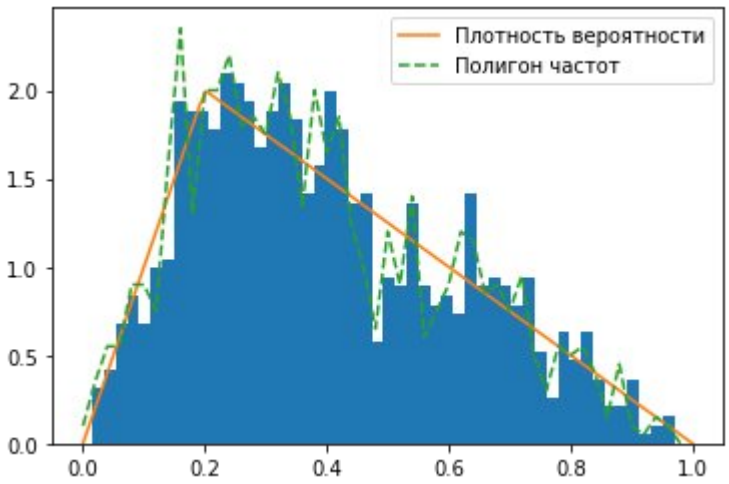

Плотность вероятности
Полигон частот
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0

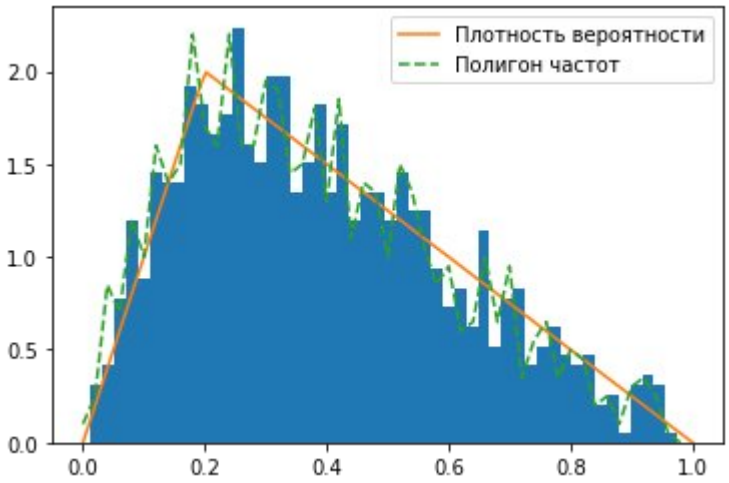

Плотность вероятности
Полигон частот
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0

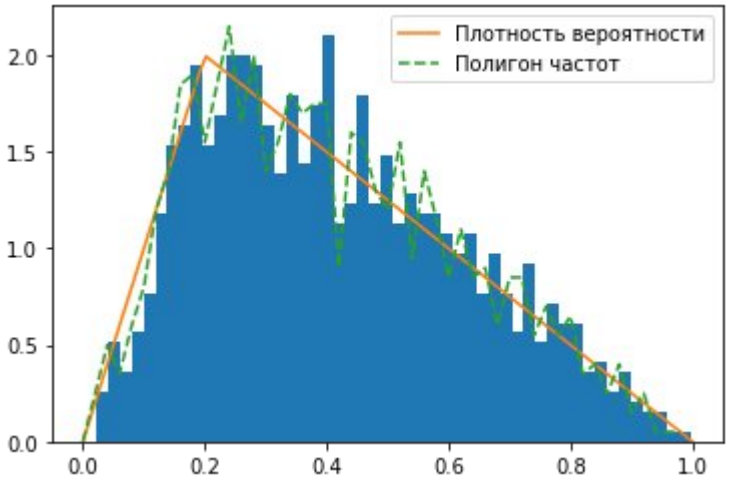

Плотность вероятности
Полигон частот
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0

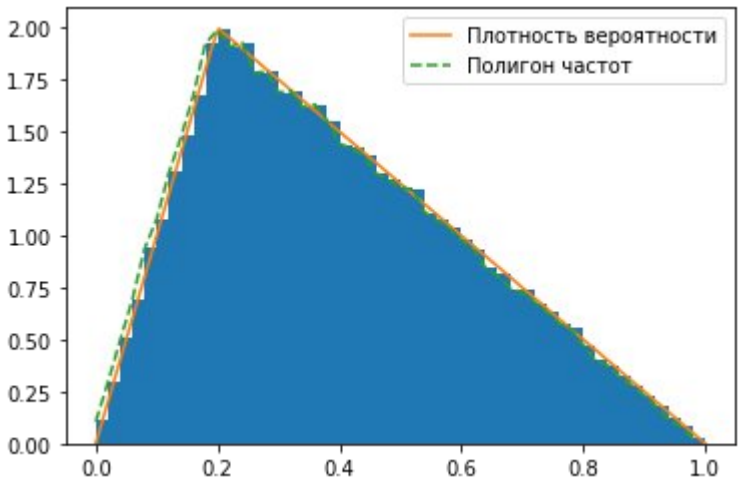

Плотность вероятности
Полигон частот
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

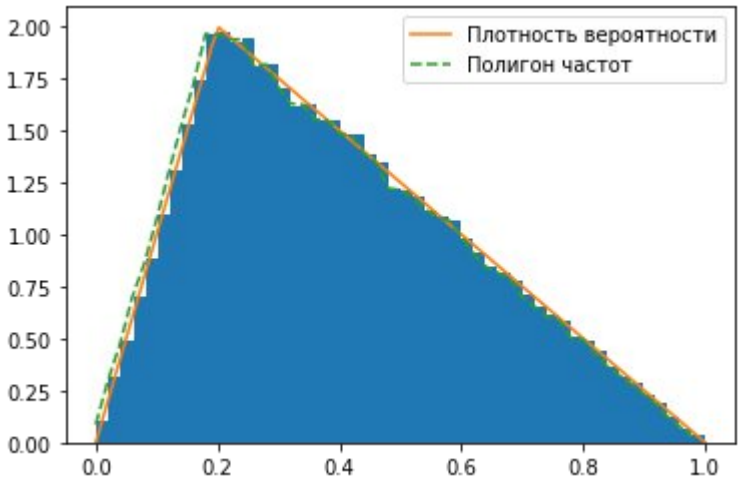

Плотность вероятности
Полигон частот
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

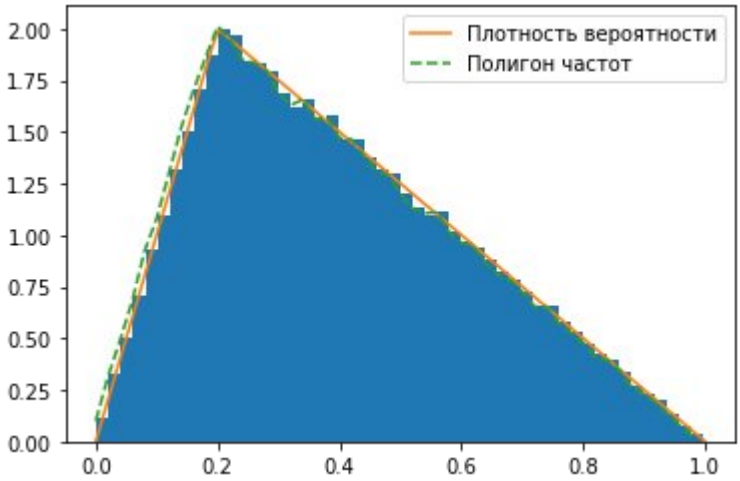

Плотность вероятности
Полигон частот
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

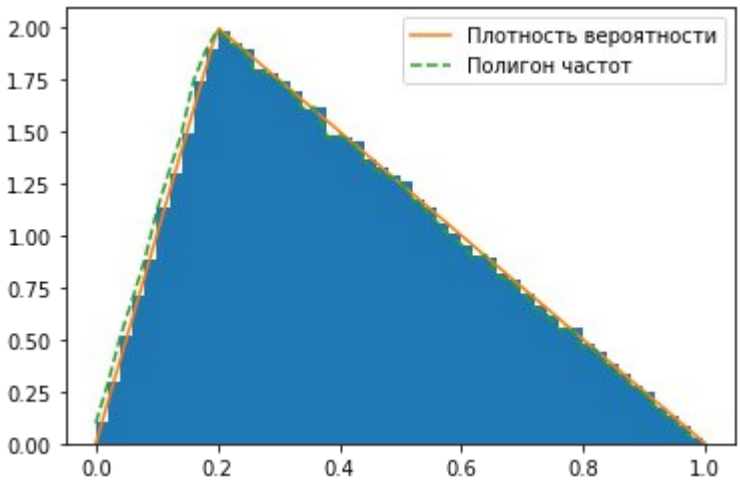

Плотность вероятности
Полигон частот
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

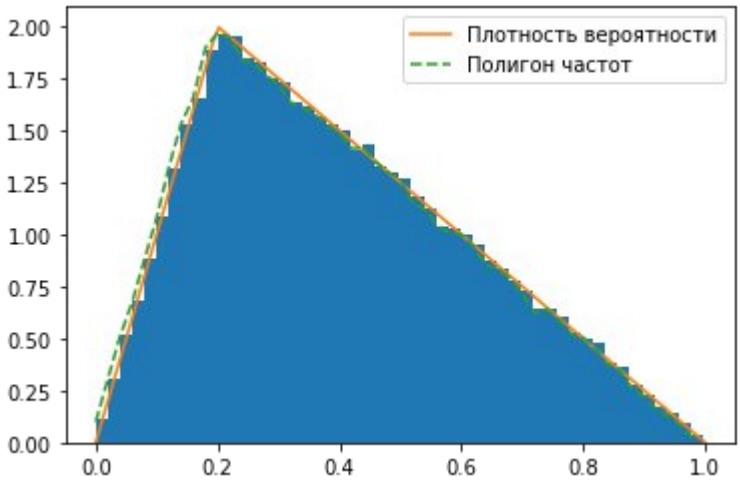

Плотность вероятности
Полигон частот
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Плотность вероятности
Полигон частот
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Плотность вероятности
Полигон частот
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Плотность вероятности
Полигон частот
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Плотность вероятности
Полигон частот
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

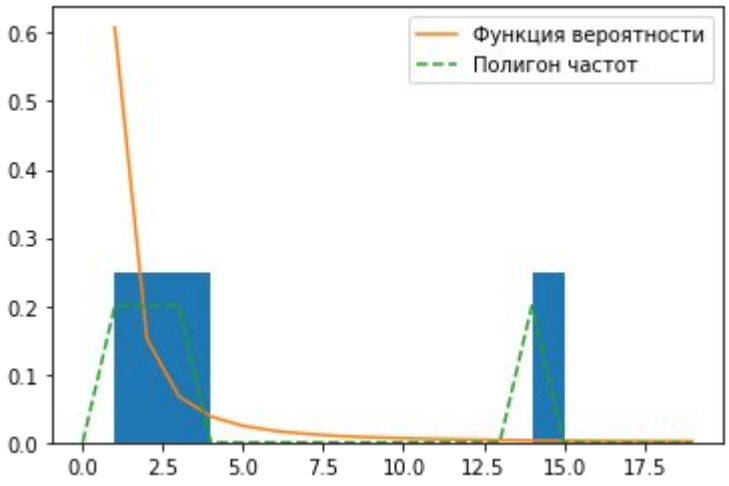


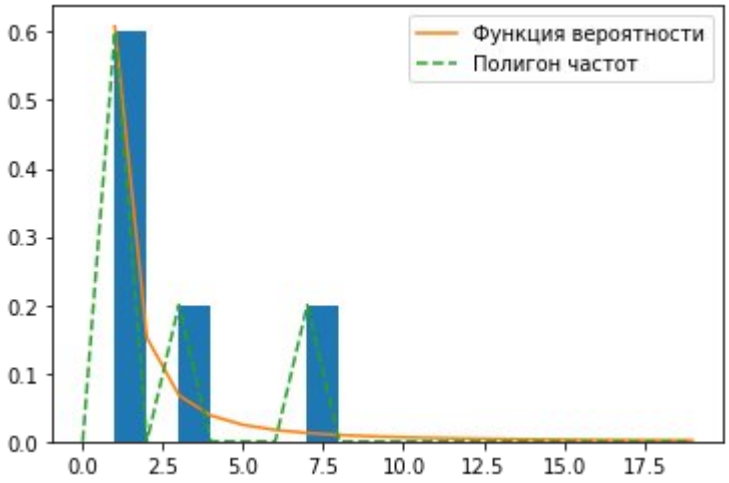
Как и в случае геометрического распределения, здесь поликон частот стремится к функции плотности вероятности случайной величины

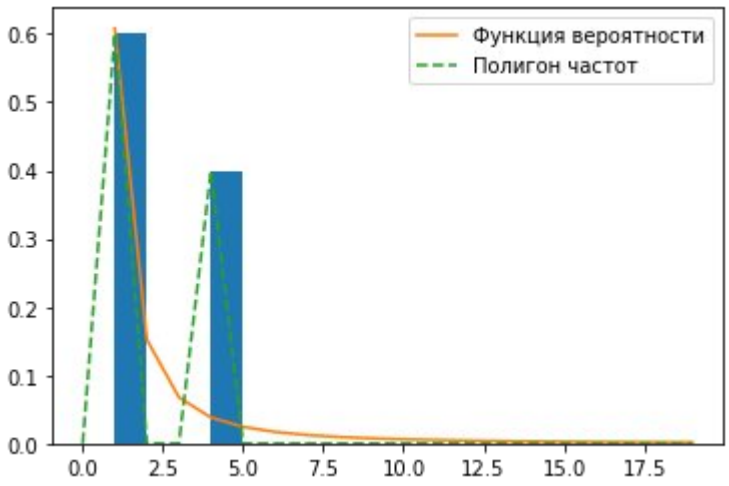
Распределение Ципфа

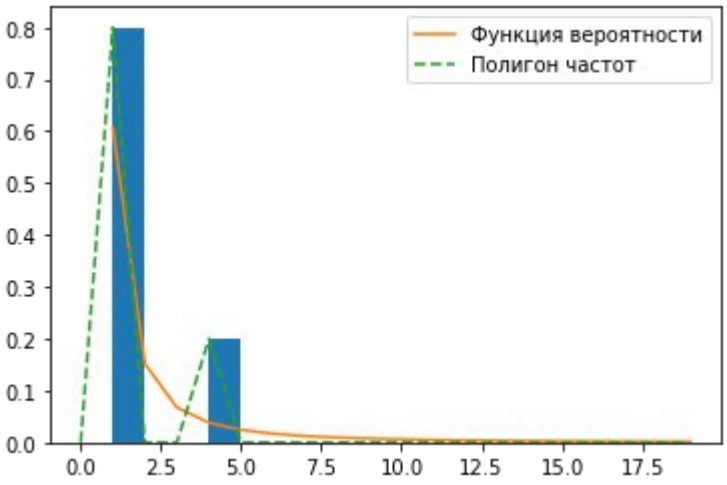
In [36]:

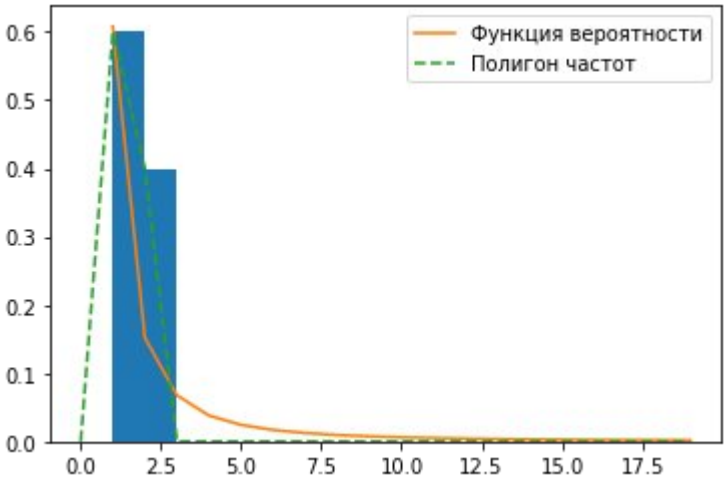
```
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        plt.hist(SampleZipf[i][j], bins = range(1, 20), density=True)

        ZipfFreqs = []
        for k in range(20): # в иделе должно быть range(len(SampleGeom[i]
[j]))
                            # но это слишком большое число
                            # и на графиках становится ничего не видно
                            # поэтому на графике показаны частоты лишь первых
20 значений
            ZipfFreqs.append(SampleZipf[i][j].count(k)/len(SampleZipf[i][j]))

        plt.plot(range(1, 20), sts.zipf.pmf(range(1, 20), 2), label='Функция
вероятности')
        plt.plot(ZipfFreqs, '--', label='Полигон частот')

        plt.legend(loc='upper right')
        plt.show()
```

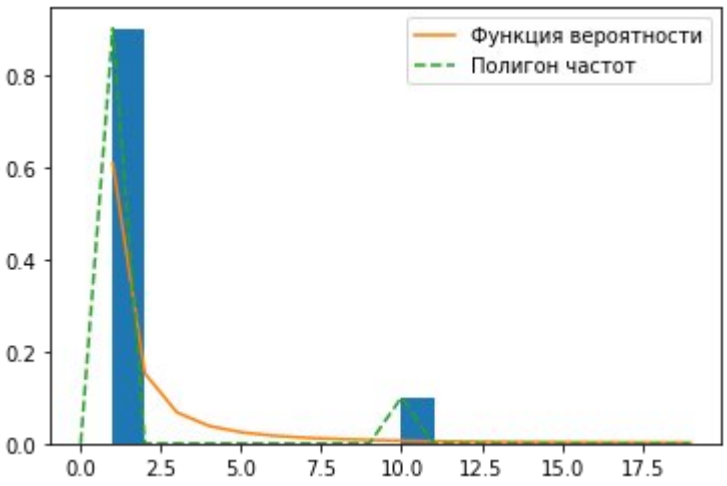

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

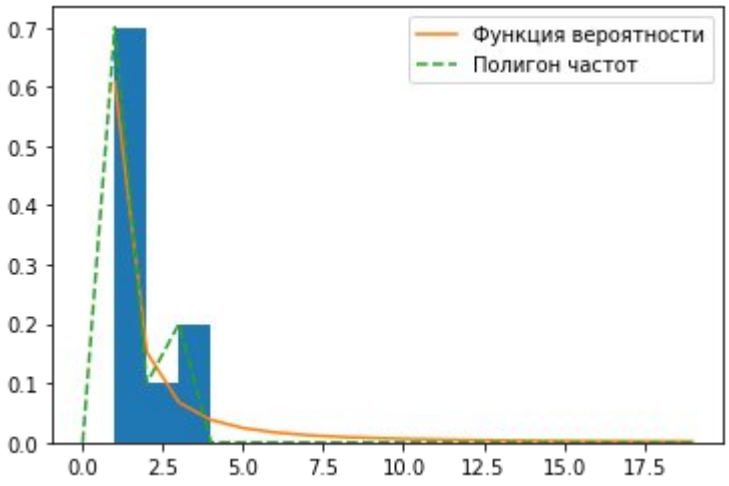

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

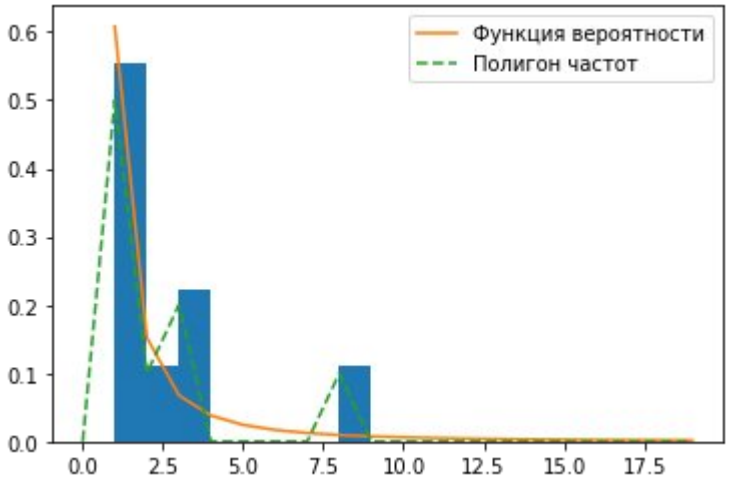

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

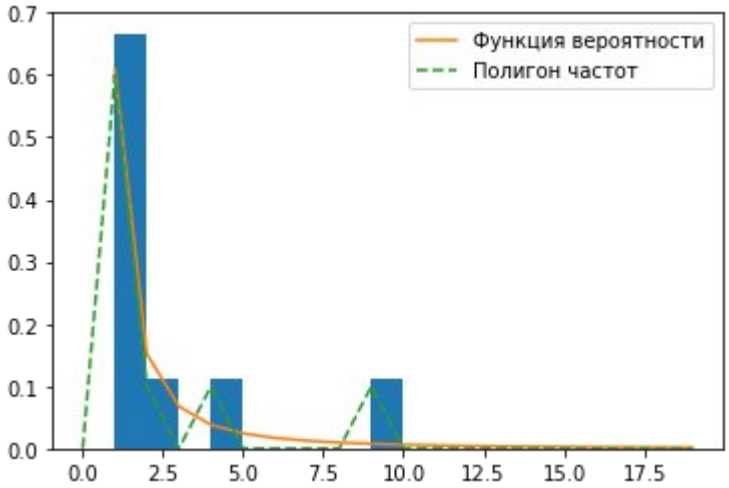

Функция вероятности
Полигон частот
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

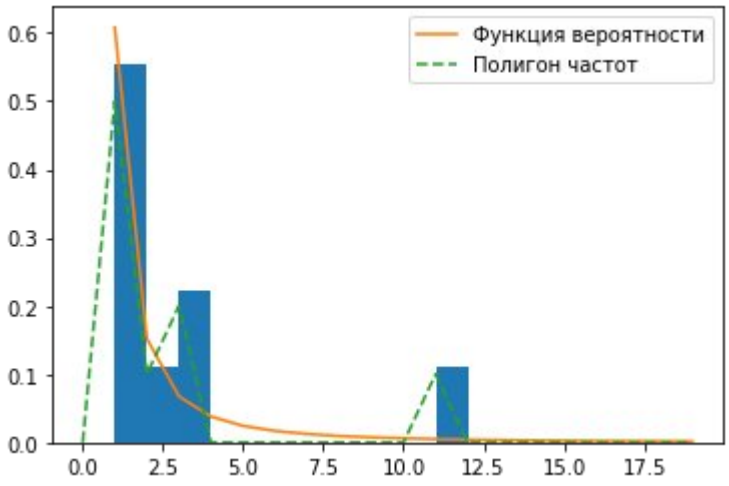

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

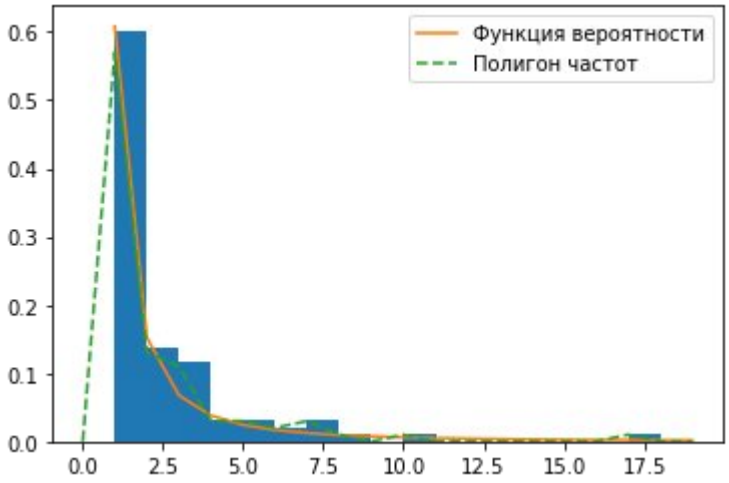

Функция вероятности
Полигон частот
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

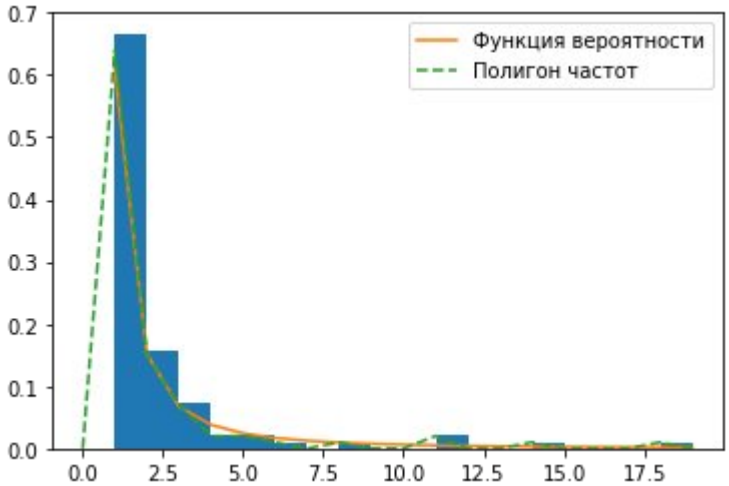

Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

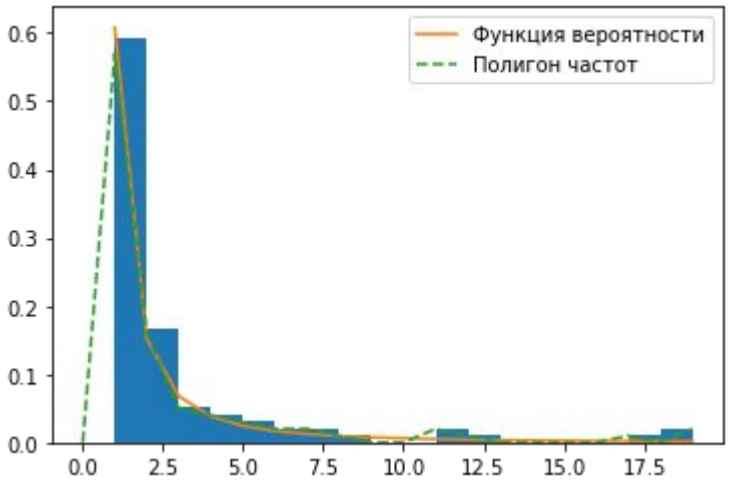

Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

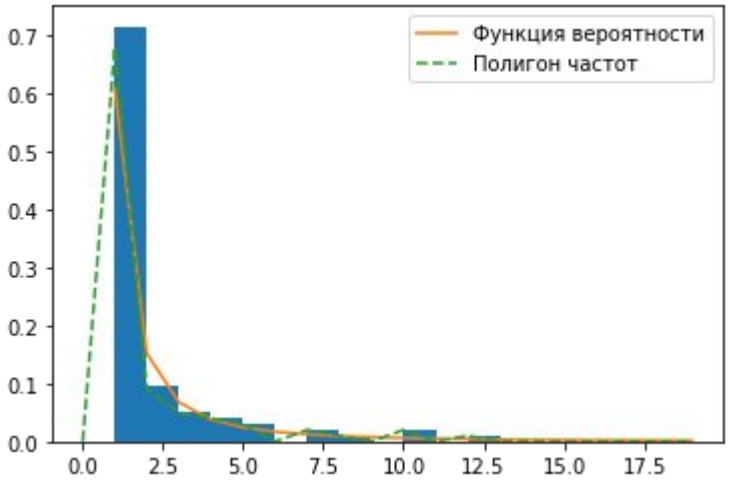

Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

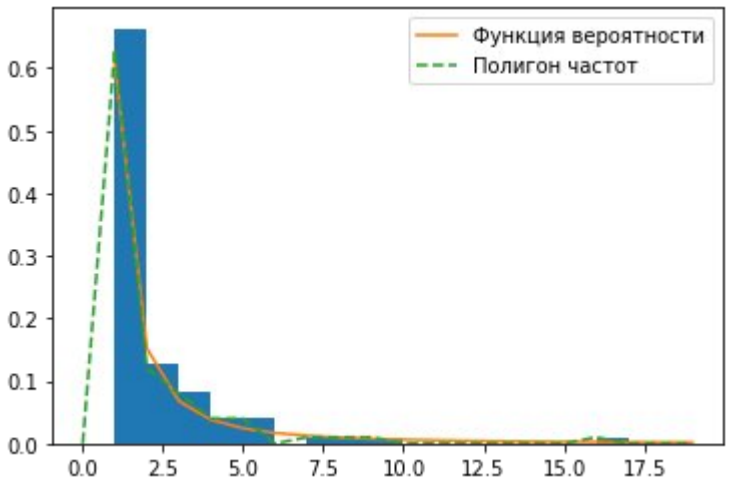

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

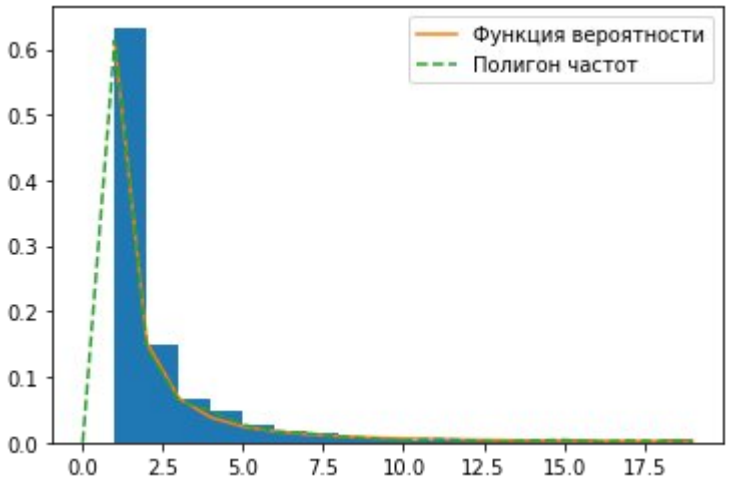

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

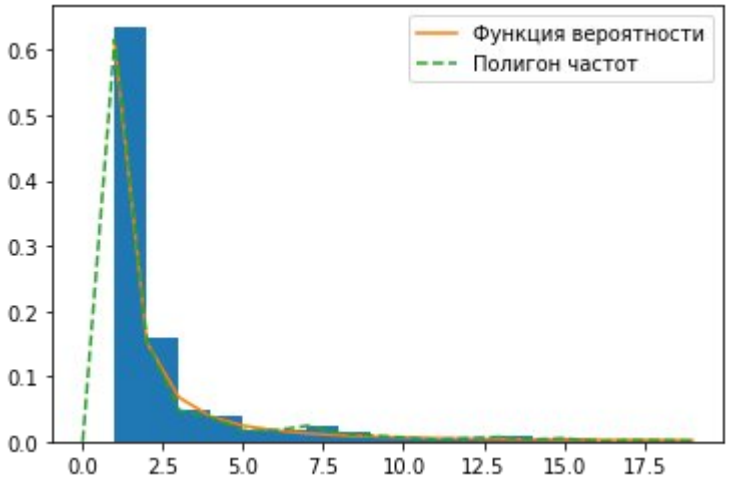

Функция вероятности
Полигон частот
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

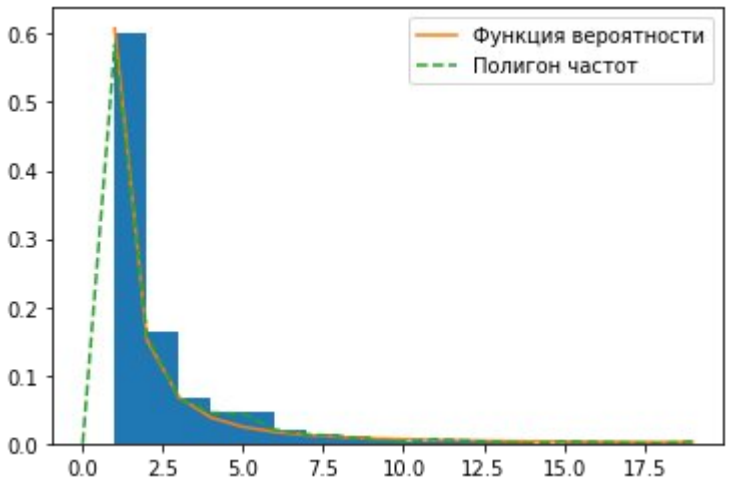

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

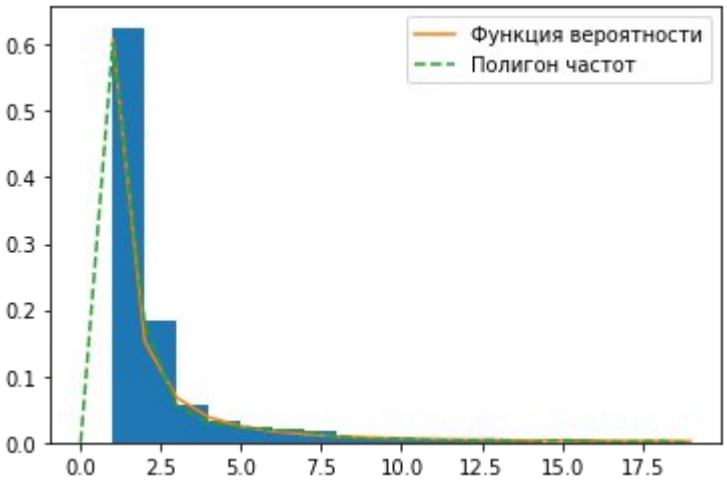

Функция вероятности
Полигон частот
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

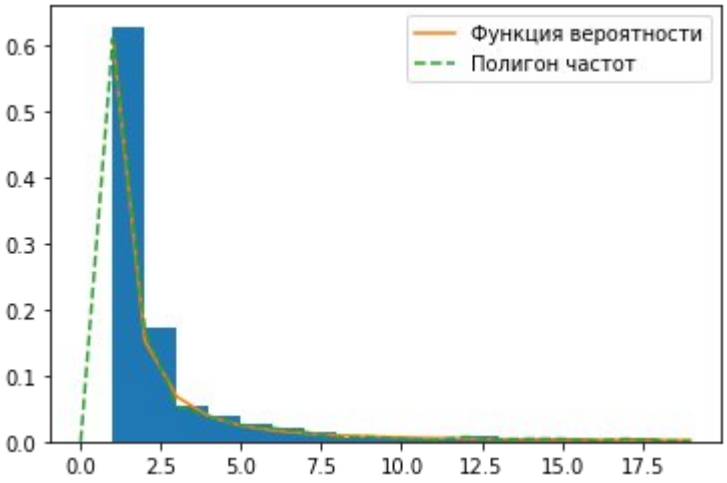

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

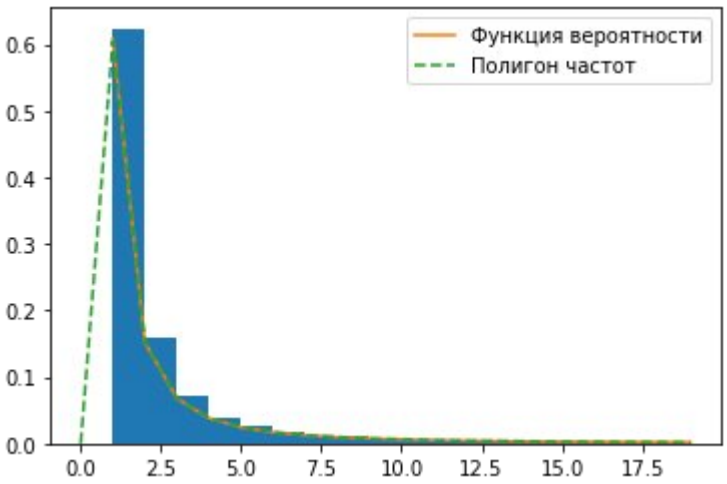

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

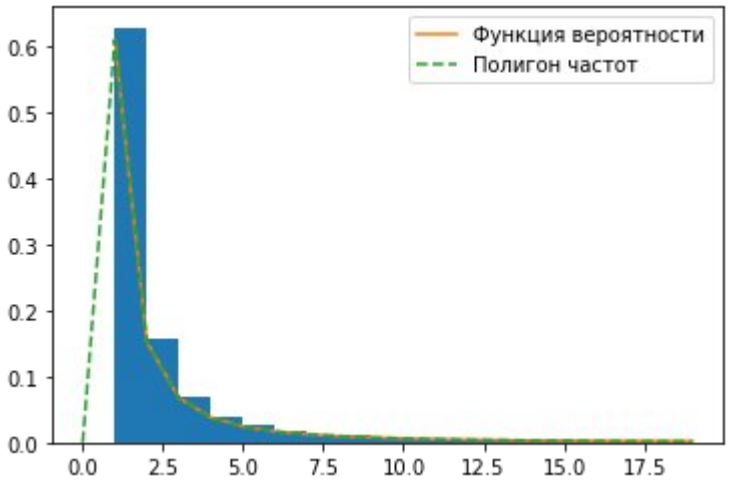

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

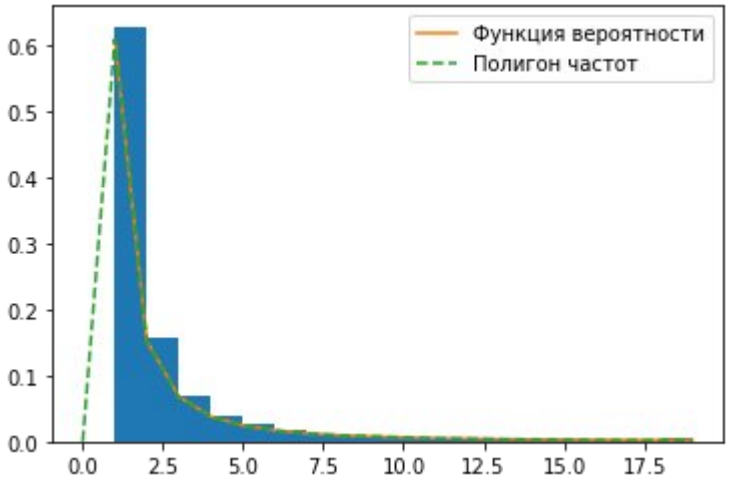

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

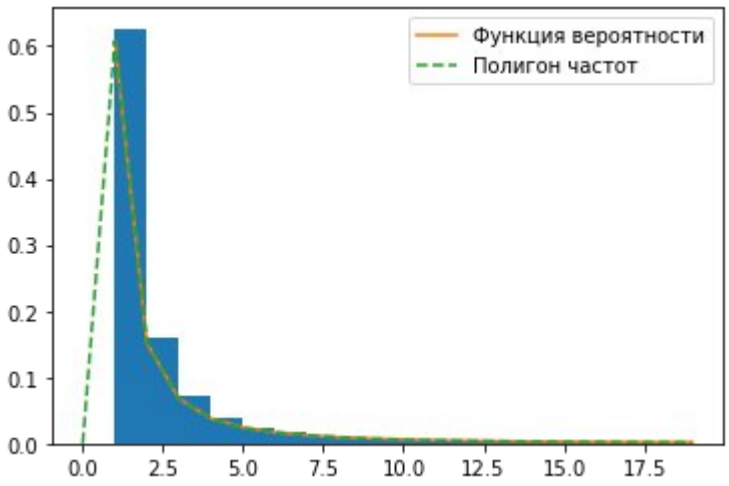

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

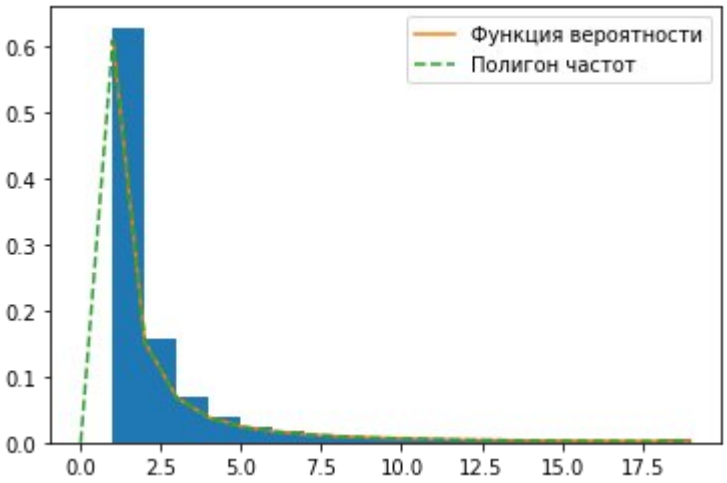

Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Функция вероятности
Полигон частот
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Аналогично с предыдущими распределениями, наблюдаем чудесную картину – полигон частот стремится к функции вероятности случайной величины.

Домашнее задание 3

3.1. Нахождение выборочного среднего и выбо- рочной дисперсии

Геометрическое распределение

Напишем свою функцию для подсчёта выборочного среднего и с помощью неё выведем выборочные средние наших выборок

In [10]:
```
def MEAN(sample):
    return sum(sample)/len(sample)
```

In [14]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print((MEAN(SampleGeom[i][j])),
              end = '\n' if j == len(SampleGeom[i])-1 else ' ')
```

```
8.2 3.8 2.2 4.6 2.8
4.3 4.1 4.8 6.7 4.8
4.91 4.6 4.68 4.73 5.21
5.249 5.04 5.116 4.984 4.923
5.0051 5.00355 4.98833 5.00381 5.01003
```

Теперь получим теоретическое среднее

In [7]:
```
sts.geom.mean(0.2)
```

Out[7]: 5.0

Исправлено: показаны свойства получаемых велечин с помозью выборочной дисперсии

Как мы видим, увеличение объёма выборки привело к очень хорошому приближению к теоретическому среднему – отклонение составляет менее одного процента.

Выборочное среднее является несмещённой оценкой, так как ошибка оценки "в среднем" отсутствует – выборочное среднее стремится к математическому ожиданию данной случайной величины.

Выборочное среднее является состоятельной оценкой, так как она сходится по вероятности к оцениваемому параметру.

Посчитаем выборочную дисперсию для каждого объёма выборки – увидим, что она стремится к нулю с ростом объёма выборки – и затем покажем свойства с помощью следующей дисперсии случайной величины: $\sqrt{n}(\hat{\theta} - \theta)$

In [12]:
```
for i in range(len(SampleGeom)):
    means = []
    for j in range(len(SampleGeom[i])):
        means.append(MEAN(SampleGeom[i][j]))
    print(VAR(means))
```

```
1.6895999999999993
2.5623999999999993
0.3153039999999999
0.03522199999999999
0.00014585070400000333
```

In [13]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print(sqrt(len(SampleGeom[i][j]))*(5-MEAN(SampleGeom[i][j])),
              end = '\n' if j == len(SampleGeom[i])-1 else ' ')
```

```
-7.155417527999326 2.683281572999748 6.260990336999411 0.8944271909999167 4.9
19349550499538
2.2135943621178664 2.846049894151543 0.6324555320336764 -5.375872022286246 0.
6324555320336764
0.8999999999999986 4.000000000000036 3.200000000000003 2.6999999999999957 -
2.0999999999999996
-7.874071373819254 -1.2649110640673529 -3.6682420857953093 0.5059644256269411
2.4349537983296505
-1.6127616066857662 -1.1226085693596863 3.6903780294163804 -1.204827788524041
-3.17176449314902
```

Далее находим выборочную дисперсию и сравним с теоретической дисперсией

In [230]:
```
def VAR(sample):
    mean = MEAN(sample)
    s = 0
    for element in sample:
        s += (element - mean)**2
    return s/len(sample)
```

In [18]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print(VAR(SampleGeom[i][j]), end = '\n' if j == len(SampleGeom[i])-1
else ' ')

sts.geom.var(0.2)
```

```
75.75999999999999 13.36 1.3599999999999999 14.64 1.3599999999999999
11.209999999999997 4.09 21.96 55.61 19.359999999999996
22.261900000000022 19.320000000000007 17.397600000000008 13.5371 22.465899999
999998
26.092999000000162 21.52439999999994 20.398544000000253 18.763744000000095 2
0.585071000000156
19.990973990008182 20.176637397515087 20.006453811093568 20.239355483903584 2
0.108089399093636
```

Out[18]:
```
20.0
```

Исправлено: добавлены сведения о смещённости выборочной дисперсии

Как мы видим, увеличение объёма выборки привело к очень хорошому приближению к теоретической дисперсии – отклонение составляет менее одного процента.

Выборочная дисперсия является смещённой оценкой – она смещена на $\frac{n+1}{n}$. Выборочная дисперсия стремится к теоретической дисперсии данной случайной величины.

Чтобы "исправить" "смещённость" выборочной дисперсии, вводят понятие несмещённой выборочной дисперсии, которая вычисляется следующим образом:

$$s^2 = \frac{n}{n+1}d$$

Выборочная дисперсия является состоятельной оценкой, так как она сходится по вероятности к оцениваемому параметру. Покажем это с помощью той дисперсии случайной величины $\sqrt{n}(\hat{\theta} - \theta)$, а предварительно посчитаем выборочные дисперсии и увидим, что они стремятся к нулю. С увеличением объёма выборки отклонения остаются в тех же пределах.

In [16]:
```
for i in range(len(SampleGeom)):
    varss = []
    for j in range(len(SampleGeom[i])):
        varss.append(VAR(SampleGeom[i][j]))
    print(VAR(varss))
```

```
100.81638399999999
164.96347999999995
31.186509445600063
1.9953192718107704
0.021461416472102816
```

In [21]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print(sqrt(len(SampleGeom[i][j]))*(20 - VAR(SampleGeom[i][j])),
              end = '\n' if j == len(SampleGeom[i])-1 else ' ')
```

```
-124.68315042538826 14.847491370598606 41.68030710059608 11.985324359398872 4
1.68030710059608
27.796420632880064 50.31183757327892 -6.198064213930026 -112.60870747859599
2.023857702507776
-22.61900000000022 6.799999999999926 26.023999999999923 64.62899999999999 -2
4.658999999999978
-192.67754621128788 -48.20576065160486 -12.603067877949478 39.093847310488194
-18.501569529128655
2.854274975758129 -55.857649611224126 -2.0408742644136653 -75.69084995870946
-34.18086920548295
```

Треугольное распределение

Аналогично с уже созданными функциями выборочных среднего и дисперсии найдём параметры для треугольного распределения и сравним их с теоретическими аналогами

In [24]:
```
print('Выборочные средние:')
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(MEAN(SampleTrig[i][j]), end = '\n' if j == len(SampleTrig[i])-1
else ' ')

print()
print('Теоретическое средние:', sts.triang.mean(0.2))

print()
print('Выборочные дисперсии:')
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(VAR(SampleTrig[i][j]), end = '\n' if j == len(SampleTrig[i])-1
else ' ')

print()
print('Теоретическая дисперсия:', sts.triang.var(0.2))
```

```
Выборочные средние:
0.3888020247237571 0.33176479310971396 0.40274091598076345 0.2645276769585177
0.4066336527889794
0.4241381767334792 0.33556962474563556 0.4179451463916902 0.4046077498624232
0.4108051733546144
0.3974512073447239 0.4127221793597357 0.3735019118802542 0.4130195630293778
0.43623634898860575
0.40070149229485835 0.4048340323481221 0.39855418225542183 0.3925837824890892
0.4003349692019517
0.4004729154237704 0.3994533168402419 0.4008760988626448 0.4001745223022422
0.39977482887658244

Теоретическое средние: 0.39999999999999997

Выборочные дисперсии:
0.08410033543620425 0.04897149411091285 0.04636443139243461 0.018291071269060
637 0.04356663308010121
0.037477037902815265 0.033856511391495384 0.05381762757513871 0.0490379630591
4886 0.06740700021693469
0.05581413274143447 0.0474823902725606 0.056743534512389913 0.044672725476870
76 0.040697633190056176
0.045924450304348725 0.04631749157614006 0.048228728859314604 0.0465666150156
6295 0.0505228034796794
0.046932521061367305 0.046655609789179917 0.04682636995791181 0.0470506971815
7668 0.046802946918012324

Теоретическая дисперсия: 0.04666666666666667
```

Исправлено: добавлено о смещённости выборочной дисперсии, иллюстрации данными

Как мы видим, увеличение объёма выборки привело к очень хорошому приближению к теоретическим параметрам – совпадение фиксируется до трёх знаков после запятой.

Выборочное среднее является несмещённой оценкой, так как ошибка оценки "в среднем" отсутствует – данные параметры стремятся к теоретическому аналогу данной случайной величины.

Выборочная дисперсия является смещённой оценкой – она смещена на $\frac{n+1}{n}$. Выборочная дисперсия стремится к теоретической дисперсии данной случайной величины. Чтобы "исправить" "смещённость" выборочной дисперсии, вводят понятие несмещённой выборочной дисперсии, которая вычисляется следующим образом: $s^2 = \frac{n}{n+1}d$

Оба параметра являются состоятельными оценками, так как они сходится по вероятности к оцениваемым параметрам. Покажем это далее с помощью введённой дисперсии случайной величины $\sqrt{n}(\hat{\theta} - \theta)$ и посчитаем выборочные средние полученных величин. С увеличением объёма выборки отклонения остаются в тех же пределах.

Заметим, что готовая функция среднего вернула значение 0.39999999999999997, когда значение по выведенной ранее функции мат.ожидания равно 0.4. Отличие несущественное, однако при определённых сравнениях это может сыграть важную роль.

В некоторых источниках – https://mipt.ru/diht/students/courses/mathematical_statistics.pdf (https://mipt.ru/diht/students/courses/mathematical_statistics.pdf) – введённая дисперсия называется асимптотической дисперсией.

In [14]:
```
print('Введённая дисперсия выборочные средних:')
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(sqrt(len(SampleTrig[i][j]))*(0.4-MEAN(SampleTrig[i][j])),
              end = '\n' if j == len(SampleTrig[i])-1 else ' ')
print()
print()
print('Введённая дисперсия выборочных дисперсий:')
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(sqrt(len(SampleTrig[i][j]))*(0.04667-VAR(SampleTrig[i][j])),
              end = '\n' if j == len(SampleTrig[i])-1 else ' ')
print()
```

```
Введённая дисперсия выборочные средних:
-0.10079258168225967 -0.1243579004797297 0.054687788242927984 -0.078408203632
8833 0.3374389835260962
0.051693461990854096 -0.19204096450244842 -0.4481843630520807 0.3531927719452
117 0.38630457193876316
0.11126908512844202 0.2839033140151437 -0.28213384307025746 0.120465616136799
51 0.0590275539644225
-0.11576051432745481 0.21280103586309818 0.0782329635330808 -0.18659233738981
82 0.045884409645745326
-0.01667590875546055 -0.17430156629377172 0.05719875788214185 -0.270531525568
4802 -0.1542873653017094


Введённая дисперсия выборочных дисперсий:
0.021785028443434 89 0.01396576600976319 0.05323784162225005 -0.06416038555982
004 0.08689202646299242
-0.042708792886868054 -0.019837591944246107 -0.004192382653797637 -0.04256899
8279369046 0.062032050170553354
-0.018547045198908096 0.022724477780484628 0.04080260624408319 0.024278026350
272736 0.033397105158467916
-0.07624071545566428 0.046577690328562286 -0.0021349176322075746 -0.099529039
75197154 -0.04306070799812741
0.02179765887669631 -0.06391262140806186 -0.06355717945816984 -0.038529042466
8915 -0.060250112719156354
```

In [23]:
```
print("Для выборчоного среднего: ")
for i in range(len(SampleTrig)):
    means = []
    for j in range(len(SampleTrig[i])):
        means.append(MEAN(SampleTrig[i][j]))
    print(VAR(means))

print("\n Для выборчоной дисперсии: ")
for i in range(len(SampleTrig)):
    varss = []
    for j in range(len(SampleTrig[i])):
        varss.append(VAR(SampleTrig[i][j]))
    print(VAR(varss))
```

```
Для выборчоного среднего:
0.005882353557849617
0.010196788213405356
0.00034692532210840716
2.0297659599086876e-05
1.370329194514013e-07

 Для выборчоной дисперсии:
0.0005069757258204282
0.0001489648664441369
4.244839698938253e-06
2.7329453619679594e-06
1.0707945562327829e-08
```

Распределение Ципфа

Аналогично с уже созданными функциями выборочных среднего и дисперсии найдём параметры для распределения Ципфа и сравним их с теоретическими аналогами.

Как видно, для этого распределения введённые методы анализа данных работают не очень качественно – дисперсия для выборочного среднего стремится к нулю слишком медленно, чтобы делать уверенные выводы.

In [19]:
```
def Harmonic(N, s):
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    return(H)

print('Выборочные средние:')
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        print(MEAN(SampleZipf[i][j]), end = '\n' if j == len(SampleZipf[i])-1
else ' ')

print()
print('Теоретическое среднее:', Harmonic(500,1)/Harmonic(500,2))

print()
print('Выборочные дисперсии:')
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        print(VAR(SampleZipf[i][j]), end = '\n' if j == len(SampleZipf[i])-1
else ' ')

print()
print('Теоретическая дисперсия:',
      Harmonic(500,0)/Harmonic(500,2) - (Harmonic(500,2)**1)/(Harmonic(500,2)
**2))
```

```
Выборочные средние:
6.6 1.8 1.6 1.0 1.2
6.9 2.4 2.7 3.7 2.0
3.03 8.89 3.59 3.05 9.19
3.987 4.199 3.155 3.625 4.616
4.15645 4.12489 4.10575 4.11521 4.09122

Теоретическое среднее: 4.134563463893318

Выборочные дисперсии:
76.63999999999999 1.3600000000000003 0.24 0.0 0.16000000000000006
83.69 4.4399999999999995 6.610000000000001 34.61 2.2
37.149099999999954 1595.237900000003 63.76189999999995 55.40750000000005 182
9.493900000001
217.66483099999968 225.46739900000017 218.68497499999071 236.600375 486.97254
39999956
298.0080933979467 282.3638724880423 282.6813469372524 284.9742966554833 276.0
8473891170365

Теоретическая дисперсия: 303.7245395447656
```

In [37]:
```
print("Для выборчоного среднего: ")
for i in range(len(SampleZipf)):
    means = []
    for j in range(len(SampleZipf[i])):
        means.append(MEAN(SampleZipf[i][j]))
    print(VAR(means))

print("\n Для выборчоной дисперсии: ")
for i in range(len(SampleZipf)):
    varss = []
    for j in range(len(SampleZipf[i])):
        varss.append(VAR(SampleZipf[i][j]))
    print(VAR(varss))
```

```
Для выборчоного среднего:
4.4064000000000005
3.1384000000000007
8.16944
0.24778623999999994
0.00047968230400000924

 Для выборчоной дисперсии:
929.2620799999997
961.80668
667112.652817256
11059.496376291825
52.172153998561825
```

In [20]:
```
print('Введённая дисперсия выборочные средних:')
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        print(sqrt(len(SampleZipf[i][j]))*(4.13-MEAN(SampleZipf[i][j])),
              end = '\n' if j == len(SampleZipf[i])-1 else ' ')
print('\n Введённая дисперсия выборочных дисперсий:')
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        print(sqrt(len(SampleZipf[i][j]))*(303.7-VAR(SampleZipf[i][j])),
              end = '\n' if j == len(SampleZipf[i])-1 else ' ')
print()
```

```
Введённая дисперсия выборочные средних:
-5.5230879044244805 5.21003838757451 5.657251983074468 6.998892769574342 6.55
1679174074383
-8.759509118666413 5.470740352091297 4.522057054040782 1.3597793938724023 6.7
35651416158648
11.0 -47.60000000000001 5.4 10.8 -50.599999999999994
4.522057054040776 -2.1819715855161803 30.8322071866417 15.969502183850311 -1
5.368669428418317
-8.364224411145532 1.6159238843460957 7.668523325908424 4.677008659388918 12.
263312766132987

 Введённая дисперсия выборочных дисперсий:
507.72159497110226 676.0527923172864 678.5571884520862 679.0938447666862 678.
7360738902861
695.7327080136452 946.3432125819892 939.4810700594238 850.9372955747092 953.4
267145407664
2665.509 -12915.37900000003 2399.3810000000003 2482.9249999999993 -15257.9390
0000001
2720.6709291751204 2473.9320643916585 2688.411143361856 2121.876451431756 -57
95.586716134125
1799.9389091437997 6747.075938556769 6646.681702715644 5921.587335730553 873
2.712321923664
```

Исправлено: добавлено о смещённости выборочной дисперсии, иллюстрации данными

К сожалению, в строенном модуле stats готовые методы оказались не в состоянии подсчитать выборочные средние и дисперсии для данного распределения, поэтому автором данной работы было принято решение создать функцию гармонического числа и с её помощью, а также с помощью выведенных ранее формул мат.ожидания и дисперсии получить собственные значения искомых параметров.

Как мы видим, увеличение объёма выборки привело к хорошому приближению к теоретическим параметрам, но существенное расхождение остаётся. Однако видно, что увеличение объёма выборки приводит к всё более и более точным результатам.

Выборочное среднее является несмещённой оценкой, так как ошибка оценки "в среднем" отсутствует – данные параметры стремятся к теоретическим аналогам данной случайной величины.

Выборочная дисперсия является смещённой оценкой – она смещена на $\frac{n+1}{n}$. Выборочная дисперсия стремится к теоретической дисперсии данной случайной величины. Чтобы "исправить" "смещённость" выборочной дисперсии, вводят понятие несмещённой выборочной дисперсии, которая вычисляется следующим образом: $s^2 = \frac{n}{n+1}d$

Оба параметра являются состоятельными оценками, так как они сходится по вероятности к оцениваемым параметрам.

Это показано с помощью введённой дисперсии случайной величины $\sqrt{n}(\hat{\theta} - \theta)$. С увеличением объёма выборки для выборочного среднего отклонения остаются в тех же пределах.

3.2. Нахождение параметров распределений событий

Геометрическое

$P(x) = P(\xi = x) = p \cdot q^{x-1}; \theta = p$

Проверим условия регулярности

1. $L(\hat{x}, \theta) = \prod_{i=1}^{n}(1-\theta)^{x_i-1}\theta = (1-\theta)^{\sum_{i=1}^{n}x_i-n} \cdot \theta^n$

$L(\hat{x}, \theta) > 0$, так как каждый "множитель" функции правдоподобия (функция вероятности) принимает только неотрицательные значения.

1. $V(x, \theta) = \frac{\partial lnL}{\partial \theta} = \frac{\partial(\sum_{i=1}^{n}x_i-n)ln(1-\theta)+nln\theta}{\partial\theta} = \frac{n}{\theta} - \frac{\sum_{i=1}^{n}x_i-n}{1-\theta}$

$M_\theta V^2(x, \theta) = M(-\frac{n}{\theta^2} - \frac{\sum_{i=1}^{n}x_i-n}{(1-\theta)^2}) = (-\frac{n}{\theta^2} - \frac{1}{(1-\theta)^2}(M(\sum_{i=1}^{n}x_i) - n)) = n(\frac{1}{\theta^2} + \frac{1}{(1-\theta)\theta})$ – ограничена.

Исправлено: изменён третий пункт регулярности

1. $\frac{\partial}{\partial\theta}\sum\limits_{x=0}^{\infty} T(\hat{x})L(\hat{x},\theta) = \sum\limits_{x=0}^{\infty} T(\hat{x})\frac{\partial L(\hat{x},\theta)}{\partial\theta}$

Докажем:

$$\frac{\partial}{\partial\theta}\sum_{x=1}^{\infty}(1-\theta)^{\sum_{i=1}^{n}x_i-n}\cdot\theta^n = \sum_{x=1}^{\infty}\frac{\partial(1-\theta)^{\sum_{i=1}^{n}x_i-n}\cdot\theta^n}{\partial\theta}$$

$$\frac{\partial}{\partial\theta}\sum_{x=1}^{\infty}(1-\theta)^{c}\cdot\theta^n = \sum_{x=1}^{\infty}\frac{\partial(1-\theta)^{c}\cdot\theta^n}{\partial\theta}$$

$$\sum_{x=1}^{\infty}\frac{\partial}{\partial\theta}(1-\theta)^{c}\cdot\theta^n = \sum_{x=1}^{\infty}\frac{\partial(1-\theta)^{c}\cdot\theta^n}{\partial\theta}$$

$$\sum_{x=1}^{\infty} n(1-\theta)^{c}\theta^{n-1} - c(1-\theta)^{c-1}\theta^n = \sum_{x=1}^{\infty} n(1-\theta)^{c}\theta^{n-1} - c(1-\theta)^{c-1}\theta^n$$

$$\sum_{x=1}^{\infty} n(1-\theta)^{\sum_{i=1}^{n}x_i-n}\theta^{n-1} - (\sum_{i=1}^{n}x_i-n)(1-\theta)^{\sum_{i=1}^{n}x_i-n-1}x^{\sum_{i=1}^{n}x_i-n} = \sum_{x=1}^{\infty} n(1-\theta)^{\sum_{i=1}^{n}x_i-n}\theta^{n-1} - (\sum_{i=1}^{n}x_i-n)(1-\theta)^{\sum_{i=1}^{n}$$

Доказано.

Оно принадлежит экспоненциальному семейству

$$f_\theta(x) = e^{ln((1-\theta)^{x_i-1}\theta)} = e^{(x_i-1)ln(1-\theta)+ln\theta}$$

Исправлено: добавлены обоснования для некоторых свойств оценок

Получим оценку параметра методом моментов

$\alpha_1(\theta) = \hat{\alpha}_1$. Для данного распределения $M\xi = \frac{1}{p}$. $p = \frac{1}{M\xi}$, тогда

$$\hat{\theta} = \frac{1}{\alpha_1} = \frac{1}{\frac{1}{n}\sum\limits_{i=1}^{n}x_i}$$

Введём параметрическую функцию $\tau(\theta) = \frac{1}{\theta}$

Оценка несмещённая: $M\tau(\theta) = M\frac{1}{\frac{1}{\alpha_1}} = M\bar{X} = M\frac{1}{n}\sum\limits_{i=1}^{n}x_i = \frac{1}{n}\sum\limits_{i=1}^{n}Mx_i = MX_i = M\xi = \frac{1}{\theta}$.

Состоятельная (так как оценка несмещённая, достаточно просто посмотреть, что дисперсия стремится к нулю): $D\bar{X} = D\frac{1}{n}\sum\limits_{i=1}^{n}x_i = \frac{1}{n^2}\sum\limits_{i=1}^{n}DX_i = \frac{1}{n^2}nD\xi = \frac{1-\theta}{n\theta^2}$, стремится к нулю.

Оптимальная: Оптимальная оценка если существует, то является функцией достаточной статистики. Применим критерий факторизации: $L(\hat{x},\theta) = h(x)\,g(\theta, T(x))$

$$\prod_{i=1}^{n}(1-\theta)^{x_i-1}\theta = (1-\theta)^{\sum_{i=1}^{n}x_i-n}\cdot\theta^n$$

$T(x) = \sum_{i=1}^{n} x_i$ – достаточная статистика.

Находим такое ϕ, что $M_\theta \phi(T(x)) = \tau(\theta)$.

Проверим предложенную оценку (методом моментов): $\phi(T(x)) = \frac{1}{\frac{1}{n}(T(x))} = \frac{1}{n}T(x)$.

$M_\theta \frac{1}{n} T(x) = M_\theta \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{\theta} = \tau(\theta)$. По теореме о полноте экспоненциальных семейств, T(x) – достаточная и полная статистика. Поэтому функция от неё $\frac{1}{n}T(x)$ является оптимальной оценкой для $\frac{1}{\theta}$

Эффективная: так как принадлежит экспоненциальному семейству, то $T(X) = \sum_{i=1}^{n} B(X_i) = \sum_{i=1}^{n} (x_i - 1) = \bar{X} - 1$ является эффективной оценкой параметра $\tau(\theta) = -\frac{C'(\theta)}{A'(\theta)} = -\frac{ln'(\theta)}{ln'(1-\theta)} = \frac{1-\theta}{\theta}$

$\bar{X} - 1 = \frac{1-\theta}{\theta}$

$\bar{X} = \frac{1}{\theta}$ – эффективная оценка.

Сравним с истинным значением оцениваемого параметра

Мы оценили величину, по своему значению обратную параметру геометрического распределения. Выборочные средние мы уже посчитали, так что значения оцениваемого параметра посчитать легко. Выведим их:

In [28]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print(1/MEAN(SampleGeom[i][j]),
              end = '\n' if j == len(SampleGeom[i])-1 else ' ')
```

```
0.22727272727272727 0.2 0.13157894736842105 0.25 0.22727272727272727
0.14705882352941177 0.2222222222222222 0.1234567901234568 0.2777777777777778
0.17543859649122806
0.18315018315018314 0.186219739292365 0.23148148148148145 0.17825311942959002
0.1663893510815308
0.19857029388403497 0.2021018593371059 0.20889910173386256 0.1960784313725490
4 0.21838829438742086
0.19991363730868264 0.20071857248951244 0.2001465072433021 0.1992170768878308
0.1999008491788073
```

Полученные значения приближаются к истинному с точностью до 3 знаков после запятой, очень хороший результат. С увеличением n очность повышается. Данная оценка является состоятельной, несмещённой, оптимальной и эффективной. С помощью выборочной дисперсии убедимся, как быстро она стремится к нулю.

In [57]:
```
for i in range(len(SampleGeom)):
    thetas = []
    for j in range(len(SampleGeom[i])):
        thetas.append(1/MEAN(SampleZipf[i][j]))
    print(VAR(thetas))
```

```
0.0825423426181002
0.015062927919565278
0.010085152235006233
0.0011874427164043383
1.6567059130719302e-06
```

Треугольное

$$f_\theta(x) = \begin{cases} 0 & \text{если } x \in [-\infty, 0] \\ \frac{2 \cdot x}{\theta} & \text{если } x \in [0, \theta] \\ \frac{2 \cdot (1-x)}{1-\theta} & \text{если } x \in [\theta, 1] \\ 0 & \text{если } x \in [1, +\infty] \end{cases}$$

$$\theta \in (0, 1)$$

</h1>

Регулярность

1. $L(\hat{x}, \theta) = \prod_{i=1}^{n} \frac{2 \cdot x_i}{\theta} Ind(x_i \in [0, \theta]) + \prod_{i=1}^{n} \frac{2 \cdot (1-x_i)}{1-\theta} Ind(x_i \in [\theta, 1])$

$L(\hat{x}, \theta) > 0$, так как каждый "множитель" функции правдоподобия (функция вероятности) принимает только неотрицательные значения.

1. $i(\theta) = M_\theta[(\frac{\partial}{\partial\theta} \ln(f_\theta(x))^2] = \frac{1}{\theta^2} P(X \le \theta) + \frac{1}{(1-\theta)^2}[1 - P(X \le \theta)] = \frac{1}{\theta} + \frac{1}{1-\theta} = \frac{1}{(1-\theta)\theta}$

$M_\theta V^2(x, \theta) = D_\theta V(x, \theta) = i_n(\theta) = ni(\theta) = \frac{n}{(1-\theta)\theta}$ – ограничена для $\theta \in (0; 1)$.

Исправлено: изменён третий пункт регулярности

1. $\frac{\partial}{\partial\theta} \int T(\hat{x}) L(\hat{x}, \theta) d\hat{x} = \int T(\hat{x}) \frac{\partial L(\hat{x}, \theta)}{\partial\theta} d\hat{x}$

Рассмотрим сформулируем эквивалентное условие регулярности:

Существует такое C, что $P_\theta(\xi in C) = 1$, при каждом $y \in C$ функция $\sqrt{f_\theta(y)}$ непрерывно дифференцируема по θ всюду в области Θ.

$$\frac{\partial}{\partial\theta}\sqrt{f_\theta(y)} = \frac{\partial}{\partial\theta}\sqrt{\frac{2 \cdot y}{\theta}} Ind(x_i \in (0, \theta)) + \frac{\partial}{\partial\theta}\sqrt{\frac{2(1-y)}{1-\theta}} Ind(x_i \in (\theta, 1)) = \frac{1}{(\sqrt{2}\sqrt{\theta y})} Ind(x_i \in (0, \theta)) + \frac{1}{\sqrt{2}(\theta-1)\sqrt{\frac{(y-1)}{(-1+p)}}} Ind(x_i \in$$

Определяем $C = (0; \theta) \cup (\theta; 1)$ – область $[0; 1]$, из которых исключили три точки, вероятсность попасть в которые равна 0. Для этой области функция всегда имеет производную. Значит, модель регулярна.

Не принадлежит экспоненциальному семейству

$f_\theta(x) = e^{ln(\frac{2 \cdot x}{\theta} Ind(x \in [0, \theta]) + \frac{2 \cdot (1-x)}{1-\theta} Ind(x \in [\theta, 1]))} = e^{ln(2x) - ln(\theta)} Ind(x \in [0, \theta]) + e^{ln(2(1-x)) - ln(1-\theta)} Ind(x \in [\theta, 1]$
. Увы, индикатора не получилось "разбить" на произведение функций от x и θ.

Получим оценку параметра методом моментов

$\alpha_1(\theta) = \hat{\alpha}_1$. Для данного распределения $M\xi = \frac{p+1}{3}$. $p = 3M\xi - 1$, тогда

$$\hat{\theta} = 3\hat{\alpha}_1 - 1 = 3(\frac{1}{n}\sum_{i=1}^{n} x_i) - 1$$

Оценка несмещённая: $M_\theta 3\hat{\alpha}_1 - 1 = 3\hat{M}_\theta\alpha_1 - 1 = 3\hat{M}_\theta(\frac{1}{n}\sum_{i=1}^{n} x_i) - 1 = 3\hat{(}\frac{1}{n}\sum_{i=1}^{n} M_\theta x_i) - 1 = 3M\xi - 1 = \theta$.

Состоятельная (так как оценка несмещённая, достаточно просто посмотреть, что дисперсия стремится к нулю): $D(3\hat{\alpha}_1 - 1) = 3^2 D\hat{\alpha}_1 = 9D\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{9}{n^2}\sum_{i=1}^{n} Dx_i = \frac{9}{n^2}nDx = \frac{9Dx}{n} = \frac{9(p^2-p+1)}{18n}$, стремится к нулю.

Оптимальная: Оптимальная оценка если существует, то является функцией достаточной статистики. Применением критерий факторизации: $L(\hat{x}, \theta) = h(x)\, g(\theta, \mathrm{T}(x))$

$$L(\hat{x}, \theta) = \prod_{i=1}^{n} \frac{2 \cdot x_i}{\theta} Ind(x_i \in [0, \theta]) + \prod_{i=1}^{n} \frac{2 \cdot (1 - x_i)}{1 - \theta} Ind(x_i \in [\theta, 1])$$

Чтобы "вытащить" из выражения достаточную статистику, попробуем подумать логически. Рассмотрим вариационный ряд выборки. В вариационном ряду найдём следующий элемент:
$X_{(y)} = min(X_{(1)} \ldots X_{(n)}, X_{(i)} > \theta)$, то есть наименьший элемент выборки, больший оцениваемого параметра. К сожалению, предугадать этот элемент невозможно, поэтому сделаем достаточной статистикой всю выборку, так как любой её элемент может оказаться $X_{(y)}$

$T(x) = X = (X_1, \ldots X_n)$ – достаточная статистика.

Находим такое ϕ, что $M_\theta\phi(T(x)) = \theta$.

Проверим предложенную оценку (методом моментов):
$\phi(T(x)) = 3(\frac{1}{n}\sum_{i=1}^{n} x_i) - 1 = 3(\frac{1}{n}\sum(X_1, \ldots X_n)) - 1 = 3(\frac{1}{n}\sum T(x)) - 1.$

$$M_\theta 3(\frac{1}{n}\sum T(x)) - 1 = M_\theta(3(\frac{1}{n}\sum_{i=1}^{n} x_i) - 1) = \theta.$$

Полнота: $M\phi(T) = 0$ только в том случае, если каждый элемент выборки равен нулю. Значит, статистика полна и достаточна. Оптимальная оценка как функция от неё: $3(\frac{1}{n}\sum T(x))$

К сожалению, эффективную оценку построить не удалось.

Сравним с истинным значением оцениваемого параметра

Оценка параматра есть функция от выборочного среднего. Выборочные средние мы уже посчитали, так что значения оцениваемого параметра посчитать легко. Выведим их:

In [101]:
```
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(int((3*MEAN(SampleTrig[i][j])-1)*10000)/10000,
              end = '\n' if j == len(SampleTrig[i])-1 else ' ')
```

```
0.064 0.0518 0.1995 0.7127 0.5271
-0.1834 0.0149 -0.1118 0.5189 0.2131
0.2229 0.2288 0.3001 0.2077 0.2889
0.1851 0.2531 0.1787 0.1896 0.1621
0.2024 0.202 0.2011 0.1975 0.1998
```

Как мы видим, с точностью до сотых мы приблизились к истинному значению параметра. При увлеичении объёма выборки она стремится всё ближе к значению 0.2. Оценка является состоятельной, несмещённой, оптимальной.

Об эффективности можем судить косвенно, изучив, насколько быстро убывают отклонения в виде выборочных дисперсий. С ростом объёма выборки отклонения действительно стремятся к нулю.

In [58]:
```
for i in range(len(SampleTrig)):
    thetas = []
    for j in range(len(SampleTrig[i])):
        thetas.append(3*MEAN(SampleTrig[i][j])+1)
    print(VAR(thetas))
```

```
0.05294118202064655
0.09177109392064822
0.003122327898975661
0.00018267893639178201
1.2332962750628467e-06
```

Распределение Ципфа

$P(x) = \frac{x^{-\theta}}{H_{N,\theta}}, x \in \{1, 2...N\}, H_{N,\theta} = \sum_{n=1}^{N} n^{-\theta}$. В встроенном модуле $N \to \infty$, поэтому во избежание с расхождениями в расчётах будем придерживаться значения $H_{N\to\infty,s} = \sum_{n=1}^{\infty} n^{-\theta} = \zeta(\theta)$. $P(x) = \frac{x^{-\theta}}{\zeta(\theta)}$

Регулярность

1. $L(\hat{x}, \theta) = \prod_{i=1}^{n} \frac{x_i^{-\theta}}{\zeta(\theta)}$

$L(\hat{x}, \theta) > 0$, так как каждый "множитель" функции правдоподобия (функция вероятности) принимает только неотрицательные значения.

1. $i(\theta) = M_\theta[(\frac{\partial}{\partial\theta}\ln(f_\theta(x))^2] = M_\theta[(\frac{\partial}{\partial\theta}\ln(\frac{x^{-\theta}}{\zeta(\theta)})^2] = M_\theta[(\frac{\partial}{\partial\theta}\ln(-\frac{\ln x_i}{x_i^\theta\zeta(\theta)} + \frac{\sum_{k=2}^{\infty}\frac{\ln k}{k^\theta}}{x_i^\theta\zeta^2(\theta)})^2]$

$M_\theta V^2(x, \theta) = D_\theta V(x, \theta) = i_n(\theta) = nM_\theta[(\frac{\partial}{\partial\theta}\ln(-\frac{\ln x_i}{x_i^\theta\zeta(\theta)} + \frac{\sum_{k=1}^{\infty}\frac{\ln k}{k^\theta}}{x_i^\theta\zeta^2(\theta)})^2]$ – ограничена как композиция ограниченных функций, не принимающих нулевые значения.

1. $\frac{\partial}{\partial\theta}\int T(\hat{x})L(\hat{x}, \theta)d\hat{x} = \int T(\hat{x})\frac{\partial L(\hat{x},\theta)}{\partial\theta}d\hat{x}$

Величины θ и X взаимно независимы. Поэтому это выражение можно рассматривать как производную от интеграла по параметру. В таком случае операции интегрирования и дифференцирования допускается менять местами.

Принадлежит экспоненциальному семейству

$f_\theta(x) = e^{ln(\frac{x^{-\theta}}{\zeta(\theta)})} = e^{ln(x^{-\theta}) - ln(\zeta(\theta))} = e^{-\theta ln(x) - ln(\zeta(\theta))}$

Получим оценку параметра методом максимального правдоподобия

$L(\hat{x}, \theta) = \prod_{i=1}^{n} \frac{x_i^{-\theta}}{\zeta(\theta)}$

$lnL(\hat{x}, \theta) = \sum_{i=1}^{n} ln\frac{x_i^{-\theta}}{\zeta(\theta)} = \sum_{i=1}^{n}(ln(x_i^{-\theta}) - ln(\zeta(\theta))) = \sum_{i=1}^{n}(-\theta ln(x_i) - ln(\zeta(\theta))) = -\theta\sum_{i=1}^{n} ln(x_i) - nln(\zeta(\theta))$

$\frac{\partial}{\partial\theta}lnL(\hat{x}, \theta) = -\sum_{i=1}^{n} ln(x_i) - n\frac{1}{\zeta(\theta)} \cdot (-(\sum_{k=2}^{\infty}\frac{\ln k}{k^\theta}))$

$\frac{\partial}{\partial\theta}lnL(\hat{x}, \theta) = 0$

$\frac{n}{\zeta(\theta)}(\sum_{k=2}^{\infty}\frac{\ln k}{k^\theta}) - \sum_{i=1}^{n} ln(x_i) = 0$

Обозначим $\tau(\theta) = \frac{\sum_{k=2}^{\infty}\frac{\ln k}{k^\theta}}{\zeta(\theta)}$. Тогда оценкой этого параметра будет $\tau(\hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} ln(x_i)$

Свойства данной оценки.

Воспользуемся критерием факторизации: $L(\hat{x}, \theta) = h(x)\, g(\theta, \mathrm{T}(x))$

$h(x)\, g(\theta, \mathrm{T}(x)) = -\theta \sum_{i=1}^{n} ln(x_i) - n ln(\zeta(\theta))$

Откуда явно можем видеть, что $T(x) = \sum_{i=1}^{n} ln(x_i)$ – достаточная статистика.

Существует такое ϕ, что $M_\theta \phi(T(x)) = \frac{\sum_{k=2}^{\infty} \frac{\ln k}{k^\theta}}{\zeta(\theta)}$.

По теореме о полноте экспоненциальных семейств, T(x) – достаточная и полная статистика.

Эффективная: так как принадлежит экспоненциальному семейству, то $T(X) = \sum_{i=1}^{n} B(X_i) = \sum_{i=1}^{n} ln(x_i)$ является эффективной оценкой параметра $\tau(\theta) = -\frac{C'(\theta)}{A'(\theta)} = -\frac{ln'(\zeta(\theta))}{(-\theta)'} = \frac{1}{\zeta(\theta)}$

Сравним с истинным значением оцениваемого параметра

Изначально мы занимались оцениванием параметра θ. Значит, необходимо получить значения именно его. Поэтому сначала получим функцию $\tau(\theta)$

In [377]:
```
def tau(theta):
    sumUp = 0
    sumDown = 0
    for k in range (2, 500):
        sumUp += np.log(k)/(k**theta)
    for k in range (1, 500):
        sumDown += k**(-theta)
    return sumUp/sumDown
```

Искомый параметр можно получить из уравнения $\tau(\hat{\theta}) = \frac{1}{n} \sum_{i=1}^{n} ln(x_i)$

Заметим, что $\frac{1}{n} \sum_{i=1}^{n} ln(x_i)$ есть константа для каждой конкретной выборки.

Посчитаем и запомним эти значения для каждой из имеющихся выборок. Они являются "целевыми" – эти значения должна достигнуть $\tau(\theta)$

In [73]:
```
aims = []

for i in range(5):
    for j in range(5):
        aim = 0
        for el in SampleZipf[i][j]:
            aim += np.log(el)
        aim = aim/len(SampleZipf[i][j])
        aims.append(aim)
```

Решим наше уравнение методом дихотомии. Сначала ищем параметр на промежутке $[0; 10]$. Если не найдём (признаком этого будет, что наше значение слишком близко в пределах установленной точности к правому краю), то повторяем алгоритм на $[10; 20]$, $[20; 30]$ и так далее, пока не найдём наше значение.

In [375]:
```
def getTheta(aim, eps = 1e-3):
    x = 9.9999
    a0 = 0
    b0 = 10
    while (abs(b0 - x) < eps):
        a = a0
        b = b0
        a0 += 10
        b0 += 10
        while abs(b - a) > eps:
            x = (a + b) / 2.0
            fx = tau(x) - aim
            fa = tau(a) - aim
            if (fx < 0 and fa < 0) or (fx > 0 and fa > 0):
                a = x
            else:
                b = x
    return x
```

Наконец выводим значения оцениваемого параметра

In [65]:
```
for i in range(25):
    print(int(getTheta(aims[i])*10000)/10000,
          end = '\n' if i%5 == 4 else ' ')
```

```
1.5362 2.2125 2.2125 9.9993 3.1683
1.5826 1.9769 1.9061 1.9024 2.0843
2.0208 1.8133 1.9866 2.0782 1.8927
1.9647 1.961 2.0904 2.033 2.0086
2.0013 2.0025 2.0037 2.0025 2.0074
```

Как и в случае предыдущих распределений, имеем значения, весьма близко соответствующие истинному значению параметра – 2. Выборочная дисперсия убывает весьма хорошо.

In [76]:
```
for i in range(5):
    thetas = []
    for j in range(5):
        thetas.append(int(getTheta(aims[i*5+j])*10000)/10000)
    print(VAR(thetas))
```

```
9.7984300864
0.028042154399999984
0.008882301599999993
0.002287270399999994
4.417600000000071e-06
```

3.3. Работа с данными*

Геометрическое распределение

К сожалению, автору работы не удалось обнаружить данных, в точности совпадающих с описанными моделями в первой домашней работе. Поэтому рассмотрим данные, модель описания которых весьма близка к примерам 1.1 и 1.2. Так что детально новая модель разбираться не будет, так как идейно имеет тот же принцип.

Рассмотрим набор данных о компьютерных сетях оптических соединений. Данные доступны по ссылке: https://www.kaggle.com/inancigdem/optical-interconnection-network?select=optical_interconnection_network.csv (https://www.kaggle.com/inancigdem/optical-interconnection-network?select=optical_interconnection_network.csv)

Исследоваться будет время ответа сервера. То есть промежуток времени между отправкой сообщения на клиенте и ответом самого сервера. Значение данной величины зависит от качества связи, состояния оптического кабеля передачи данных, расстояния между клиентом и сервером, количества коммутаторов и маршрутизаторов между хостами.

"Разобьём" исследуемую компьютерную сеть на участки одинаковой длины. Для простоты модели будем считать сеть однородной, и вероятность встретить нужный сервер на этом участке равна p. Соответственно, если с вероятностью $q = 1 - p$ на этом участке сервер не встречен, но на следующем участке остаётся эта же вероятность p встретить сервер.

Поставим вопрос на языке вероятности:

Какова вероятность того, что сервер будет встречен на x-ом участке? Считая события встречи (или её отсутствия) сервера независимыми, мы можем перемножить вероятности каждого из событий. Тогда

$$p(x) = p \cdot q^{x-1}$$

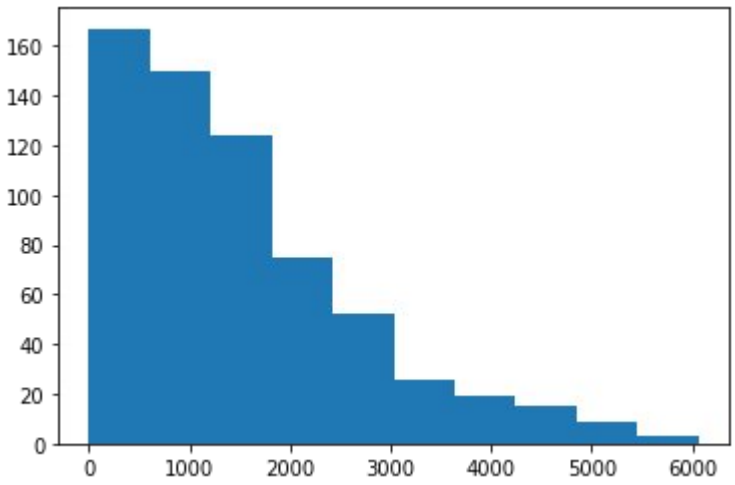
Заметим, что время ответа t сервера прямо пропорционально расстоянию, которое проходит сообщение по сети: грубо говоря, $x = const \cdot t$. Тогда мы можем выписать модель для времени:

$$p(const \cdot t) = p \cdot q^{const \cdot t - 1}$$

Начнём исследование. Загрузим данные в список $durs$ и отобразим их на гистограмме.

In [207]:
```
import csv

durs = []
with open('optical_interconnection_network.csv', newline='') as File:
    reader = csv.DictReader(File)
    for row in reader:
        durs.append(int(row['Network Response Time'][:row['Network Response T
ime'].find(",")]))

import matplotlib.pyplot as plt
plt.hist(durs, bins = 10)
```

Out[207]:
```
(array([167., 150., 124.,  75.,  52.,  26.,  19.,  15.,   9.,   3.]),
 array([   0. ,  606.5, 1213. , 1819.5, 2426. , 3032.5, 3639. , 4245.5,
        4852. , 5458.5, 6065. ]),
 <a list of 10 Patch objects>)
```



Выборочное среднее и выборочная дисперсия

Ранее мы писали функции для их нахождения, воспользуемся ими же.

In [208]:
```
print(MEAN(durs), VAR(durs))
```

```
1503.740625 1444006.0014746047
```

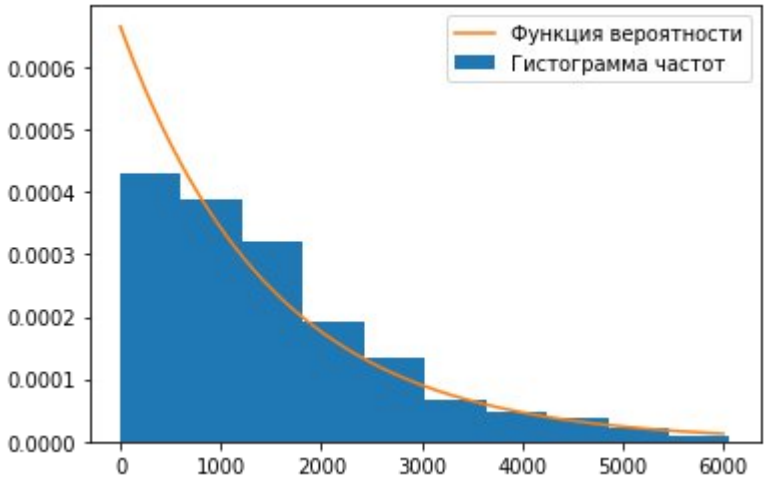
Получим значение предложенной ранее оценки

In [109]:
```
p_geom = 1/MEAN(durs)
print(p_geom)
```

```
0.000665008302213023
```

Отобразим визуально полученную функцию плотности распределния вместе с гистограммой частот

In [150]:
```
plt.hist(durs, bins = 10, density=True, label='Гистограмма частот')
plt.plot(range(1, 6000), sts.geom.pmf(range(1, 6000), p_geom), label='Функция вероятности')
plt.legend(loc='upper right')
plt.show()
```



Графически получили весьма похожее поведение величины. По мере увеличения времени ответа сервера гистограмма начинает всё больше напоминать график полученной функции вероятнсости.

Конечно, полное соответствие модели геометрического распределения получить невозможно – предложенная автором модель сильно упрощает реальную ситуацию, связанную с распределением времени ответа сервера. Ведь не учитывается множество факторов – неравномерное распределение элементов сети, неравномерное распределение нагрузки серверов и так далее. Так что полученный результат пускай и не точно, но соответстсвует полученным значениям.

Знание того, как меняется данная величина, необходимо при проектировании компьютерных сетей и разработке протоколов передачи данных (например, при выборе размера передаваемого пакета данных).

Треугольное распределение

Вспомним пример 2.2. Треугольное распределение можно использовать при недостатке информации о случайной величине. Пусть распределение случайной величины неравномерное, а плотность с какой-то точки начинает заметно возрастать, а пройдя максимум – заметно убывать, где после какой-то точки она постепенно близится к нулю. Если посмотреть на график плотности, то отметив точку, где он начинает резко возрастать, как 0, а точку максимума как p, и точку, где функция перестаёт резко убывать, как 1, то получится треугольное распределение.

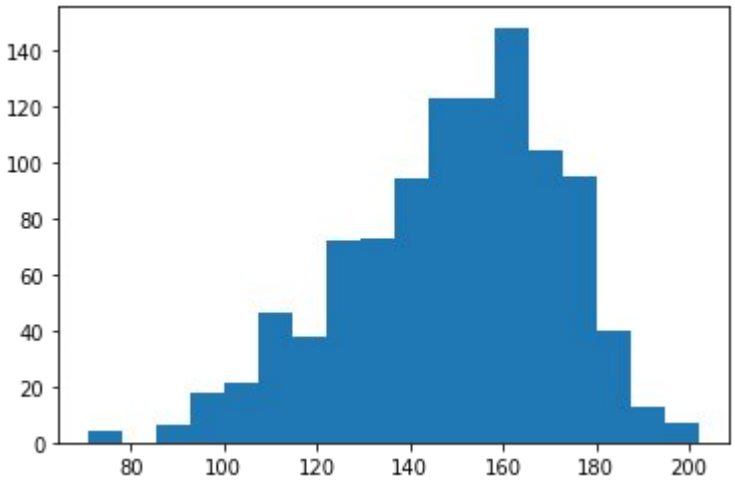
В медицинских данных часто зависимость случайных величин невозможно определить точно и однозначно – слишком много факторов влияет на какой-либо признак. Поэтому обратимся к набору данных о сердечных заболеваниях. Этот набор данных датируется 1988 годом и состоит из четырех баз данных: Кливленд, Венгрия, Швейцария и Лонг-Бич V. Он содержит большое количество данных о пациентах, обратим внимание на критерий, где фиксировалась максимальная частота сердцебиения.

Набор данных представлен по ссылке: https://www.kaggle.com/johnsmith88/heart-disease-dataset (https://www.kaggle.com/johnsmith88/heart-disease-dataset)

Представим данные граффически

In [348]:
```
import csv

rateTrig = []
rates = []
with open('heart.csv', newline='') as File:
    reader = csv.DictReader(File)
    for row in reader:
        rates.append(int(row['thalach']))

import matplotlib.pyplot as plt
plt.hist(rates, bins = (max(rates)-min(rates))//7)
```

Out[348]:
```
(array([  4.,   0.,   6.,  18.,  21.,  46.,  38.,  72.,  73.,  94., 123.,
        123., 148., 104.,  95.,  40.,  13.,   7.]),
 array([ 71.        ,  78.27777778,  85.55555556,  92.83333333,
        100.11111111, 107.38888889, 114.66666667, 121.94444444,
        129.22222222, 136.5       , 143.77777778, 151.05555556,
        158.33333333, 165.61111111, 172.88888889, 180.16666667,
        187.44444444, 194.72222222, 202.        ]),
 <a list of 18 Patch objects>)
```



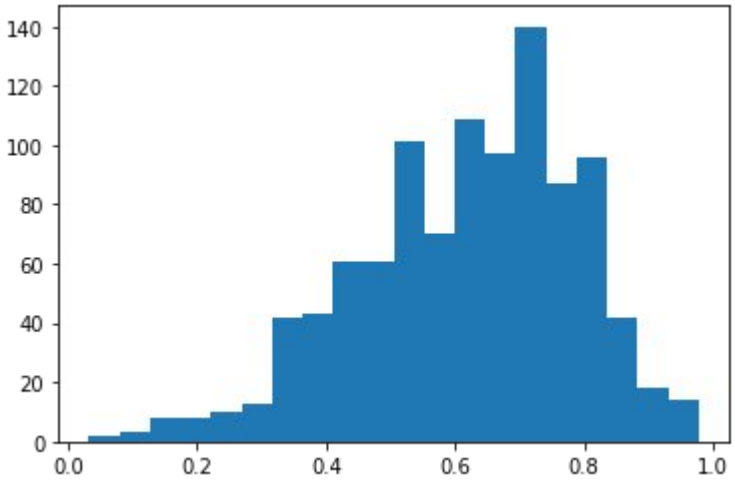
Вспомним, что описанное нами треугольное распределение "работает" со случайными величинами в диапазоне $[0, 1]$, поэтому напишем функции, которые пропорционально переведут полученные данные в нужный диапазон и обратно:

In [316]:
```
def ratesToTrig(rates):
    new = rates
    for i in range(len(rates)):
        new[i] = rates[i] - min(rates)
    for i in range(len(rates)):
        new[i] = new[i]/max(new)
    return new

def ratesFromTrig(rates, max, min):
    new = rates
    for i in range(len(rates)):
        new[i] = new[i]*max
    for i in range(len(rates)):
        new[i] = rates[i] + min
    return new
```

Обработанные данные графически

In [349]:
```
rateTrig = ratesToTrig(rates)
plt.hist(rateTrig, bins = 20)
```

Out[349]:
```
(array([  2.,   3.,   8.,   8.,  10.,  13.,  42.,  43.,  61.,  61., 101.,
         70., 109.,  97., 140.,  87.,  96.,  42.,  18.,  14.]),
 array([0.03235259, 0.07958738, 0.12682216, 0.17405695, 0.22129173,
        0.26852652, 0.3157613 , 0.36299609, 0.41023088, 0.45746566,
        0.50470045, 0.55193523, 0.59917002, 0.6464048 , 0.69363959,
        0.74087437, 0.78810916, 0.83534394, 0.88257873, 0.92981351,
        0.9770483 ]),
 <a list of 20 Patch objects>)
```



Выборочное среднее и выборочная дисперсия

Ранее мы писали функции для их нахождения, воспользуемся ими же.

In [350]:
```
print(MEAN(rateTrig), VAR(rateTrig))
```

```
0.6172202980133177 0.02900346983064208
```

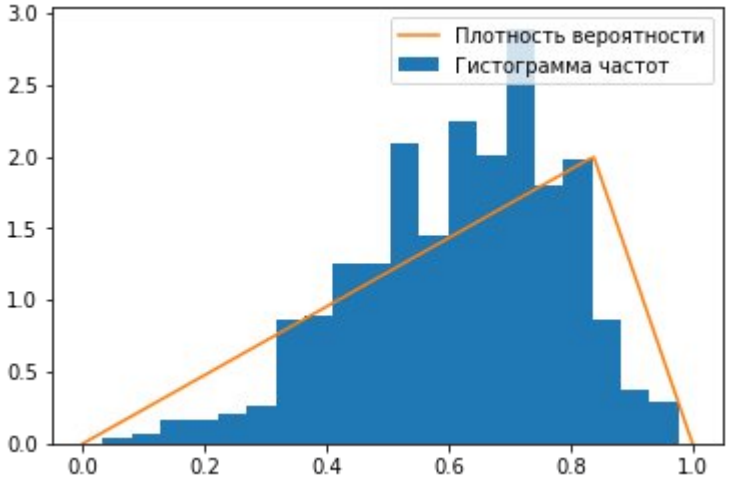
Получим значение предложенной ранее оценки

In [346]:
```
p_trig = MEAN(rateTrig)*3 - 1
print(p_trig)
```

```
0.837835781893661
```

Теперь же мы можем получить функцию плотности распределения. Изобразим её графически и с равним с гистограммой частот

In [351]:
```
plt.hist(rateTrig, bins = 20, density=True, label='Гистограмма частот')
x = np.linspace(sts.triang.ppf(0, p_trig), sts.triang.ppf(1, p_trig), 100)
plt.plot(x, sts.triang.pdf(x, p_trig), label='Плотность вероятности')
plt.legend(loc='upper right')
plt.show()
```



В результате получен относительно неожиданный результат. Значения максимума частоты гистограммы и максимума потности вероятности не совпадают. Это вызвано тем, что набор данных не точно соответствует треугольному распределению. Тем не менее, "общая картина" несколько приближена к полученному закону распределения.

Изучение медицинских данных важно. Например, зная подобные распределения, можно с высокой точностью предсказывать заболевания у потенциальных пациентов.

Распределение Ципфа

В примере 3.2 говорилось, что если упорядочить все слова языка по возрастанию, то распределение частот их использования будет близко к закону Ципфа.

Предположим, что пароли, придумываемые пользователями Интернета тоже подчиняются закону Ципфа. Поводы так считать имеются – очень часто люди в качестве паролей используют не псевдослучайно сгенерированные комбинации символов, а слова из жизни, либо же их какое-то сочетание или изменённую их версию. Раз за основу взят язык, на котором люди разговаривают, то производные "слова" (в данном случае – пароли) вполне могут следовать этому же закону.

Для анализа данных будем использовать пароли пользователей SkTorrent, список которых доступен на: https://github.com/duyet/bruteforce-database/find/master (https://github.com/duyet/bruteforce-database/find/master)

Для начала прочитаем файл, занесём в список пароли.

In [380]:
```
f = open('38650-password-sktorrent.txt', 'r', errors="ignore")
str = f.read()
pwlist = str.split('\n')
```

Так как база данных не велика, при анализе будем учитывать только первые 6 символов паролей, чтобы появлялось больше совпадений. Создадим список, в котором будут храниться первые 6 символов каждого из паролей.

Так же для наглядности на графике будем изображать только первые 20 самых часто встречаемых комбинаций символов. Так как мы "обрезали хвост" у гистограммы, следует учитывать, что функция вероятности закона Ципфа стремится к нулю, что в нашем случае невозможно, так как 20-ая по встречаемости комбинация символов всё ещё имеет ненулевую частоту. Это можно "исправить", отняв от каждой частоты частоту 21-ого по встречаемости слова.

In [386]:
```
pwl = []
l = []

import collections

for pw in pwlist:
        pwl.append(pw[:6])

counter=collections.Counter(pwl)

for i in range(20):
    l += [i+1]*(counter.most_common(20)[i][1]-counter.most_common(21)[20][1])
```

Выборочное среднее и выборочная дисперсия

In [387]:
```
print(MEAN(list), VAR(list))
```

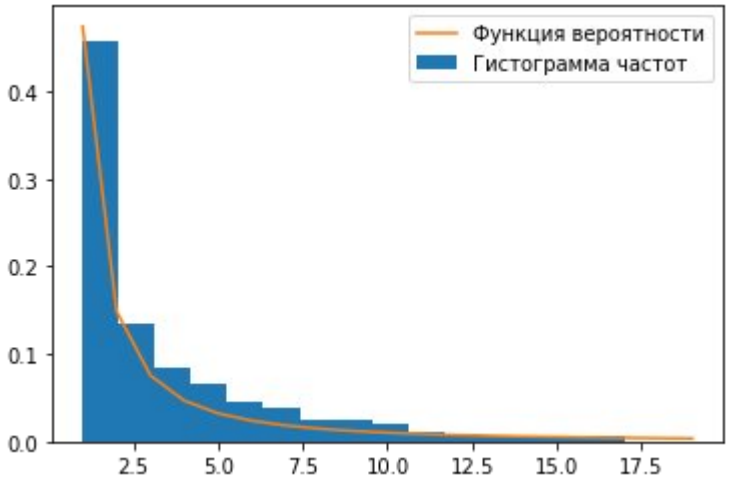
```
3.6006389776357826 9.888434096499925
```

Поиск оцениваемого параметра

In [388]:
```
aim = 0
for el in l:
    aim += np.log(el)
aim = aim/len(l)

p_zipf = getTheta(aim)
print(getTheta(aim))
```

```
1.6729736328125
```

In [389]:
```
plt.hist(l, bins = 15, density=True, label='Гистограмма частот')
plt.plot(range(1, 20), sts.zipf.pmf(range(1, 20), p_zipf), label='Функция вер
оятности')
plt.legend(loc='upper right')
plt.show()
```



Как оказалось, график весьма точно соответствует описанной нами модели. Это поддерживает предположение, что распределение паролей пользователей можно описать законом Ципфа.

Изучив эти данные, можно повысить информационную безопасность организации или сайта, изучив, какие пароли и их составные части чаще всего используются, и предупредив их использование внутри компании.

В частности, наиболее частыми сочетаниями символов в начале паролей оказались *martin*, *michal*, 123456, *domini*, *daniel* – имена и цифры в ряд. Ниже выведен список 20 самых частых паролей из изученной базы данных.

In [196]:
```
counter.most_common(20)
```

Out[196]:
```
[('martin', 125),
 ('michal', 68),
 ('123456', 65),
 ('domini', 48),
 ('daniel', 42),
 ('torren', 35),
 ('patrik', 33),
 ('marian', 28),
 ('nikola', 28),
 ('sktorr', 27),
 ('Martin', 24),
 ('veroni', 23),
 ('kristi', 22),
 ('andrej', 22),
 ('monika', 21),
 ('samsun', 21),
 ('159753', 21),
 ('tomas1', 20),
 ('sparta', 20),
 ('robert', 20)]
```

Домашнее задание 4

4.1. Задание для рассматриваемых распределений

Геометрическое распределение

Критерий согласия Колмогорова (Смирнова)

$$D_n = D_n(X) = sup\,|\hat{F}_n(x) - F_n(x)\,|$$

Собственно, критерий заключается в проверке неравенства $\sqrt{n}D_n < \lambda_\alpha$. Мы можем однозначно определить λ_α из выражения $1 - K(\lambda_\alpha) = \alpha$, где $K(\lambda_\alpha)$ – статистика Колмогорова.

$$K(x) = \sum_{i=-\infty}^{\infty} (-1)^i exp(-2i^2x^2)$$

В качестве уровня значимости обычно берут $\alpha = 0.05$, ему соответствует табличное значение $\lambda_\alpha = 1.36$.

Теперь займёмся проверкой. Так как за выборкой второго домашнего задания ходить уже далековато, создадим новую:

In [2]:
```
from random import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
from math import sqrt
```

In [3]:
```
%matplotlib inline

def GeomCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    s = 0
    i = 1
    q = 1 - p
    n = random()
    while s <= n:
        s += p * q**(i-1)
        i += 1
    return i-1

volumes = (5, 10, 100, 1000, 100000)
SampleGeom = [[[GeomCustom(0.2) for i in range(N)] for i in range(5)] for N i
n volumes]

print(*SampleGeom[0],'\n', sep='\n')
print(*SampleGeom[1], sep='\n')
```

```
[6, 3, 3, 2, 3]
[2, 11, 1, 1, 13]
[3, 1, 1, 2, 5]
[1, 1, 15, 6, 5]
[4, 1, 2, 3, 5]


[9, 3, 7, 7, 1, 4, 4, 11, 2, 3]
[1, 8, 7, 2, 3, 6, 1, 28, 17, 2]
[6, 3, 4, 4, 8, 2, 10, 1, 5, 3]
[9, 4, 3, 7, 3, 2, 2, 3, 2, 3]
[2, 13, 12, 2, 9, 5, 5, 5, 2, 7]
```

Искать будем следующем образом. Определим функции:

$$D_n^+ = max|\frac{k}{n} - F(x_{(k)})|$$

$$D_n^- = max|F(x_{(k)}) - \frac{k-1}{n}|$$

Тогда $D_n = max(D_n^+, D_n^-)$

Осталось вспомнить, что критерий Колмогорова не работает с "разрывными" случайными величинами, поэтому применим классическое преобразование к непрерывному равномерному на отрезке $[0, 1]$.

$$U_i = F(X_i -) + Y_i[F(X_i - F(X_i -))], Y_i \to R[0, 1]$$

Заполнение каждого из "кусочков" будем осуществлять с помощью заполнения соответствующих участков равномерно распределёнными случайными числами. Создадим выборку раверно распределённой случайной величины, куда будем заносить значения из равномерного распределения $R[x_{i-1}, x_i]$.

Применяем данный алгоритм

In [4]:
```
# Преобразование к равномерно распределённой случайной величине

SampleU = [[[1 for i in range(N)] for i in range(5)] for N in volumes]
rv = sts.geom(0.2)

for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        for k in range(len(SampleGeom[i][j])):
            SampleU[i][j][k] = rv.cdf(SampleGeom[i][j][k]-1) + random()*(
            rv.cdf(SampleGeom[i][j][k])-rv.cdf(SampleGeom[i][j][k]-1))

SampleU[1][1] # Покажем одну из получившихся выборок
```

Out[4]:
```
[0.05451811379396391,
 0.8043364890028112,
 0.7606713133111433,
 0.21482876978418738,
 0.3816873777756956,
 0.702127242068725,
 0.04053063577047534,
 0.9978254424446127,
 0.9725411414734756,
 0.26620206514344846]
```

In [5]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):

        SampleU[i][j] = sorted(SampleU[i][j])
        D_plus_list = []
        D_minus_list = []
        rv_u = sts.uniform()

        n = len(SampleU[i][j])
        for k in range(n):
                D_plus_list.append(abs((k+1)/n - SampleU[i][j][k]))
                D_minus_list.append(abs((k)/n - SampleU[i][j][k]))
        D_minus = max(D_minus_list)
        D_plus = max(D_minus_list)
        D = max(D_minus, D_plus)
        print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  5 ; D =  0.3209730543992567 ; sqrt(n)*D =  0.7177175685824759
n =  5 ; D =  0.2974088315195841 ; sqrt(n)*D =  0.6650263643865721
n =  5 ; D =  0.17123382699678777 ; sqrt(n)*D =  0.3828904772122561
n =  5 ; D =  0.22767142711221966 ; sqrt(n)*D =  0.5090887875573118
n =  5 ; D =  0.16726547966365657 ; sqrt(n)*D =  0.37401698281704476
n =  10 ; D =  0.1915137766571639 ; sqrt(n)*D =  0.6056197375374259
n =  10 ; D =  0.202127242068725 ; sqrt(n)*D =  0.6391824621053753
n =  10 ; D =  0.22824335374455032 ; sqrt(n)*D =  0.7217688586283003
n =  10 ; D =  0.25683973290084394 ; sqrt(n)*D =  0.8121985495959524
n =  10 ; D =  0.32015265869070203 ; sqrt(n)*D =  1.012411600421119
n =  100 ; D =  0.08988392113140332 ; sqrt(n)*D =  0.8988392113140332
n =  100 ; D =  0.09225430627045816 ; sqrt(n)*D =  0.9225430627045816
n =  100 ; D =  0.15006178329525474 ; sqrt(n)*D =  1.5006178329525475
n =  100 ; D =  0.13111189685386399 ; sqrt(n)*D =  1.31111896853864
n =  100 ; D =  0.04452707019258597 ; sqrt(n)*D =  0.4452707019258597
n =  1000 ; D =  0.022405017596864474 ; sqrt(n)*D =  0.7085088662224395
n =  1000 ; D =  0.02207926918959835 ; sqrt(n)*D =  0.6982077971111086
n =  1000 ; D =  0.03594549870466329 ; sqrt(n)*D =  1.1366964753736815
n =  1000 ; D =  0.019495381631168374 ; sqrt(n)*D =  0.6164980980870073
n =  1000 ; D =  0.014295265872251717 ; sqrt(n)*D =  0.4520559991398904
n =  100000 ; D =  0.002462861800952232 ; sqrt(n)*D =  0.7788252853233305
n =  100000 ; D =  0.0031953310631619436 ; sqrt(n)*D =  1.0104524037879092
n =  100000 ; D =  0.0018349353087633302 ; sqrt(n)*D =  0.5802574934756447
n =  100000 ; D =  0.002989755565449559 ; sqrt(n)*D =  0.9454437233985222
n =  100000 ; D =  0.0026706660047115527 ; sqrt(n)*D =  0.8445387444470484
```

Как видно по результатам, гипотеза о виде распределения не противоречит критерию согласия Колмогорова ни для одной из выборок (нет ни одного результата $sqrt(n) * D$, большего 1.36).

Заметим, что при малых объёмах выборки данный критерий использовать неправильно, но тем не менее, он тоже сработал.

Теперь перейдём к случаю сложной гипотезы.

Построение оценки неизвестного параметра и вычисление его значения выполнено в предыдущем домашнем задании. В случае сложной гипотезы Колмогорова оценка параметра должна быть получена методом максимального правдоподобия.

Использования метода рассматривалось в предыдущей части работы. Вспомним его для геометрического распределения:

$$L(x;\theta) = \theta^n \cdot (1-\theta)^{\sum_i^n x_i - n}$$

$$(lnL(x;\theta))' = \frac{n}{\theta} - \frac{\sum_i^n x_i - n}{1-\theta} = 0$$

$$\theta = \frac{n}{\sum_i^n x_i}$$

Значение параметра стоит вычислить для новых реализаций выборок.

In [6]:
```
def MEAN(sample):
    return sum(sample)/len(sample)
```

In [7]:
```
for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):
        print(1/MEAN(SampleGeom[i][j]), end = '\n' if j == len(SampleGeom[i])
-1 else ' ')
```

```
0.29411764705882354 0.17857142857142858 0.4166666666666667 0.1785714285714285
8 0.3333333333333333
0.19607843137254904 0.13333333333333333 0.2173913043478261 0.2631578947368421
0.16129032258064516
0.1763668430335097 0.2173913043478261 0.2237136465324385 0.17699115044247787
0.20202020202020202
0.1984520738241715 0.19116803670426305 0.21101498206372654 0.1968503937007873
8 0.20177562550443906
0.20011046097445792 0.20003520619629056 0.20016573723042677 0.200022802599496
34 0.19920040955604207
```

К сожалению, для геометрического распределения верхние процентные точки подсчитать не удалось, и в литературе их не удалось обнаружить. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём одной значение оценки неизвестного параметра, а по другим выборкам такого же объёма проверим гипотезу о виде распределения.

In [61]:
```
SampleU_1000 = [[0 for i in range(1000)] for i in range(4)]
SampleU_100000 = [[0 for i in range(100000)] for i in range(4)]

rv_1000 = sts.geom(0.20177562550443906)
rv_100000 = sts.geom(0.19920040955604207)

for i in range(4):
    for k in range(len(SampleGeom[3][i])):
        SampleU_1000[i][k] = rv_1000.cdf(SampleGeom[3][i][k]-1
            ) + random()*(rv_1000.cdf(SampleGeom[3][i][k])-rv_1000.cdf(
                SampleGeom[3][i][k]-1))

for i in range(4):
    for k in range(len(SampleGeom[4][i])):
        SampleU_100000[i][k] = rv_100000.cdf(SampleGeom[4][i][k]-1
            ) + random()*(rv_100000.cdf(SampleGeom[4][i][k]
                )-rv_100000.cdf(SampleGeom[4][i][k]-1))

SampleU_100000[3][0:9] # Выведем часть выборки чисто из интереса
```

Out[61]:
```
[0.6785600217376309,
 0.2705904325546926,
 0.2764824068002055,
 0.04329644194450607,
 0.5315139174346986,
 0.6358857115611911,
 0.9394364626091856,
 0.29300124137751876,
 0.343318472522626]
```

In [63]:
```
rv_u = sts.uniform()

for i in range(4):
    SampleU_1000[i] = sorted(SampleU_1000[i])
    D_plus_list = []
    D_minus_list = []
    n = len(SampleU_1000[i])
    for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU_1000[i][k]))
        D_minus_list.append(abs((k)/n - SampleU_1000[i][k]))
    D_minus = max(D_minus_list)
    D_plus = max(D_minus_list)
    D = max(D_minus, D_plus)
    print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))

for i in range(4):
    SampleU_100000[i] = sorted(SampleU_100000[i])
    D_plus_list = []
    D_minus_list = []
    n = len(SampleU_100000[i])
    for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU_100000[i][k]))
        D_minus_list.append(abs((k)/n - SampleU_100000[i][k]))
    D_minus = max(D_minus_list)
    D_plus = max(D_minus_list)
    D = max(D_minus, D_plus)
    print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  1000 ; D =  0.024766641648585663 ; sqrt(n)*D =  0.783189976027182
n =  1000 ; D =  0.02768631143429845 ; sqrt(n)*D =  0.8755180414114635
n =  1000 ; D =  0.030956996680126386 ; sqrt(n)*D =  0.9789461902747035
n =  1000 ; D =  0.024736086119779233 ; sqrt(n)*D =  0.78222372536579
n =  100000 ; D =  0.002429963806082014 ; sqrt(n)*D =  0.7684220258990881
n =  100000 ; D =  0.005568079726402497 ; sqrt(n)*D =  1.760781412883908
n =  100000 ; D =  0.003058670601812974 ; sqrt(n)*D =  0.967236571392694
n =  100000 ; D =  0.002147227466918644 ; sqrt(n)*D =  0.6790129449936767
```

Для одной из выборок мы вынуждены отвергнуть гипотезу – для второй выборки объёма 100 000 полученное значение превосходит границу 1.36. Для остальных критерий не отвергается.

Критерий согласия хи-квадрат

Геометрическое распределение относится к бесконечным дискретным (случайная величина может принимать бесконечное количество значений). Поэтому для лпределения числа интервалов воспользуемся эмпирическим правилом Стёрджеса. Тогда количество интервалов для провекри критерия должно быть равно:

$N = 1 + [\log_2 n]$

В таком случае воспользуемся следующей группировкой интервалов: вероятности первых $[\log_2 n]$ значений и вероятность того, что случайная величина примет одно из оставшихся значений.

Далее определяем статистику хи-квадрат:

$X_N^2 = \sum_{i=1}^{N} \frac{(v_i^{(n)} - np_i)^2}{np_i} = \frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$, где w_i – отклонение от ожидаемой частоты встречи определённого элемента.

Алгоритм для подсчёта статистики предлагается следующий:

а) Найти частоты встречи каждого значения элемента выборки v_i (с учётом определённого количества интервалов)

б) Вычислить отклонения от соответстсвующей теоретической частоты встречи значения как $w_i = p_i \cdot n - v_i$

в) Рассчитать сумму $\frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$

Реализуем этот алгоритм:

In [64]:
```
rv = sts.geom(0.2)

for i in range(len(SampleGeom)):
    for j in range(len(SampleGeom[i])):

        n = len(SampleGeom[i][j])
        N = 1 + int(np.log2(n))
        frequency = []
        for k in range(1, N):
            frequency.append(SampleGeom[i][j].count(k))
        frequency.append(n - sum(frequency))

        deviation = []
        for k in range(N-1):
            deviation.append(frequency[k] - n*rv.pmf(k+1))
        deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

        s = 0
        for k in range(N-1):
            s += deviation[k]**2 / (n*rv.pmf(k+1))
        s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
        print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.2500000000000002
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.4999999999999996
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.4999999999999996
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.8125000000000002
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.062499999999999827
n =  10 ; N =  4 ; Статистика хи-квдрат =  1.2812499999999998
n =  10 ; N =  4 ; Статистика хи-квдрат =  0.16406249999999992
n =  10 ; N =  4 ; Статистика хи-квдрат =  1.2812499999999998
n =  10 ; N =  4 ; Статистика хи-квдрат =  9.882812499999996
n =  10 ; N =  4 ; Статистика хи-квдрат =  5.195312499999999
n =  100 ; N =  7 ; Статистика хи-квдрат =  14.714709472656251
n =  100 ; N =  7 ; Статистика хи-квдрат =  5.925793457031244
n =  100 ; N =  7 ; Статистика хи-квдрат =  15.968017578125005
n =  100 ; N =  7 ; Статистика хи-квдрат =  11.806408691406258
n =  100 ; N =  7 ; Статистика хи-квдрат =  2.675830078124999
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.778107852935805
n =  1000 ; N =  10 ; Статистика хи-квдрат =  6.779962844848637
n =  1000 ; N =  10 ; Статистика хи-квдрат =  11.632664585113519
n =  1000 ; N =  10 ; Статистика хи-квдрат =  8.278559589385988
n =  1000 ; N =  10 ; Статистика хи-квдрат =  4.472120666503901
n =  100000 ; N =  17 ; Статистика хи-квдрат =  9.880146554819786
n =  100000 ; N =  17 ; Статистика хи-квдрат =  18.941007666966343
n =  100000 ; N =  17 ; Статистика хи-квдрат =  9.844903687190595
n =  100000 ; N =  17 ; Статистика хи-квдрат =  18.7994364888529
n =  100000 ; N =  17 ; Статистика хи-квдрат =  20.707288278430056
```

Возьмём тот же уровень значимости $\alpha = 0.05$. Число степеней свободы равно $N - 1$. Для выборки объёма 5 ему соответствует табличное значение 6.0 – гипотеза не отвергается ни дня одной из выборок. Для выборки объёма 10 – значение 7.8 – гипотеза отклоняется для четвёртой выборки. Для объёма 100 – значение 12.6 – отклоняем гипотезу для первой и третьей выборки. Для объёма 1000 – значение 16.9 – гипотеза подтверждается для всех выборок. Для объёма 100000 – значение 26.3 – гипотеза подтверждается для всех выборок.

Как видно, критерий хи-квадрат дал "сбой" только для выборок с малым объёмом. Для больших объёмов данных критерий сработал без осечек.

Случай сложной гипотезы.

Для рассматриваемого распределения предельные значения предельных статистик неизвестны. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём из одной значение оценки неизвестного параметра, а к оставшимся 4 выборкам такого же объёма проверим гипотезу о виде распределения.

In [210]:

```
rv = sts.geom(0.20794343938448742)

for i in range(4):
    n = len(SampleGeom[3][i])
    N = 1 + int(np.log2(n))
    frequency = []
    for k in range(1, N):
        frequency.append(SampleGeom[3][i].count(k))
    frequency.append(n - sum(frequency))

    deviation = []
    for k in range(N-1):
        deviation.append(frequency[k] - n*rv.pmf(k+1))
    deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

    s = 0
    for k in range(N-1):
        s += deviation[k]**2 / (n*rv.pmf(k+1))
    s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
    print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)

for i in range(4):
    rv = sts.geom(0.20028119479749568)

    n = len(SampleGeom[4][i])
    N = 1 + int(np.log2(n))
    frequency = []
    for k in range(1, N):
        frequency.append(SampleGeom[4][i].count(k))
    frequency.append(n - sum(frequency))

    deviation = []
    for k in range(N-1):
        deviation.append(frequency[k] - n*rv.pmf(k+1))
    deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

    s = 0
    for k in range(N-1):
        s += deviation[k]**2 / (n*rv.pmf(k+1))
    s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
    print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  1000 ; N =  10 ; Статистика хи-квдрат =  10.606786243381553
n =  1000 ; N =  10 ; Статистика хи-квдрат =  11.132523191101633
n =  1000 ; N =  10 ; Статистика хи-квдрат =  6.876480323435954
n =  1000 ; N =  10 ; Статистика хи-квдрат =  11.515559470878243
n =  100000 ; N =  17 ; Статистика хи-квдрат =  10.204317711128962
n =  100000 ; N =  17 ; Статистика хи-квдрат =  18.738025928732668
n =  100000 ; N =  17 ; Статистика хи-квдрат =  9.460893398951272
n =  100000 ; N =  17 ; Статистика хи-квдрат =  19.064811460686894
```

В случае сложной гипотезы считаем, что количество степеней свободы равно $N - 2$. Из-за параметра распределения, по выборке которого мы сделали оценку и из-за параметра распределения, статистику которого мы считали.

Для объёма 1000 – допустимая граница 15.5. Для объёма 100000 – допустимая граница 25.0. Гипотезу можно считать подтверждённой, так как полученные значения лежат в допустимых границах в соответстсвии с табличными значениями (найдены ранее).

Треугольное распределение

Критерий согласия Колмогорова (Смирнова)

$$D_n = D_n(X) = sup\,|\hat{F}_n(x) - F_n(x)|$$

Собственно, критерий заключается в проверке неравенства $\sqrt{n}D_n < \lambda_\alpha$. Мы можем однозначно определить λ_α из выражения $1 - K(\lambda_\alpha) = \alpha$, где $K(\lambda_\alpha)$ – статистика Колмогорова.

$$K(x) = \sum_{i=-\infty}^{\infty} (-1)^i exp(-2i^2x^2)$$

В качестве уровня значимости обычно берут $\alpha = 0.05$, ему соответствует табличное значение $\lambda_\alpha = 1.36$.

Теперь займёмся проверкой. Так как за выборкой второго домашнего задания ходить уже далековато, создадим новую:

In [67]:
```
def TrigCustom(p):
    if (p <= 0 or p >= 1):
        return 0 #Проверка на корректность ввода параметра
    n = random()
    if n < p:
        return sqrt(p * n)
    else:
        return 1 - sqrt((1-p)*(1-n))

volumes = (5, 10, 100, 1000, 100000)
SampleTrig = [[[TrigCustom(0.2) for i in range(N)] for i in range(5)] for N i
n volumes]

for i in range(5):
    for j in range(5):
        print("%.2f" % SampleTrig[0][i][j], end = '\n' if j == 4 else ' ')

print()

for i in range(5):
    for j in range(10):
        print("%.2f" % SampleTrig[1][i][j], end = '\n' if j == 9 else ' ')
```

```
0.31 0.49 0.41 0.12 0.69
0.63 0.66 0.14 0.72 0.37
0.17 0.23 0.29 0.15 0.52
0.59 0.22 0.74 0.54 0.32
0.34 0.23 0.73 0.48 0.17

0.28 0.35 0.04 0.10 0.37 0.87 0.39 0.14 0.33 0.31
0.27 0.22 0.14 0.76 0.77 0.21 0.16 0.46 0.20 0.16
0.14 0.30 0.79 0.41 0.61 0.50 0.11 0.44 0.74 0.47
0.37 0.30 0.26 0.53 0.19 0.66 0.39 0.18 0.25 0.14
0.04 0.63 0.17 0.27 0.27 0.18 0.11 0.20 0.41 0.55
```

Искать будем следующем образом. Определим функции:

$$D_n^+ = max|\frac{k}{n} - F(x_{(k)})|$$

$$D_n^- = max|F(x_{(k)}) - \frac{k-1}{n}|$$

Тогда $D_n = max(D_n^+, D_n^-)$

В этот раз нам повезло больше. Функция распределения непрерывна, а значит, в преобразовании к равномерному распределению нужды нет.

Применяем данный алгоритм

In [69]:
```
rv = sts.triang(0.2)

for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):

        SampleTrig[i][j] = sorted(SampleTrig[i][j])
        D_plus_list = []
        D_minus_list = []

        n = len(SampleTrig[i][j])
        for k in range(1,n):
                D_plus_list.append(abs((k+1)/n - rv.cdf(
                    SampleTrig[i][j][k])))
                D_minus_list.append(abs((k)/n - rv.cdf(
                    SampleTrig[i][j][k])))
        D_minus = max(D_minus_list)
        D_plus = max(D_minus_list)
        D = max(D_minus, D_plus)
        print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  5 ; D =  0.19994443982927246 ; sqrt(n)*D =  0.44708935918136966
n =  5 ; D =  0.4284461505566254 ; sqrt(n)*D =  0.9580347173427237
n =  5 ; D =  0.2275699559341795 ; sqrt(n)*D =  0.5088618911054571
n =  5 ; D =  0.33353330481947197 ; sqrt(n)*D =  0.7458031423364976
n =  5 ; D =  0.10658072832670851 ; sqrt(n)*D =  0.23832175362995764
n =  10 ; D =  0.2625226715119374 ; sqrt(n)*D =  0.8301695794099215
n =  10 ; D =  0.26601273343250154 ; sqrt(n)*D =  0.8412061242539258
n =  10 ; D =  0.27044070680382354 ; sqrt(n)*D =  0.8552086055258779
n =  10 ; D =  0.1647761966163266 ; sqrt(n)*D =  0.5210680854873221
n =  10 ; D =  0.26564278783898404 ; sqrt(n)*D =  0.8400362535680677
n =  100 ; D =  0.11416733627141457 ; sqrt(n)*D =  1.1416733627141458
n =  100 ; D =  0.08451801887556348 ; sqrt(n)*D =  0.8451801887556348
n =  100 ; D =  0.06328859571880208 ; sqrt(n)*D =  0.6328859571880208
n =  100 ; D =  0.13350213727078944 ; sqrt(n)*D =  1.3350213727078943
n =  100 ; D =  0.09547841928645859 ; sqrt(n)*D =  0.9547841928645859
n =  1000 ; D =  0.025930839844792775 ; sqrt(n)*D =  0.8200051555059228
n =  1000 ; D =  0.027648034659723653 ; sqrt(n)*D =  0.8743076235200516
n =  1000 ; D =  0.04248270590498776 ; sqrt(n)*D =  1.3434211182684608
n =  1000 ; D =  0.015610691072172694 ; sqrt(n)*D =  0.49365339637321676
n =  1000 ; D =  0.014458164863551226 ; sqrt(n)*D =  0.45720731755039445
n =  100000 ; D =  0.002144755546745203 ; sqrt(n)*D =  0.6782312551994574
n =  100000 ; D =  0.0018727617099294491 ; sqrt(n)*D =  0.5922192518128632
n =  100000 ; D =  0.002374780521606501 ; sqrt(n)*D =  0.750971539127925
n =  100000 ; D =  0.004072319720211659 ; sqrt(n)*D =  1.2877805676288476
n =  100000 ; D =  0.0016991333501413797 ; sqrt(n)*D =  0.5373131434799142
```

Как видно по результатам, ни один из результатов не опровергается критерием согласия Колмогорова (нет ни одного результата $sqrt(n) * D$, большего 1.36).

Заметим, что при малых объёмах выборки данный критерий использовать неправильно, но тем не менее, он тоже сработал. Более точная работа с выборками малого объёма относится к дополнительным заданиям и будет выполнена позже.

Теперь перейдём к случаю сложной гипотезы.

Построение оценки неизвестного параметра и вычисление его значения выполнено в предыдущем домашнем задании.

Выведем оценку методом максимального правдоподобия. Получим её с помощью численного приближения. Определим плотность треугольного распределения, затем логарифм функции правдоподобия. Затем с точностью в 0.01 численно найдём такое θ, при котором функция логарифма имеет максимальное значение.

In [207]:
```
def cdfTrig(theta):
    def f(x):
        if (x > 0 and x < theta):
            return 2*x/theta
        elif (x >= theta and x < 1):
            return(2*(1-x)/(1-theta))
        else:
            return 0
    return f

def likehoodTrigLog(theta, sample):
    p = 0
    for i in range(len(sample)):
        p += np.log(cdfTrig(theta)(sample[i]))
    return p

def getMax(sample):
    n = 100
    m = 0
    c = 0
    for i in range(n):
        if likehoodTrig((i/n), sample) > m:
            m = likehoodTrig((i/n), sample)
            c = i
    return c/n
```

In [208]:
```
for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):
        print(getMax(SampleTrig[i][j]),
              end = '\n' if j == len(SampleTrig[i])-1 else ' ')
```

```
0.31 0.65 0.15 0.54 0.17
0.04 0.16 0.43 0.18 0.04
0.05 0.21 0.23 0.32 0.27
0.19 0.18 0.19 0.2 0.2
0.2 0.2 0.2 0.2 0.2
```

В случае треугольного распределения верхние процентные точки подсчитать не удалось, и в литературе их не удалось обнаружить. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём у одной значение оценки неизвестного параметра, а от остальных выборок такого же объёма проверим гипотезу о виде распределения.

In [209]:
```
rv3 = sts.triang(0.2)
rv4 = sts.triang(0.2)

for i in range(4):
    SampleTrig[3][i] = sorted(SampleTrig[3][i])
    D_plus_list = []
    D_minus_list = []

    n = len(SampleTrig[3][0])
    for k in range(1,n):
        D_plus_list.append(abs((k+1)/n - rv.cdf(
            SampleTrig[3][i][k])))
        D_minus_list.append(abs((k)/n - rv.cdf(
            SampleTrig[3][i][k])))
    D_minus = max(D_minus_list)
    D_plus = max(D_minus_list)
    D = max(D_minus, D_plus)
    print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))

for i in range(4):
    SampleTrig[i][j] = sorted(SampleTrig[i][j])
    D_plus_list = []
    D_minus_list = []

    n = len(SampleTrig[4][0])
    for k in range(1,n):
        D_plus_list.append(abs((k+1)/n - rv.cdf(
            SampleTrig[4][i][k])))
        D_minus_list.append(abs((k)/n - rv.cdf(
            SampleTrig[4][i][k])))
    D_minus = max(D_minus_list)
    D_plus = max(D_minus_list)
    D = max(D_minus, D_plus)
    print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  1000 ; D =  0.025930839844792775 ; sqrt(n)*D =  0.8200051555059228
n =  1000 ; D =  0.027648034659723653 ; sqrt(n)*D =  0.8743076235200516
n =  1000 ; D =  0.04248270590498776 ; sqrt(n)*D =  1.3434211182684608
n =  1000 ; D =  0.015610691072172694 ; sqrt(n)*D =  0.49365339637321676
n =  100000 ; D =  0.002144755546745203 ; sqrt(n)*D =  0.6782312551994574
n =  100000 ; D =  0.0018727617099294491 ; sqrt(n)*D =  0.5922192518128632
n =  100000 ; D =  0.002374780521606501 ; sqrt(n)*D =  0.750971539127925
n =  100000 ; D =  0.004072319720211659 ; sqrt(n)*D =  1.2877805676288476
```

Полученные значения лежат в пределах, соответствующих уровню значимости 0.05. Данные каждой из выборок не противоречат гипотезе.

Критерий согласия хи-квадрат

Треугольное распределение относится к непрерывным распредениям. Поэтому для определения числа интервалов воспользуемся эмпирическим правилом Стёрджеса. Тогда оптимальное количество интервалов для провекри критерия должно быть равно:

$N = 1 + [\log_2 n]$

В таком случае воспользуемся следующей группировкой интервалов: разделим промежуток $[0; 1]$ на $[\log_2 n] + 1$ равных частей. Соответствующую определённому интервалу $[a_i; x_{a+1}]$ вероятность можно легко посчитать как $p_i = F(a_{i+1}) - F(a_i)$.

Далее определяем статистику хи-квадрат:

$X_N^2 = \sum_{i=1}^{N} \frac{(v_i^{(n)} - np_i)^2}{np_i} = \frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$, где w_i – отклонение от ожидаемой частоты встречи определённого элемента.

Алгоритм для подсчёта статистики предлагается следующий:

а) Найти частоты встречи каждого множества значений элемента выборки v_i (с учётом определённого количества интервалов).

б) Вычислить отклонения от соответстсвующей теоретической частоты встречи множества значений как $w_i = p_i \cdot n - v_i$

в) Рассчитать сумму $\frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$

Реализуем этот алгоритм:

In [211]:

```
rv = sts.triang(0.2)

for i in range(len(SampleTrig)):
    for j in range(len(SampleTrig[i])):

        n = len(SampleTrig[i][j])
        N = 1 + int(np.log2(n))
        frequency = [0]*N # Массив с частотами
        for element in SampleTrig[i][j]:
            frequency[int(element*N)] += 1 # При встрече элемента - инкремент
соответствующей частоты

        deviation = []
        for k in range(N):
            deviation.append(frequency[k] - n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))

        s = 0
        for k in range(N-1):
            s += deviation[k]**2 / (n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))
        print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.02555555555555555
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.0755555555555558
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.985555555555555
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.02555555555555555
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.02555555555555555
n =  10 ; N =  4 ; Статистика хи-квдрат =  3.466328947368421
n =  10 ; N =  4 ; Статистика хи-квдрат =  6.369065789473686
n =  10 ; N =  4 ; Статистика хи-квдрат =  0.6727850877192977
n =  10 ; N =  4 ; Статистика хи-квдрат =  0.35699561403508745
n =  10 ; N =  4 ; Статистика хи-квдрат =  1.650469298245615
n =  100 ; N =  7 ; Статистика хи-квдрат =  8.315939068960933
n =  100 ; N =  7 ; Статистика хи-квдрат =  1.7777978924903384
n =  100 ; N =  7 ; Статистика хи-квдрат =  2.4047756702680987
n =  100 ; N =  7 ; Статистика хи-квдрат =  11.852966519941294
n =  100 ; N =  7 ; Статистика хи-квдрат =  5.648605735627565
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.618570762570765
n =  1000 ; N =  10 ; Статистика хи-квдрат =  5.432075036075069
n =  1000 ; N =  10 ; Статистика хи-квдрат =  13.014091686091714
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.775662115662163
n =  1000 ; N =  10 ; Статистика хи-квдрат =  4.371858807858804
n =  100000 ; N =  17 ; Статистика хи-квдрат =  11.546321774482406
n =  100000 ; N =  17 ; Статистика хи-квдрат =  9.19713844347544
n =  100000 ; N =  17 ; Статистика хи-квдрат =  13.94390699322816
n =  100000 ; N =  17 ; Статистика хи-квдрат =  30.66824845072764
n =  100000 ; N =  17 ; Статистика хи-квдрат =  13.325766979129025
```

Возьмём тот же уровень значимости $\alpha = 0.05$. Число степеней свободы равно $N - 1$. Для выборки объёма 5 ему соответствует табличное значение 6.0 – гипотеза не отвергается ни для одной из выборок. Для выборки объёма 10 – значение 7.8 – не отвергается ни для одной из выборок. Для объёма 100 – значение 12.6 – не отвергается ни для одной из выборок. Для объёма 1000 – значение 16.9 – не отвергается ни для одной из выборок. Для объёма 100000 – значение 26.3 – гипотеза отвергается для четвёртой выборки.

На удивление гипотезу пришлось отвергнуть для выборки с очень большим объёмом. Предположительно, так просто очень "повезло" с выборкой, так как статистика для остальных выборок такого объёма очень уверенно лежит в нужных границах.

Случай сложной гипотезы.

Для рассматриваемого распределения предельные значения предельных статистик неизвестны. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём из одной значение оценки неизвестного параметра, а к остальным выборкам такого же объёма проверим гипотезу о виде распределения.

In [213]:
```
for i in range(4):
    n = len(SampleTrig[3][i])
    N = 1 + int(np.log2(n))
    frequency = [0]*N
    for element in SampleTrig[3][i]:
        frequency[int(element*N)] += 1 # При встрече элемента - инкремент соо
тветствующей частоты

    deviation = []
    for k in range(N):
        deviation.append(frequency[k] - n*(rv3.cdf((k+1)/N)-rv3.cdf(k/N)))

    s = 0
    for k in range(N-1):
        s += deviation[k]**2 / (n*(rv3.cdf((k+1)/N)-rv3.cdf(k/N)))
    print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)

for i in range(4):
    n = len(SampleTrig[4][i])
    N = 1 + int(np.log2(n))
    frequency = [0]*N
    for element in SampleTrig[4][i]:
        frequency[int(element*N)] += 1 # При встрече элемента - инкремент соо
тветствующей частоты

    deviation = []
    for k in range(N):
        deviation.append(frequency[k] - n*(rv4.cdf((k+1)/N)-rv4.cdf(k/N)))

    s = 0
    for k in range(N-1):
        s += deviation[k]**2 / (n*(rv4.cdf((k+1)/N)-rv4.cdf(k/N)))
    print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.618570762570765
n =  1000 ; N =  10 ; Статистика хи-квдрат =  5.432075036075069
n =  1000 ; N =  10 ; Статистика хи-квдрат =  13.014091686091714
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.775662115662163
n =  100000 ; N =  17 ; Статистика хи-квдрат =  11.546321774482406
n =  100000 ; N =  17 ; Статистика хи-квдрат =  9.19713844347544
n =  100000 ; N =  17 ; Статистика хи-квдрат =  13.94390699322816
n =  100000 ; N =  17 ; Статистика хи-квдрат =  30.66824845072764
```

В случае сложной гипотезы считаем, что количество степеней свободы равно $N-2$. Из-за параметра распределения, по выборке которого мы сделали оценку и из-за параметра распределения, статистику которого мы считали.

Для объёма 1000 – допустимая граница 15.5. Для объёма 100000 – допустимая граница 25.0. Гипотеза отвергается только для одной из выборок объёма 100000.

Распределение Ципфа

Критерий согласия Колмогорова (Смирнова)

$D_n = D_n(X) = sup|\hat{F}_n(x) - F_n(x)|$

Собственно, критерий заключается в проверке неравенства $\sqrt{n}D_n < \lambda_\alpha$. Мы можем однозначно определить λ_α из выражения $1 - K(\lambda_\alpha) = \alpha$, где $K(\lambda_\alpha)$ – статистика Колмогорова.

$$K(x) = \sum_{i=-\infty}^{\infty} (-1)^i exp(-2i^2x^2)$$

В качестве уровня значимости обычно берут $\alpha = 0.05$, ему соответствует табличное значение $\lambda_\alpha = 1.36$.

Теперь займёмся проверкой. Так как за выборкой второго домашнего задания ходить уже далековато, создадим новую:

In [70]:
```
def ZipfCustom(s, N):
    if (N != (N//1) or N<0):
        return 0 #Проверка на корректность ввода параметра
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    sum = 0
    i = 1
    n = random()
    while sum <= n:
        i += 1
        sum += ((i-1)**(-s))/H
    return i-1

volumes = (5, 10, 100, 1000, 100000)
SampleZipf = [[[ZipfCustom(2, 500) for i in range(N)] for i in range(5)] for
N in volumes]

print(*SampleZipf[0],'\n', sep='\n')
print(*SampleZipf[1], sep='\n')
```

```
[1, 1, 4, 1, 1]
[2, 2, 12, 2, 11]
[1, 1, 1, 4, 1]
[3, 1, 1, 1, 4]
[1, 1, 4, 3, 31]


[1, 1, 1, 1, 308, 1, 1, 2, 6, 1]
[1, 4, 1, 1, 1, 1, 2, 1, 31, 5]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 13]
[6, 2, 1, 4, 73, 2, 8, 1, 1, 1]
[1, 1, 1, 1, 1, 2, 1, 1, 1, 2]
```

Искать будем следующем образом. Определим функции:

$$D_n^+ = max|\frac{k}{n} - F(x_{(k)})|$$

$$D_n^- = max|F(x_{(k)}) - \frac{k-1}{n}|$$

Тогда $D_n = max(D_n^+, D_n^-)$

Как и в случае геометрического распределения, учитываем, что критерий Колмогорова не работает с "разрывными" случайными величинами, поэтому применим классическое преобразование к непрерывному равномерному на отрезке $[0, 1]$.

$$U_i = F(X_i -) + Y_i[F(X_i - F(X_i -))], Y_i \to R[0, 1]$$

Заполнение каждого из "кусочков" будем осуществлять с помощью заполнения соответствующих участков равномерно распределёнными случайными числами.

Применяем данный алгоритм

In [31]:
```
# Преобразование к равномерно распределённой случайной величине

SampleU = [[[1 for i in range(N)] for i in range(5)] for N in volumes]
rv = sts.zipf(2)

for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):
        for k in range(len(SampleZipf[i][j])):
            SampleU[i][j][k] = rv.cdf(SampleZipf[i][j][k]-1) + random()*(
            rv.cdf(SampleZipf[i][j][k])-rv.cdf(SampleZipf[i][j][k]-1))

SampleU[1][1] # Покажем одну из получившихся выборок
```

Out[31]:
```
[0.09048471040067242,
 0.5716432006155451,
 0.9773291849064072,
 0.22847019278018957,
 0.7448207701177925,
 0.505512019226276,
 0.4589574519535783,
 0.770689954241188,
 0.9223681057038453,
 0.2891212996458868]
```

In [32]:
```
for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):

        SampleU[i][j] = sorted(SampleU[i][j])
        D_plus_list = []
        D_minus_list = []
        rv_u = sts.uniform()

        n = len(SampleU[i][j])
        for k in range(n):
                D_plus_list.append(abs((k+1)/n - SampleU[i][j][k]))
                D_minus_list.append(abs((k)/n - SampleU[i][j][k]))
        D_minus = max(D_minus_list)
        D_plus = max(D_minus_list)
        D = max(D_minus, D_plus)
        print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  5 ; D =  0.2454994191356823 ; sqrt(n)*D =  0.5489533896240983
n =  5 ; D =  0.41246936103587445 ; sqrt(n)*D =  0.9223095299121183
n =  5 ; D =  0.31249265443042995 ; sqrt(n)*D =  0.6987548177757922
n =  5 ; D =  0.5081505712908836 ; sqrt(n)*D =  1.1362592202117687
n =  5 ; D =  0.18620696363339873 ; sqrt(n)*D =  0.4163714285681108
n =  10 ; D =  0.23658034251270027 ; sqrt(n)*D =  0.7481327319628956
n =  10 ; D =  0.1589574519535783 ; sqrt(n)*D =  0.5026675992300892
n =  10 ; D =  0.27259486141868505 ; sqrt(n)*D =  0.862020640541003
n =  10 ; D =  0.13088390711740583 ; sqrt(n)*D =  0.4138912555529256
n =  10 ; D =  0.16071931314916543 ; sqrt(n)*D =  0.5082390935292119
n =  100 ; D =  0.08411234282120919 ; sqrt(n)*D =  0.8411234282120919
n =  100 ; D =  0.06805479442112616 ; sqrt(n)*D =  0.6805479442112616
n =  100 ; D =  0.09513078393939817 ; sqrt(n)*D =  0.9513078393939817
n =  100 ; D =  0.11768822197497925 ; sqrt(n)*D =  1.1768822197497926
n =  100 ; D =  0.05342780536395922 ; sqrt(n)*D =  0.5342780536395921
n =  1000 ; D =  0.024534777195477808 ; sqrt(n)*D =  0.7758577782246807
n =  1000 ; D =  0.029023746242701154 ; sqrt(n)*D =  0.9178114435768979
n =  1000 ; D =  0.027584534206885336 ; sqrt(n)*D =  0.8722995628858398
n =  1000 ; D =  0.016608630974091154 ; sqrt(n)*D =  0.5252110269534904
n =  1000 ; D =  0.01599987401364847 ; sqrt(n)*D =  0.5059604415886914
n =  100000 ; D =  0.0019518892520394848 ; sqrt(n)*D =  0.6172415776847231
n =  100000 ; D =  0.0028493514358455263 ; sqrt(n)*D =  0.9010440391543004
n =  100000 ; D =  0.0028723556720525734 ; sqrt(n)*D =  0.9083186173789786
n =  100000 ; D =  0.0029657116597812394 ; sqrt(n)*D =  0.9378403728227099
n =  100000 ; D =  0.0024993267727002194 ; sqrt(n)*D =  0.7903565218770637
```

Как видно по результатам, гипотеза о каждой из выборок не отвергается критерием согласия Колмогорова (нет ни одного результата $sqrt(n) * D$, большего 1.36).

Заметим, что при малых объёмах выборки данный критерий использовать неправильно, но тем не менее, он тоже сработал. Более точная работа с выборками малого объёма относится к дополнительным заданиям и будет выполнена позже.

Теперь перейдём к случаю сложной гипотезы.

Построение оценки неизвестного параметра и вычисление его значения выполнено в предыдущем домашнем задании. Однако значение параметра стоит вычислить вновь, так как у нас теперь новые реализации выборки. Для этого воспользуемся уже используемой ранее функцией.

В предудыщей части работы оценка параметра была построена с помощью функции правдоподобия.

In [39]:

```
def tau(theta):
    sumUp = 0
    sumDown = 0
    for k in range (2, 500):
        sumUp += np.log(k)/(k**theta)
    for k in range (1, 500):
        sumDown += k**(-theta)
    return sumUp/sumDown

def getTheta(aim, eps = 1e-3):
    x = 9.9999
    a0 = 0
    b0 = 10
    while (abs(b0 - x) < eps):
        a = a0
        b = b0
        a0 += 10
        b0 += 10
        while abs(b - a) > eps:
            x = (a + b) / 2.0
            fx = tau(x) - aim
            fa = tau(a) - aim
            if (fx < 0 and fa < 0) or (fx > 0 and fa > 0):
                a = x
            else:
                b = x
    return x

aims = []

for i in range(5):
    for j in range(5):
        aim = 0
        for el in SampleZipf[i][j]:
            aim += np.log(el)
        aim = aim/len(SampleZipf[i][j])
        aims.append(aim)

for i in range(25):
    print((getTheta(aims[i])), end = '\n' if i%5 == 4 else ' ')
```

```
1.9805908203125 2.0086669921875 1.9183349609375 1.5106201171875 2.73376464843
75
2.6922607421875 1.8414306640625 1.8511962890625 2.6263427734375 3.16833496093
75
1.9659423828125 2.1270751953125 1.8450927734375 1.9671630859375 1.94763183593
75
2.0098876953125 1.9879150390625 2.0025634765625 2.0489501953125 2.01477050781
25
1.9976806640625 2.0001220703125 1.9927978515625 1.9989013671875 1.99768066406
25
```

Для закона Ципфа верхние процентные точки подсчитать не удалось, и в литературе их не удалось обнаружить. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём одной значение оценки неизвестного параметра, а от другой выборке такого же объёма проверим гипотезу о виде распределения.

In [41]:
```
SampleU_1000 = []
SampleU_100000 = []

rv_1000 = sts.zipf(2.0147705078125)
rv_100000 = sts.zipf(1.9976806640625)

for k in range(len(SampleZipf[3][0])):
    SampleU_1000.append(rv_1000.cdf(SampleZipf[3][0][k]-1) + random()*(
            rv_1000.cdf(SampleZipf[3][0][k])-rv_1000.cdf(
                SampleZipf[3][0][k]-1)))

for k in range(len(SampleZipf[4][0])):
    SampleU_100000.append(rv_100000.cdf(SampleZipf[4][0][k]-1
             ) + random()*(rv_100000.cdf(SampleZipf[4][0][k]
                )-rv_100000.cdf(SampleZipf[4][0][k]-1)))

SampleU_100000[0:9] # Выведем часть выборки чисто из интереса
```

Out[41]:
```
[0.9102514672093804,
 0.7270333026847698,
 0.2549543283518632,
 0.8604882358514232,
 0.24540604778953254,
 0.9313967255929458,
 0.03653544312229743,
 0.24118572790140652,
 0.009760218564680906]
```

In [42]:
```
SampleU_1000 = sorted(SampleU_1000)
D_plus_list = []
D_minus_list = []
rv_u = sts.uniform()

n = len(SampleU_1000)
for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU_1000[k]))
        D_minus_list.append(abs((k)/n - SampleU_1000[k]))
D_minus = max(D_minus_list)
D_plus = max(D_minus_list)
D = max(D_minus, D_plus)
print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))

SampleU_100000 = sorted(SampleU_100000)
D_plus_list = []
D_minus_list = []

n = len(SampleU_100000)
for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU_100000[k]))
        D_minus_list.append(abs((k)/n - SampleU_100000[k]))
D_minus = max(D_minus_list)
D_plus = max(D_minus_list)
D = max(D_minus, D_plus)
print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  1000 ; D =  0.015530824611972138 ; sqrt(n)*D =  0.4911277971443273
n =  100000 ; D =  0.003134065121287888 ; sqrt(n)*D =  0.9910784118561591
```

Полученные значения лежат в пределах, соответствующих критерию. Гипотеза не отвергается.

Критерий согласия хи-квадрат

Распределение Ципфа относится к бесконечным дискретным (случайная величина может принимать бесконечное количество значений). Поэтому для определения числа интервалов воспользуемся эмпирическим правилом Стёрджеса. Тогда оптимальное количество интервалов для провекри критерия должно быть равно:

$N = 1 + [\log_2 n]$

В таком случае воспользуемся следующей группировкой интервалов: вероятности первых $[\log_2 n]$ значений и вероятность того, что случайная величина примет одно из оставшихся значений.

Далее определяем статистику хи-квадрат:

$X_N^2 = \sum_{i=1}^{N} \frac{(v_i^{(n)} - np_i)^2}{np_i} = \frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$, где w_i – отклонение от ожидаемой частоты встречи определённого элемента.

Алгоритм для подсчёта статистики предлагается следующий:

а) Найти частоты встречи каждого значения элемента выборки v_i (с учётом определённого количества интервалов)

б) Вычислить отклонения от соответстсвующей теоретической частоты встречи значения как $w_i = p_i \cdot n - v_i$

в) Рассчитать сумму $\frac{1}{n}\sum_{i=1}^{N} \frac{w_i^2}{p_i}$

Реализуем этот алгоритм:

In [47]:
```
rv = sts.zipf(2)

for i in range(len(SampleZipf)):
    for j in range(len(SampleZipf[i])):

        n = len(SampleZipf[i][j])
        N = 1 + int(np.log2(n))
        frequency = []
        for k in range(1, N):
            frequency.append(SampleZipf[i][j].count(k))
        frequency.append(n - sum(frequency))

        deviation = []
        for k in range(N-1):
            deviation.append(frequency[k] - n*rv.pmf(k+1))
        deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

        s = 0
        for k in range(N-1):
            s += deviation[k]**2 / (n*rv.pmf(k+1))
        s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
        print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.963962727813634
n =  5 ; N =  3 ; Статистика хи-квдрат =  16.384142869026945
n =  5 ; N =  3 ; Статистика хи-квдрат =  0.963962727813634
n =  5 ; N =  3 ; Статистика хи-квдрат =  4.142087563775288
n =  5 ; N =  3 ; Статистика хи-квдрат =  1.0968060691284431
n =  10 ; N =  4 ; Статистика хи-квдрат =  1.7651150878977695
n =  10 ; N =  4 ; Статистика хи-квдрат =  0.37843066087562527
n =  10 ; N =  4 ; Статистика хи-квдрат =  0.5429240675604479
n =  10 ; N =  4 ; Статистика хи-квдрат =  2.172512094676877
n =  10 ; N =  4 ; Статистика хи-квдрат =  3.159472534785813
n =  100 ; N =  7 ; Статистика хи-квдрат =  10.02194199506082
n =  100 ; N =  7 ; Статистика хи-квдрат =  7.016374401236185
n =  100 ; N =  7 ; Статистика хи-квдрат =  4.247651527659724
n =  100 ; N =  7 ; Статистика хи-квдрат =  6.440647477839305
n =  100 ; N =  7 ; Статистика хи-квдрат =  2.6905151557056683
n =  1000 ; N =  10 ; Статистика хи-квдрат =  12.74814515949598
n =  1000 ; N =  10 ; Статистика хи-квдрат =  6.621252271451196
n =  1000 ; N =  10 ; Статистика хи-квдрат =  19.59444935586827
n =  1000 ; N =  10 ; Статистика хи-квдрат =  5.154479863460608
n =  1000 ; N =  10 ; Статистика хи-квдрат =  2.774547769448004
n =  100000 ; N =  17 ; Статистика хи-квдрат =  17.828182400324593
n =  100000 ; N =  17 ; Статистика хи-квдрат =  14.182963956212681
n =  100000 ; N =  17 ; Статистика хи-квдрат =  34.51134472429103
n =  100000 ; N =  17 ; Статистика хи-квдрат =  14.619759781461093
n =  100000 ; N =  17 ; Статистика хи-квдрат =  16.33206949207905
```

Возьмём тот же уровень значимости $\alpha = 0.05$. Число степеней свободы равно $N - 1$. Для выборки объёма 5 ему соответствует табличное значение 6.0 – гипотеза отвергается для второй выборки. Для выборки объёма 10 – значение 7.8 – не отвергается ни для одной из выборок. Для объёма 100 – значение 12.6 – не отвергается ни для одной из выборок. Для объёма 1000 – значение 16.9 – отвергается для третьей выборки. Для объёма 100000 – значение 26.3 – отвергается для третьей выборки.

Распределение Ципфа показало себя менее устойчивым для данного критерия. Предположительно это можно объяснить тем, что дисперсии некоторых оценкок его параметров достаточно медленно стремятся к нулю с увеличением объёма выборки.

Случай сложной гипотезы.

Для рассматриваемого распределения предельные значения предельных статистик неизвестны. Поэтому поступим следующим образом:

От каждой выборки достаточного объёма (от 1000) возьмём из одной значение оценки неизвестного параметра, а к другой выборке такого же объёма проверим гипотезу о виде распределения.

In [49]:
```
rv = sts.zipf(2.0147705078125)

n = len(SampleZipf[3][1])
N = 1 + int(np.log2(n))
frequency = []
for k in range(1, N):
    frequency.append(SampleZipf[3][1].count(k))
frequency.append(n - sum(frequency))

deviation = []
for k in range(N-1):
    deviation.append(frequency[k] - n*rv.pmf(k+1))
deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

s = 0
for k in range(N-1):
    s += deviation[k]**2 / (n*rv.pmf(k+1))
s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)

rv = sts.zipf(1.9976806640625)

n = len(SampleZipf[4][1])
N = 1 + int(np.log2(n))
frequency = []
for k in range(1, N):
    frequency.append(SampleZipf[4][1].count(k))
frequency.append(n - sum(frequency))

deviation = []
for k in range(N-1):
    deviation.append(frequency[k] - n*rv.pmf(k+1))
deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

s = 0
for k in range(N-1):
    s += deviation[k]**2 / (n*rv.pmf(k+1))
s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  1000 ; N =  10 ; Статистика хи-квдрат =  7.290348658317714
n =  100000 ; N =  17 ; Статистика хи-квдрат =  16.31617530438343
```

Гипотезу можно считать подтверждённой, так как полученные значения лежат в допустимых границах в соответствии с табличными значениями.

Проверяем гипотезу однородности

Геометрическое распределение

Критерий однородности хи-квадрат

Так как критерий однородности Смирнова применим только для непрерывных распределений, воспользуемся сразу критерием хи-квадрат.

Формулируем критерий проверки:

$$n\sum_{i=1}^{k}\sum_{j=1}^{N}\frac{1}{n_i v_j}(v_{i,j}-v_j\frac{n_i}{n})^2 > t_\alpha$$

По заданию требуется сделать сравнение для каждой пары реализации выборок. Тогда можем упростить формулу для частного случая $k = 2$:

$$n_1 n_2\sum_{i=1}^{r}\frac{\frac{\mu_i}{n_1}-\frac{v_i}{n_2}}{\mu_i+v_i} > t_\alpha$$

Реализуем алгоритм поиска этой величины следующим образом:

а) Определяем, как находить величины:

$n_1 = n1$ – объём первой выборки;

$n_2 = n2$ – объём первой выборки;

r – количество итераций сложения, определим его по правилу $r = min(log_2(n_1), log_2(n_1)) + 1$ – по подобию правила Стёрджеса, интервалы делим аналогично предыдущему пункту работы;

$mu[i] = \mu_i$ – абсолютная частота встречаемости i-ого элемента с учётом смещения от нуля в первой выборке;

$nu[i] = v_i$ – абсолютная частота встречаемости i-ого элемента с учётом смещения от нуля во второй выборке.

б) Подсчитываем частоты в каждой выборке.

в) Само суммирование с проверкой, не равен ли нулю знаменатель.

Реализация:

In [227]:
```
def compareHi(sample1, sample2):
    n1 = len(sample1)
    n2 = len(sample2)
    r = int(1 + min(np.log2(n1), np.log2(n2)))
    mu = []
    nu = []
    for i in range(r-1):
        mu.append(sample1.count(i+1))
        nu.append(sample2.count(i+1))
    mu.append(n1 - sum(mu))
    nu.append(n2 - sum(nu))
    s = 0
    for i in range(r):
        if (mu[i] + nu[i] > 0):
            s += (mu[i]/n1 - nu[i]/n2)**2 / (mu[i] + nu[i])
        else:
            print("Caramba!")
    return s*n1*n2
```

Теперь сравним каждую из 25 выборок с каждой. Сочетаний из 25 элементов прилично, поэтому посчитаем все из них и выведем максимальное значение.

In [239]:
```
Hi = []

for i in range(25):
    for j in range(25):
        if (not ((int(i/5) == int(j/5)) and (i%5 == j%5))):
            Hi.append(compareHi(SampleGeom[int(i/5)][i%5],
                        SampleGeom[int(j/5)][j%5]))

print("Максимум: ", max(Hi),
     "\nСреднее: ", MEAN(Hi),
     "\nДисперсия: ", VAR(Hi))
```

```
Максимум:  25.966617812787256
Среднее:  5.131613444310244
Дисперсия:  27.23368901488325
```

Степени свободы находим по формуле $(k-1)\cdot(N-1)$, где k = 2 – количество сравнвиаемых выборок. Вспомним, что верхняя граница для наибольшей степени свободы из списка попарных сравнений выборок равна 26,3. Так же выборочное среднее достаточно мало, поэтому можем полагать, что большинство гипотез об однородности не отвергается.

Треугольное распределение

Критерий однородности Смирнова

Рассмотрим статистику:

$D_{n,m} = sup|\hat{F}_{1n} - \hat{F}_{2m}|$, где $\hat{F}_{1n}$ - эмпирическая функция распределения по первой выборке объёма n, $\hat{F}_{2m}$ - эмпирическая функция распределения по второй выборке объёма m.

По теореме Смирнова статистика:

$\sqrt{\frac{n\cdot m}{n+m}}D_{n,m}$ - имеет распределение Колмогорова.

Формулируем критерий:

Отвергаем гипотезу, если $D_{n,m} > t_\alpha(n,m)$, $t_\alpha(n,m) = \sqrt{\frac{1}{n}+\frac{1}{m}}t_\alpha$

Определяем способ нахождения $D_{n,m}$:

$D^+_{n,m} = max|\frac{k}{m} - \hat{F}_{1n}(Y_{(k)})| = max|\hat{F}_{2m}(X_{(k)})\frac{k-1}{n}|$

$D^-_{n,m} = max|\hat{F}_{1n}(Y_{(k)})\frac{k-1}{m}| = max|\frac{k}{n} - \hat{F}_{2m}(X_{(k)})|$

Тогда $D_{n,m} = max(D^+_{n,m}, D^-_{n,m})$.

Определяем алгоритм:

а) Определяем уже ранее используемую эмперическую функцию распределения по выборке

б) Строим вариационные ряды выборок, определяем их размеры

в) Создаём списки значений $|\frac{k}{m} - \hat{F}_{1n}(Y_{(k)})|$, $|\frac{k}{n} - \hat{F}_{2m}(X_{(k)})|$, получаем из них максимальное значение $D_{n,m}$

г) Считаем результат $\sqrt{\frac{n\cdot m}{n+m}}D_{n,m}$, которые остаётся лишь потом сравнить с соответстсвующим t_α.

Реализуем алгоритм:

In [3]:
```
def ECDF(x):
    counts = []
    xset = np.sort(list(set(x)))
    x = np.sort(x)
    for el in xset:
        counts.append(np.count_nonzero(x == el))
    def result(v):
        s = 0
        i = 0
        for i in range(len(xset)):
            if (v < xset[i]):
                break
            s += counts[i]
        return s / x.size
    return result
```

In [57]:
```
def compareSmirnov(sample1, sample2):
    sample1 = np.sort(list(sample1))
    sample2 = np.sort(list(sample2))
    n = len(sample1)
    m = len(sample2)
    dev1 = []
    dev2 = []
    F1 = ECDF(sample1)
    F2 = ECDF(sample2)
    for k in range(m):
        dev1.append(abs((k+1)/m - F1(sample2[k])))
    for k in range(n):
        dev2.append(abs((k+1)/n - F2(sample1[k])))
    return max(max(dev1), max(dev2))*sqrt(n*m/(n+m))
```

In [24]:
```
Smirnov = []

for i in range(25):
    for j in range(25):
        Smirnov.append(compareSmirnov(SampleTrig[int(i/5)][i%5],
                       SampleTrig[int(j/5)][j%5]))

max(Smirnov)
```

Out[24]:
```
1.363451502621197
```

Мы применили критерий для каждой пары из 25 выборок, а затем вывели наибольшее значение среди получившихся статистик. Оно оказлось равным 1.36, что очень точно соответстсвует границе при уровне значимости 0.05. Полученные выборки со скрипом, но не отвергаются критерию однородности Смирнова.

Теперь для получения более интересных результатов вспомним, что в первом домашнем задании мы использовали три различных способа моделирования треугольной случайной величины.

Насколько однородными будут выборки из всех трёх способов? Создадим соответстсвующие выборки по 1000 элементов и узнаем.

Моделируем как сумму двух независимых равномерно распределённых случайных величин:

In [74]:
```
def TrigCustomR():
        return 0.5*(random()+random())

SampleSum = [[int(TrigCustomR()*100)/100 for i in range (1000)]
             for j in range(5)]
```

Моделируем с помощью библиотечной функции:

In [75]:
```
SampleLib = [[int(sts.triang.rvs(0.5)*100)/100 for i in range (1000)]
             for j in range(5)]
```

Моделируем с помощью обратного преобразования (как и делали раньше):

In [76]:
```
def TrigCustom(p):
    if (p <= 0 or p >= 1):
        return 0
    n = random()
    if n < p:
        return sqrt(p * n)
    else:
        return 1 - sqrt((1-p)*(1-n))
SampleRev = [[TrigCustom(0.5) for i in range(1000)]
             for j in range(5)]
```

Выведем результаты попарного сравнения каждого из трёх видов моделирования:

In [77]:
```
for i in range(5):
    print("Сравнение выборок под номерами ", i+1)

    print("Обратное преобразование и сумма двух равномерных: ",
      compareSmirnov(SampleRev[i], SampleSum[i]))

    print("Обратное преобразование и библиотечная функция: ",
      compareSmirnov(SampleRev[i], SampleLib[i]))

    print("Библиотечная функция и сумма двух равномерных: ",
      compareSmirnov(SampleLib[i], SampleSum[i]))
```

```
Сравнение выборок под номерами  1
Обратное преобразование и сумма двух равномерных:  1.2521980673998823
Обратное преобразование и библиотечная функция:  1.0285912696499029
Библиотечная функция и сумма двух равномерных:  0.6931810730249355
Сравнение выборок под номерами  2
Обратное преобразование и сумма двух равномерных:  1.0956733089748967
Обратное преобразование и библиотечная функция:  1.744133022449835
Библиотечная функция и сумма двух равномерных:  1.229837387624883
Сравнение выборок под номерами  3
Обратное преобразование и сумма двух равномерных:  0.9167878707749146
Обратное преобразование и библиотечная функция:  0.7379024325749313
Библиотечная функция и сумма двух равномерных:  0.9167878707749146
Сравнение выборок под номерами  4
Обратное преобразование и сумма двух равномерных:  1.0509519494249022
Обратное преобразование и библиотечная функция:  1.1180339887498947
Библиотечная функция и сумма двух равномерных:  0.8944271909999167
Сравнение выборок под номерами  5
Обратное преобразование и сумма двух равномерных:  0.8720665112249163
Обратное преобразование и библиотечная функция:  0.827345151674923
Библиотечная функция и сумма двух равномерных:  1.4758048651498625
```

Делаем выводы:

Отклонение между выборками, смоделированными с помощью обратного преобразования и суммы двух равномерных, всегда лежет в допустимых границах, что подтверждает, что данные выборки однородные.

В случае сравнения с выборкой, сгенерированной библиотечной функцией, встречаются отклонения, превышающие допустимые границы. По рассматриваемому критерию гипотеза об однородности выборок должна быть отвергнута. Следовательно, можем предположить, что наиболее вероятно, библиотечная функция моделирует менее однородные выборки.

Критерий однородности хи-квадрат

Поступим аналогично случаю геометрического расспределения.

Формулируем критерий проверки:

$$n\sum_{i=1}^{k}\sum_{j=1}^{N}\frac{1}{n_i\nu_j}(\nu_{i,j}-\nu_j\frac{n_i}{n})^2 > t_\alpha$$

Для частного случая $k = 2$:

$$n_1n_2\sum_{i=1}^{r}\frac{\frac{\mu_i}{n_1}-\frac{\nu_i}{n_2}}{\mu_i+\nu_i} > t_\alpha$$

Реализуем алгоритм поиска этой величины следующим образом:

а) Определяем количество интервалов частот правилом Стёрджеса $N = 1 + \lfloor \log_2 n \rfloor$

б) Проходимся по каждому элементу в выборке, увеличиваем частоту встречи определённого интервала в соответствии с текущим элементом

в) Рассчитываем сумму

In [243]:
```
def compareHi(sample1, sample2):
    n1 = len(sample1)
    n2 = len(sample2)
    N = 1 + int(np.log2(min(n1,n2)))
    mu = [0]*N
    nu = [0]*N
    for element in sample1:
        mu[int(element*N)] += 1
    for element in sample2:
        nu[int(element*N)] += 1
    s = 0
    for i in range(N):
        if (mu[i] + nu[i] > 0):
            s += (mu[i]/n1 - nu[i]/n2)**2 / (mu[i] + nu[i])
        else:
            print("Caramba!")
    return s*n1*n2
```

Сравнение каждой пары выборки, выведем максимальное значение:

In [244]:
```
Hi = []

for i in range(25):
    for j in range(25):
        if (not ((int(i/5) == int(j/5)) and (i%5 == j%5))):
            Hi.append(compareHi(SampleTrig[int(i/5)][i%5],
                        SampleTrig[int(j/5)][j%5]))

print("Максимум: ", max(Hi),
     "\nСреднее: ", MEAN(Hi),
     "\nДисперсия: ", VAR(Hi))
```

```
Максимум:  21.154952357285335
Среднее:  4.046276744308426
Дисперсия:  18.970662486690067
```

Раз максимальное значение статистики лежит в допустимых границах для каждой пары выборок, и среднее мало, то делаем вывод, что каждая из выборок является однородной по данному критерию. Конечно, большая дисперсия уменьшает нашу уверенность в этом.

Теперь проверим гипотезы для выборок, смоделированных разными путями.

In [97]:
```
for i in range(5):
    print("Сравнение выборок под номерами ", i+1)

    print("Обратное преобразование и сумма двух равномерных: ",
      compareHi(SampleRev[i], SampleSum[i]))

    print("Обратное преобразование и библиотечная функция: ",
      compareHi(SampleRev[i], SampleLib[i]))

    print("Библиотечная функция и сумма двух равномерных: ",
      compareHi(SampleLib[i], SampleSum[i]))
```

```
Сравнение выборок под номерами  1
Обратное преобразование и сумма двух равномерных:  9.81615006089104
Обратное преобразование и библиотечная функция:  6.769810307543738
Библиотечная функция и сумма двух равномерных:  0.8872846111958138
Сравнение выборок под номерами  2
Обратное преобразование и сумма двух равномерных:  8.670941406248291
Обратное преобразование и библиотечная функция:  10.792554353360158
Библиотечная функция и сумма двух равномерных:  8.332598282527062
Сравнение выборок под номерами  3
Обратное преобразование и сумма двух равномерных:  5.815346482958771
Обратное преобразование и библиотечная функция:  6.82115325846833
Библиотечная функция и сумма двух равномерных:  4.455501810813648
Сравнение выборок под номерами  4
Обратное преобразование и сумма двух равномерных:  2.7994126351479642
Обратное преобразование и библиотечная функция:  6.550533723115377
Библиотечная функция и сумма двух равномерных:  9.685099867926938
Сравнение выборок под номерами  5
Обратное преобразование и сумма двух равномерных:  6.437540982645065
Обратное преобразование и библиотечная функция:  3.8752430983629162
Библиотечная функция и сумма двух равномерных:  8.223952895118563
```

Значения статистик лежат в допустимых границах. Получается, критерий хи-квадрат не смог выявить в некоторой степени неоднородность выборки, построенной по библиотечной функции, по отношению к другим выборкам. Гипотезы об однородности подтверждаются критерием хи-квадрат.

Распределение Ципфа

Критерий однородности хи-квадрат

Так как критерий однородности Смирнова применим только для непрерывных распределений, воспользуемся сразу критерием хи-квадрат.

Формулируем критерий проверки:

$$n\sum_{i=1}^{k}\sum_{j=1}^{N}\frac{1}{n_i v_j}(v_{i,j}-v_j\frac{n_i}{n})^2 > t_\alpha$$

Для частного случая $k = 2$:

$$n_1 n_2\sum_{i=1}^{r}\frac{\frac{\mu_i}{n_1}-\frac{v_i}{n_2}}{\mu_i+v_i} > t_\alpha$$

Реализуем алгоритм поиска этой величины следующим образом:

а) Определяем, как находить величины:

$n_1 = n1$ – объём первой выборки;

$n_2 = n2$ – объём первой выборки;

r – количество итераций сложения, можно найти как наибольшая реализация случайной величины из обеих выборок;

$mu[i] = \mu_i$ – абсолютная частота встречаемости i-ого элемента с учётом смещения от нуля в первой выборке;

$nu[i] = v_i$ – абсолютная частота встречаемости i-ого элемента с учётом смещения от нуля во второй выборке.

б) Подсчитываем частоты в каждой выборке.

в) Само суммирование с проверкой, не равен ли нулю знаменатель.

Реализация:

In [247]:
```
def compareHi(sample1, sample2):
    n1 = len(sample1)
    n2 = len(sample2)
    r = int(1 + min(np.log2(n1), np.log2(n2)))
    mu = []
    nu = []
    for i in range(r-1):
        mu.append(sample1.count(i+1))
        nu.append(sample2.count(i+1))
    mu.append(n1 - sum(mu))
    nu.append(n2 - sum(nu))
    s = 0
    for i in range(r):
        if (mu[i] + nu[i] > 0):
            s += (mu[i]/n1 - nu[i]/n2)**2 / (mu[i] + nu[i])
        else:
            print("Caramba!")
    return s*n1*n2
```

Теперь сравним каждую из 25 выборок с каждой. Сочетаний из 25 элементов прилично, поэтому посчитаем все из них и выведем максимальное значение.

In [249]:
```
Hi = []

for i in range(25):
    for j in range(25):
        if (not ((int(i/5) == int(j/5)) and (i%5 == j%5))):
            Hi.append(compareHi(SampleZipf[int(i/5)][i%5],
                        SampleZipf[int(j/5)][j%5]))

print("Максимум: ", max(Hi),
     "\nСреднее: ", MEAN(Hi),
     "\nДисперсия: ", VAR(Hi))
```

```
Максимум:  25.14367195326957
Среднее:  4.5457399660343984
Дисперсия:  19.735481053761504
```

Аналогично с предыдущими распределениями, большинство гипотез об однородности пар выборок следует считать не отклонёнными критерием.

Работа с данными*

При наличии данных мы можем лишь по признакам разной степени косвенности предполагать, относится ли полученная выборка к какому-либо распределению. Поэтому мы можем проверить лишь критерии согласия для сложных гипотез.

Задача ухудшается тем, что для рассматриваемых распределений верхние процентные точки не подсчитаны, даже в литературе их вычисления не встречаются.

Геометрическое распределение

Критерий согласия Смирнова

В предыдущей части работы для данных, (предположительно) распределённых геометрически, была предложена оценка неизвестного параметра, его значение было получено.

Чтобы проверить, относится ли исследуемая выборка к геометрическому распределнию, смоделируем геометрическое распределение, соответстсвующее полученному значению оценки неизвестного параметра. Алгоритм проверки сложной гипотезы оставим тот же, что и для искусственно созданных выборок геометрического распределения.

В этот раз у нас нет различных выборок, чтобы по оценке одной сравнить другую. Поэтому разделим данные на две части, по одним данным построим оценки, а по другим проверим гипотезу.

In [260]:
```
import csv
        #Выгружаем данные
durs = []
with open('optical_interconnection_network.csv', newline='') as File:
    reader = csv.DictReader(File)
    for row in reader:
        durs.append(int(row['Network Response Time'][:row['Network Response T
ime'].find(",")]))

durs1 = durs[:len(durs)//2]
durs2 = durs[len(durs)//2:]

p_geom =  1/MEAN(durs2)
p_geom # Оценка параметра
```

Out[260]:
```
0.0007514982997350968
```

In [261]:
```
durs = durs1

SampleU = []

rv = sts.geom(p_geom)

for k in range(len(durs)):
    SampleU.append(rv.cdf(durs[k]-1) + random()*(
            rv.cdf(durs[k])-rv.cdf(durs[k]-1)))
```

In [262]:
```
SampleU = sorted(SampleU)
D_plus_list = []
D_minus_list = []
rv_u = sts.uniform()

n = len(SampleU)
for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU[k]))
        D_minus_list.append(abs((k)/n - SampleU[k]))
D_minus = max(D_minus_list)
D_plus = max(D_minus_list)
D = max(D_minus, D_plus)
print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  320 ; D =  0.1759824204209593 ; sqrt(n)*D =  3.1480692392496974
```

К сожалению, полученное значение превосходит критические числа Колмогорова-Смирнова при всех обычно используемых уровнях значимости (0.1, 0.05, 0.01). Значит, мы не можем считать, что данная выборка относится к геометрическому распределнию с вероятностью, близкой к единице. Полученное значение близко к уровню значимости 0.00005.

Критерий согласия Смирнова

Реализован алгоритм, который равнее применялся к искусственно сгенерированным выборкам геометрического распределения.

In [263]:
```
rv = sts.geom(p_geom)

n = len(durs)
N = 1 + int(np.log2(n))
frequency = []
for k in range(1, N):
    frequency.append(durs.count(k))
frequency.append(n - sum(frequency))

deviation = []
for k in range(N-1):
    deviation.append(frequency[k] - n*rv.pmf(k+1))
deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

s = 0
for k in range(N-1):
    s += deviation[k]**2 / (n*rv.pmf(k+1))
s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  320 ; N =  9 ; Статистика хи-квдрат =  1.9303578948275966
```

Полученное значение лежит в допустимых границах. Поэтому с точки зрения критерия хи-квадрат гипотеза о том, что эта выборка принадлежит геометрическому распрделению, подтверждается.

Треугольное распределение

Критерий согласия Смирнова

Применяем аналогичный алгоритм, как и в случае выборок треугольного распределения, сгенерированных искусственно.

In [369]:
```
import csv

rates = []
with open('heart.csv', newline='') as File:
    reader = csv.DictReader(File)
    for row in reader:
        rates.append(int(row['thalach']))

def ratesToTrig(rates):
    new = rates
    for i in range(len(rates)):
        new[i] = rates[i] - min(rates)
    for i in range(len(rates)):
        new[i] = new[i]/max(new)
    return new

rateTrig = ratesToTrig(rates)

rateTrig1 = []
rateTrig2 = []

for i in range(len(rateTrig)//2):
    rateTrig1.append(rateTrig[i*2])
    rateTrig2.append(rateTrig[i*2+1])

p_trig = 3*MEAN(rateTrig2) - 1
p_trig
```

Out[369]:
```
0.8172081717681228
```

In [371]:
```
rateTrig = rateTrig1

rv = sts.triang(p_trig)

rateTrig = sorted(rateTrig)
D_plus_list = []
D_minus_list = []

n = len(rateTrig)
for k in range(1,n):
        D_plus_list.append(abs((k+1)/n - rv.cdf(
            rateTrig[k])))
        D_minus_list.append(abs((k)/n - rv.cdf(
            rateTrig[k])))
D_minus = max(D_minus_list)
D_plus = max(D_minus_list)
D = max(D_minus, D_plus)
print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  512 ; D =  0.1009377487255052 ; sqrt(n)*D =  2.283960531248273
```

Гипотезу следует опровергнуть, так как полученное значение не лежит в границах для уровня значимомости 0.05. Однако оно близко к уровню значимости 0.005.

Критерий согласия хи-квадрат

In [372]:
```
n = len(rateTrig)
N = 1 + int(np.log2(n))
frequency = [0]*(N+1)
for element in rateTrig:
    frequency[int(element*N)] += 1 # При встрече элемента - инкремент соответ
ствующей частоты

deviation = []
for k in range(N):
    deviation.append(frequency[k] - n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))

s = 0
for k in range(N):
    s += deviation[k]**2 / (n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))
print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  512 ; N =  10 ; Статистика хи-квдрат =  89.66715112506998
```

Полученное значение превышает допостимые границы. Гипотезу в связи с этим критерием следует отклонить.

Делаем, вывод, что предположение, что данная выборка относится к треугольному распределению, неверное.

Распределение Ципфа

Критерий согласия Смирнова

In [396]:
```
f = open('38650-password-sktorrent.txt', 'r', errors="ignore")
str = f.read()
pwlist = str.split('\n')

pwl = []
l = []

import collections

for pw in pwlist:
        pwl.append(pw[:6])

pwl1 = durs[:len(pwl)//2]
pwl2 = durs[len(pwl)//2:]

counter=collections.Counter(pwl)

for i in range(20):
    l += [i+1]*(counter.most_common(20)[i][1]-counter.most_common(21)[20][1])

p_zipf = 1.6729736328125 # Считалось ранее
```

In [397]:
```
aim = 0
for el in l:
    aim += np.log(el)
aim = aim/len(l)

p_zipf = getTheta(aim)
print(p_zipf)
```

```
1.6729736328125
```

In [126]:
```
SampleU = []

rv = sts.zipf(p_zipf)

for k in range(len(l)):
    SampleU.append(rv.cdf(l[k]-1) + random()*(
            rv.cdf(l[k])-rv.cdf(
                l[k]-1)))

SampleU = sorted(SampleU)
D_plus_list = []
D_minus_list = []
rv_u = sts.uniform()

n = len(SampleU)
for k in range(n):
        D_plus_list.append(abs((k+1)/n - SampleU[k]))
        D_minus_list.append(abs((k)/n - SampleU[k]))
D_minus = max(D_minus_list)
D_plus = max(D_minus_list)
D = max(D_minus, D_plus)
print('n = ', n, '; D = ', D, '; sqrt(n)*D = ', D*sqrt(n))
```

```
n =  313 ; D =  0.15966423825477516 ; sqrt(n)*D =  2.8247487304095724
```

Уровень значимости слишком мал, чтобы полученное значение входило в допустимые границы. Не удалось найти соответстсвующего значения для присвоения уровня значимости. Гипотезу следует отклонить.

Критерий согласия хи-квадрат

In [127]:
```
n = len(l)
N = 1 + int(np.log2(n))
frequency = []
for k in range(1, N):
    frequency.append(l.count(k))
frequency.append(n - sum(frequency))

deviation = []
for k in range(N-1):
    deviation.append(frequency[k] - n*rv.pmf(k+1))
deviation.append(frequency[N-1] - n*(1-rv.cdf(N-1)))

s = 0
for k in range(N-1):
    s += deviation[k]**2 / (n*rv.pmf(k+1))
s += deviation[N-1]**2 / (n*(1-rv.cdf(N-1)))
print('n = ', n, '; N = ', N, '; Статистика хи-квдрат = ', s)
```

```
n =  313 ; N =  9 ; Статистика хи-квдрат =  88.5693644769273
```

Полученное значение не лежит в допустимых границах. Поэтому, увы, гипотезу следует отклонить.

Поэтому маловероятно, что исследуемые данные относится к выбранному распределению.

Домашнее задание 5

Выбор данных

Выберем данные самого сложного распределения из имеющихся – данные распределения Ципфа.

In [1]:
```
# импортирование библиотек

from random import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
import math
from math import sqrt
%matplotlib inline
```

Создадим две выборки объёма 10 000 с параметрами 1.5 и 2. К сожалению, не удалось найти реальные данные, соответстсвующие сему закону – в предудщей части работы выяснилось, что найденные данные не стоит считать распределёнными по закону Ципфа.

In [92]:
```
def getH(s, N):
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    return H

def ZipfCustom(s, N):
    if (N != (N//1) or N<0):
        return 0 #Проверка на корректность ввода параметра
    H = getH(s, N)
    sum = 0
    i = 1
    n = random()
    while sum <= n:
        i += 1
        sum += ((i-1)**(-s))/H
    return i-1

Sample1 = [ZipfCustom(2, 500) for i in range(10000)]
Sample2 = [ZipfCustom(1.5, 500) for i in range(10000)]


print(*Sample1[:25], sep=' ') # Поглядим немного, что получилось
print(*Sample2[:25], sep=' ')
```

```
1 1 1 13 1 1 1 7 1 1 1 2 3 1 1 2 2 3 5 4 1 1 1 1 1
35 1 266 1 1 8 3 1 3 6 1 81 2 11 6 2 1 2 1 1 27 1 93 1 3
```

Постановка задачи

Пусть нам известно, что исследуемые данные распределены по закону Ципфа: $P(x) = \frac{x^{-\theta}}{H_{500,\theta}}$. Точно известно, что s равняется либо 1.5, либо 2. Нам нужно определить, чему равно θ. Соответственно выдвигаем две простые гипотезы – H_0 и H_1.

H_0: $\theta = \theta_1 = 2$;

H_1: $\theta = \theta_2 = 1.5$.

Формулируем определения.

Мы рассматриваем две простые гипотезы, в этом случае вводятся понятия ошибки первого и второго рода.

$P(X \in X_1 \mid H_0) = \alpha$ — ошибка 1 рода – отвергаем истину;

$P(X \in X_0 \mid H_1) = \beta$ — ошибка 2 рода – принимаем истину за ложь.

Функцией мощности критерия W называется функционал:

$W(F_X) = W(F_X; X_{1,\alpha}) = P(X \in X_{1,\alpha} \mid F_X)$ – вероятность попасть в $X_{1,\alpha}$ (X_1 при уровне значимости α) при условии, что верно F_x – искомое неизвестное распределние.

Вычисление функции отношения правдоподобия

$$\bar{l}(x) = \frac{\bar{L}(x,\theta_1)}{\bar{L}(x,\theta_0)} = \frac{\prod_{i=1}^{n} f_1(x_i)}{\prod_{i=1}^{n} f_0(x_i)} = \frac{\prod_{i=1}^{n} \frac{x_i^{-\theta_1}}{H_{500,\theta_1}}}{\prod_{i=1}^{n} \frac{x_i^{-\theta_0}}{H_{500,\theta_0}}} = \frac{\prod_{i=1}^{n} \frac{x_i^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_1}}}{\prod_{i=1}^{n} \frac{x_i^{-\theta_0}}{\sum_{k=1}^{500} k^{-\theta_0}}} = \left(\frac{\sum_{k=1}^{500} k^{-\theta_0}}{\sum_{k=1}^{500} k^{-\theta_1}}\right)^n \cdot \prod_{i=1}^{n} x_i^{\theta_0 - \theta_1}$$

Трудностей в вычислении нет – по виду это некоторая вычислимая константа умножить на произведение элементов выборки в нужной степени.

Вычисление критической области/количества материала

Выберем некоторую константу c, ведём условие, что если

$\bar{}$

$l(x) \geq c$, то принимаем, что верная гипотеза H_1.

$$(\frac{\sum_{k=1}^{500} k^{-\theta_0}}{\sum_{k=1}^{500} k^{-\theta_1}})^n \cdot \prod_{i=1}^{n} x_i^{\theta_0-\theta_1} \geq c$$

$$\prod_{i=1}^{n} x_i^{\theta_0-\theta_1} \geq c \cdot (\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^n$$

$$\sum_{i=1}^{n} ln(x_i^{\theta_0-\theta_1}) \geq ln(c \cdot \frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^n$$

$$(\theta_0 - \theta_1)\sum_{i=1}^{n} ln(x_i) \geq ln(c \cdot \frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^n$$

$(\theta_0 - \theta_1)$ положительно, поэтому знак неравенства сохраняется.

$$\sum_{i=1}^{n} ln(x_i) \geq ln(c \cdot \frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^n \cdot \frac{1}{(\theta_0-\theta_1)}$$

$$\prod_{i=1}^{n} x_i \geq c^{\frac{1}{(\theta_0-\theta_1)}} \cdot (\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^{\frac{n}{(\theta_0-\theta_1)}}$$

Конкретному уровню значимости соответстсвует своя константа c_α

$$(\frac{\sum_{k=1}^{500} k^{-\theta_0}}{\sum_{k=1}^{500} k^{-\theta_1}})^n \cdot \prod_{i=1}^{n} x_i^{\theta_0-\theta_1} \geq c_\alpha$$

Обозначим $c_\alpha^{\frac{1}{(\theta_0-\theta_1)}} \cdot (\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^{\frac{n}{(\theta_0-\theta_1)}} = t_\alpha$

$$\prod_{i=1}^{n} x_i \geq t_\alpha.$$

$$\alpha = P(\prod_{i=1}^{n} x_i \geq t_\alpha)$$

Обозначаем $r(X) = ln(\prod_{i=1}^{n} x_i) = \sum_{i=1}^{n} ln(x_i)$.

В случае больших выборок по Ц.П.Т:

$\varrho_\theta(r(X)) \sim N(\mu, \sigma^2)$,

$$\mu = M(r(X)) = M(\sum_{i=1}^{n} ln(x_i)) = \sum_{i=1}^{n} Mln(\xi) = nMln(\xi) = \frac{n}{H_{N,\theta}} \cdot \sum_{i=1}^{N} (ln(i) \cdot i^{-\theta})$$

$$\sigma^2 = D(r(X)) = D(\sum_{i=1}^{n} ln(x_i)) = n \cdot D(ln(x_i)) = n \cdot (Mln^2(\xi) - (Mln(\xi))^2) = n \cdot (\frac{1}{H_{N,\theta}} \cdot \sum_{i=1}^{N} (ln^2(i) \cdot i^{-\theta}) - (Mln(\xi))^2) = \frac{n}{H_{N,\theta}}$$

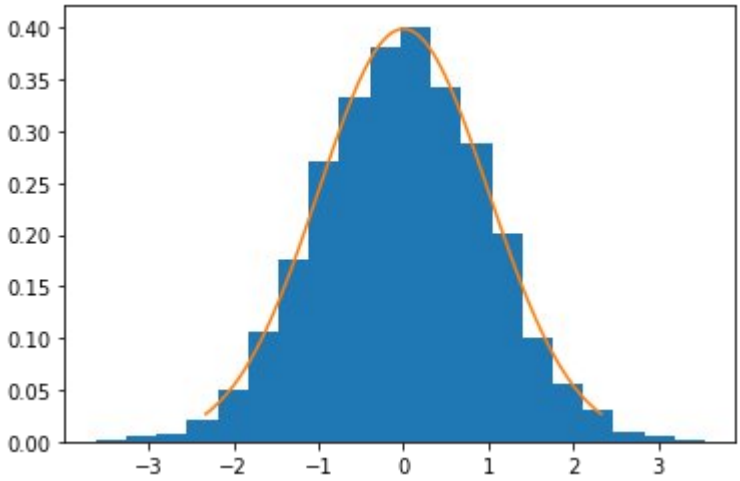
Проверим, что Ц.Т.П. хорошо выполняется для нашего объёма выборки. Создадим функции матожидания и дисперсии, функцию "сведения" к стандартному номральному распределению (всего лишь сместим на матожидание и сузим корнем из дисперсии).

Создадим 10000 выборок объёма 10000, для каждой выборки посчитаем статистику $r(X)$, и увидим, что рапспределение результатов близко к стандартному нормальному.

In [102]:
```
def getH(N, s):
    i = 0
    H = 0
    while i < N:
        i += 1
        H += i**(-s)
    return H

def getM(N, theta, n):
    i = 0
    s = 0
    while i < N:
        i += 1
        s += np.log(i)*i**(-theta)
    return n*s/getH(N, theta)

def getD(N, theta, n):
    i = 0
    s = 0
    while i < N:
        i += 1
        s += (i**(-theta))*(np.log(i)**2)
    return n*(s/getH(N, theta) - getM(N, theta, 1)**2)

def toNormal(N, s, sample):
    T = 0
    n = len(sample)
    for i in range(n):
        T += np.log(sample[i])
    return (T - getM(N, s, n))/(sqrt(getD(N, s, n)))
```

In [10]:
```
s = 3
N = 10000
zipfSamples = [sts.zipf.rvs(s, size=10000) for i in range(N)]

l = [toNormal(500, s, zipfSamples[i]) for i in range(N)]
plt.hist(l, bins = 20, density=True)
x = np.linspace(sts.norm.ppf(0.01),sts.norm.ppf(0.99), 100)
plt.plot(x, sts.norm.pdf(x, 0, 1))
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x122ddfd68>]
```



В качестве параметра взято $\theta = 3$, но это совершенно неважно – функция сведения это учитывает. Далее с помощью критерия согласия хи-квадрат попробуем опровергнуть гипотезу, что полученная выборка (сумм логарифомов случайных величин) в "сведённном" виде принадлежит стандартному нормальному распределнию.

Проверим как и в предудущей части работы.

In [285]:
```
rv = sts.norm()

n = len(l)
N = 1 + int(np.log2(n))
frequency = [0]*(N) # Массив с частотами
ma = max(l)+0.0001
mi = min(l)
for i in range(len(l)):
        frequency[int((l[i]-mi)/(ma-mi)*N)] += 1 # При встрече элемента - инк
ремент соответствующей частоты

deviation = []
for k in range(N):
        deviation.append(frequency[k]-
            n*(rv.cdf(mi+(ma-mi)*(k+1)/N)-rv.cdf(mi+(ma-mi)*k/N)))
deviation.append(rv.cdf(mi))
deviation.append(1 - rv.cdf(ma)) #в этих зонах нет значений выборок,
    #  хотя теоретически должны быть, поэтому надо добавть в отклонения
s = 0
for k in range(N):
    s += deviation[k]**2 / (n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))
print('n = ', n, '; N = ', N+2, '; Статистика хи-квдрат = ', s)
```

```
n =  10000 ; N =  16 ; Статистика хи-квдрат =  50.11264332558905
```

Пусть и статистика хи-квадрат небольшая, к сожалению, наша гипотеза должна быть отвергнута критерием. Однако поступим очень грубо и будем считать, что наша сведённая выборка всё равно стремится к стандартному нормальному распределению.

В этом можно убедиться, увеличивая объём генерируемой выборки выборки – чем больше становится объём, тем меньше статистика хи-квадрат, а значит, полагаем что при $n \to \infty$ мы действительно получим стандартное нормальное распределение. Так же вспомним, что с объёмом с увеличением объёма выборки среднее закона Ципфа не очень быстро стремится к нулю по сравнению с другими распределениями (это было изучено в предыдущих частях работы). Поэтому при столь больших объёмах статистика хи-квадрат до сих пор не лежит в допустимых границах. Тем не менее, видим, что с увеличением объёма выборки эта статистика к ним приближается. Ниже приведены поиски статистик для выборок Ципфа объёмами 100 и 1000.

In [6]:

```
s = 3
N = 10000
zipfSamples = [sts.zipf.rvs(s, size=100) for i in range(N)]

l = [toNormal(500, s, zipfSamples[i]) for i in range(N)]

rv = sts.norm()

n = len(l)
N = 1 + int(np.log2(n))
frequency = [0]*(N) # Массив с частотами
ma = max(l)+0.0001
mi = min(l)
for i in range(len(l)):
        frequency[int((l[i]-mi)/(ma-mi)*N)] += 1 # При встрече элемента - инк
ремент соответствующей частоты

deviation = []
for k in range(N):
        deviation.append(frequency[k]-
            n*(rv.cdf(mi+(ma-mi)*(k+1)/N)-rv.cdf(mi+(ma-mi)*k/N)))
deviation.append(rv.cdf(mi))
deviation.append(1 - rv.cdf(ma)) #в этих зонах нет значений выборок,
    #  хотя теоретически должны быть, поэтому надо добавть в отклонения
s = 0
for k in range(N):
    s += deviation[k]**2 / (n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))
print('n = ', n, '; N = ', N+2, '; Статистика хи-квдрат = ', s)
```

```
n =  10000 ; N =  16 ; Статистика хи-квдрат =  454.24784792042846
```

In [4]:

```
s = 3
N = 10000
zipfSamples = [sts.zipf.rvs(s, size=1000) for i in range(N)]

l = [toNormal(500, s, zipfSamples[i]) for i in range(N)]

rv = sts.norm()

n = len(l)
N = 1 + int(np.log2(n))
frequency = [0]*(N) # Массив с частотами
ma = max(l)+0.0001
mi = min(l)
for i in range(len(l)):
        frequency[int((l[i]-mi)/(ma-mi)*N)] += 1 # При встрече элемента - инк
ремент соответствующей частоты

deviation = []
for k in range(N):
        deviation.append(frequency[k]-
            n*(rv.cdf(mi+(ma-mi)*(k+1)/N)-rv.cdf(mi+(ma-mi)*k/N)))
deviation.append(rv.cdf(mi))
deviation.append(1 - rv.cdf(ma)) #в этих зонах нет значений выборок,
    #  хотя теоретически должны быть, поэтому надо добавть в отклонения
s = 0
for k in range(N):
    s += deviation[k]**2 / (n*(rv.cdf((k+1)/N)-rv.cdf(k/N)))
print('n = ', n, '; N = ', N+2, '; Статистика хи-квдрат = ', s)
```

```
n =  10000 ; N =  16 ; Статистика хи-квдрат =  71.06000253515772
```

Теперь будем работать с приведённой выборкой как с выборкой стандартного нормального распределения.

Её ошибка первого рода:

$$\alpha = P(\prod_{i=1}^{n} x_i \geq t_\alpha) = P(r(X) \geq ln(t_\alpha)) = P(\frac{r(X) - n\mu_0}{\sqrt{n}\sigma_0} \geq \frac{ln(t_\alpha) - n\mu_0}{\sqrt{n}\sigma_0}) = 1 - \Phi(\frac{ln(t_\alpha) - n\mu_0}{\sqrt{n}\sigma_0}) = 1 - \Phi(r_\alpha)$$

Находим квантиль уровня для $\alpha = 0.05$

$0.95 = \Phi(r_\alpha)$

$r_\alpha = 1.645$

Вспоминаем, что $c^{\frac{1}{(\theta_0 - \theta_1)}} \cdot (\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^{\frac{n}{(\theta_0 - \theta_1)}} = t_\alpha$

$$c_\alpha = (\frac{t_\alpha}{(\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^{\frac{n}{(\theta_0 - \theta_1)}}})^{(\theta_0 - \theta_1)}$$

$$c_\alpha = (\frac{e^{\mu_0 - \frac{r_\alpha}{\sqrt{n}}\sigma_0}}{(\frac{\sum_{k=1}^{500} k^{-\theta_1}}{\sum_{k=1}^{500} k^{-\theta_0}})^{\frac{n}{(\theta_0 - \theta_1)}}})^{(\theta_0 - \theta_1)} = e^{(n\mu_0 - r_\alpha\sqrt{n}\sigma_0) \cdot (\theta_0 - \theta_1)} \cdot (\frac{\sum_{k=1}^{500} k^{-\theta_0}}{\sum_{k=1}^{500} k^{-\theta_1}})^n$$

Все параметры для нахождения c_α известны.

То есть в нашем случае критерий Неймана-Пирсона задаётся критической областью

$X_{1\alpha}^{*+} = \{r(X) \leq t_\alpha\}$ или, если расписать

$$\{\sum_{i=1}^{n} ln(x_i) \leq \mu_0 - \frac{r_\alpha}{\sqrt{n}}\sigma_0\}$$

Возвращаемся к двум выборкам, сгенерированным в начале этой части работы.

Находим константу t_α

In [91]:
```
r = sts.norm.ppf(0.95)
theta = 2
N = 500
n = 10000
t_alpha = getM(N, theta, n) - r*sqrt(getD(N, theta, n))/sqrt(n)
t_alpha
```

Out[91]: 5617.292088621546

Статистика $r(X)$ для первой выборки:

In [93]:
```
s = 0
for el in Sample1:
    s += np.log(el)
s
```

Out[93]: 5579.959291567311

Статистика $r(X)$ для второй выборки:

In [84]:
```
s = 0
for el in Sample2:
    s += np.log(el)
s
```

Out[84]: 12636.49243448158

Значение статистики для второй попадает в критическую область, поэтому гипотеза $H_0 : \theta = \theta_0 = 2$ не отклоняется только для первой выборки.

Пусть ошибкой второго рода будет $\beta = \alpha = 0.05$

Выразим ошибку второго рода:

$$1 - \beta = P(\frac{\sqrt{n}}{\sigma}(r(X) - \theta_0) \geq t_\alpha) =$$

$$= P(\frac{\sqrt{n}}{\sigma}(r(X) - \theta_1) \geq t_\alpha - \frac{\sqrt{n}}{\sigma}(\theta_1 - \theta_0) = \Phi(-t_\alpha + \frac{\sqrt{n}}{\sigma}(\theta_1 - \theta_0))$$

$$\beta = \Phi(t_\alpha - \frac{\sqrt{n}}{\sigma}(\theta_1 - \theta_0))$$

$$t_\beta = t_\alpha - \frac{\sqrt{n}}{\sigma}(\theta_1 - \theta_0)$$

$$n = \sigma^2 \cdot \frac{(t_\beta - t_\alpha)^2}{(\theta_1 - \theta_0)^2}$$

Так как n - целое:

$$n = 1 + \lfloor \sigma^2 \cdot \frac{(t_\beta - t_\alpha)^2}{(\theta_1 - \theta_0)^2} \rfloor$$

Пусть ошибкой второго рода будет $\beta = 0.05$

In [134]:
```
t_alpha = t_beta = - sts.norm.ppf(0.05)
theta_0 = 2
theta_1 = 1.5

1 + int(getD(N, theta, n)*((t_beta+t_alpha)**2)/((theta_1 - theta_0)**2))
```

Out[134]: 359542

Таков минимальный объём выборки, чтобы с заданными ошибками первого и второго родов различить две простые гипотезы с заданными параметрами.