Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ФУНДАМЕНТАЛЬНЫЕ НАУКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА (ФН11)

НАПРАВЛЕНИЕ ПОДГОТОВКИ МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ (02.03.01)

О т ч е т

по лабораторной работе № 1

Название лабораторной работы:
Первоначальная обработка статистических данных

Вариант № 9

Дисциплина:
Теория вероятности и математическая статистика

Студент группы ФН11-52Б _____ **Очкин Н.В.**
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Облакова Т.В.**
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Содержание

1 Задание

По данной выборке

1. Найдите крайние члены вариационного ряда и размах выборки
2. Осуществите группировку данных (количество интервалов находим по правилу Стерджеса)
3. По сгруппированным данным постройте гистограмму относительных частот
4. Вычислите выборочное среднее и выборочную дисперсию.
5. По виду гистограммы определите возможный закон распределения, оцените параметры этого закона по методу моментов, постройте совмещенные графики гистограммы и плотности предполагаемого закона
6. Найдите эмпирическую функцию распределения и постройте совмещенные графики эмпирической и теоретической функций распределения

2 Исходные данные

14.495	4.715	7.175	8.428	11.093	3.375	12.906	8.415	8.916	13.48
5.343	17.985	15.992	13.89	9.838	13.924	9.012	9.458	17.69	6.542
14.396	8.592	8.206	14.237	7.357	10.821	12.767	16.058	12.959	4.354
12.888	10.268	9.182	5.647	8.282	2.903	15.988	12.959	14.919	6.339
2.375	17.921	9.097	15.85	11.449	11.095	9.493	12.175	7.479	13.535
9.234	6.078	4.964	6.355	13.957	12.911	15.694	14.286	9.869	5.175
5.811	7.241	5.814	3.086	6.875	3.878	5.333	15.134	12.924	9.159
4.727	4.646	15.535	9.919	17.117	10.351	16.892	12.423	10.511	4.942
4.843	9.927	15.864	3.635	17.963	8.25	5.14	6.734	12.622	13.325
3.377	16.195	12.04	12.768	2.744	14.186	9.354	15.439	14.612	15.649
8.681	5.006	3.608	2.867	12.177	15.506	7.683	14.022	17.103	8.905
12.173	17.757	6.883	2.666	9.861	5.743	16.175	15.308	7.039	15.238

Таблица 1: Исходные данные

3 Решение

Данная работа была выполнена с использованием языка программирования PYTHON и следующих модулей и библиотек:

- numpy
- scipy
- pyplot
- pandas
- IPython
- display
- matplotlib
- math
- stats

3.1 Найдите крайние члены вариационного ряда и размах выборки

Обратимся к листингу 3 и найдем крание члены вариационного ряда как минимальное и максимальное значения набора данных, а также размах выборки, как их разницу.

Крайние члены: 2.375, 17.985

Размах выборки: 15.61

3.2 Осуществите группировку данных

Для начала определим количество интервалов, воспользовавшись правилом Стерджеса:

$$n = 1 + \lfloor \log_2 N \rfloor,$$

где N — общее число наблюдений величины,
$\log_2$ — логарифм по основанию 2,
$\lfloor x \rfloor$ — обозначает целую часть числа x.

И определим шаг интервала разделив размах выборки на количество интервалов. Все операции представлены в листинге 4.

Количество интервалов: 7

Шаг интервала: 2.23

Теперь сгруппируем данные. Алгоритм группировки, представленный в листингах 5 и 6, опишем в виде псевдокода:

Псевдокод алгоритма группировки данных

```
интервалы = [i для i в диапазоне от 1 до k включительно]

интервалы = []
частоты = []
относительныеЧастоты = []
серединыИнтервалов = []

текущий = минимальныйЭлемент
пока текущий + шаг <= максимальныйЭлемент:
    первыйЭлемент = текущий
    второйЭлемент = текущий + шаг

    интервалы.добавить(первыйЭлемент, второйЭлемент)

    серединаИнтервала = (первыйЭлемент + второйЭлемент) / 2
    серединыИнтервалов.добавить(серединаИнтервала)

    частота = найтиЧастоту(первыйЭлемент, второйЭлемент, данные)
    частоты.добавить(частота)

    относительнаяЧастота = частота / общееКоличествоЭлементов
    относительныеЧастоты.добавить(относительнаяЧастота)

    текущий = текущий + шаг

функция найтиЧастоту(минимальный, максимальный, данные):
    счетчик = 0

    для элемента в данные:
        если минимальный <= элемент < максимальный:
            счетчик += 1

    вернуть счетчик
```

Представим полученную группировку в виде таблицы:

номер интервала	интервал	середина интервала	частота	относительная частота
0	[2.375; 4.605)	3.490	12	0.100
1	[4.605; 6.835)	5.720	20	0.167
2	[6.835; 9.065)	7.950	18	0.150
3	[9.065; 11.295)	10.180	18	0.150
4	[11.295; 13.525)	12.410	17	0.142
5	[13.525; 15.755)	14.640	20	0.167
6	[15.755; 17.985)	16.870	14	0.117

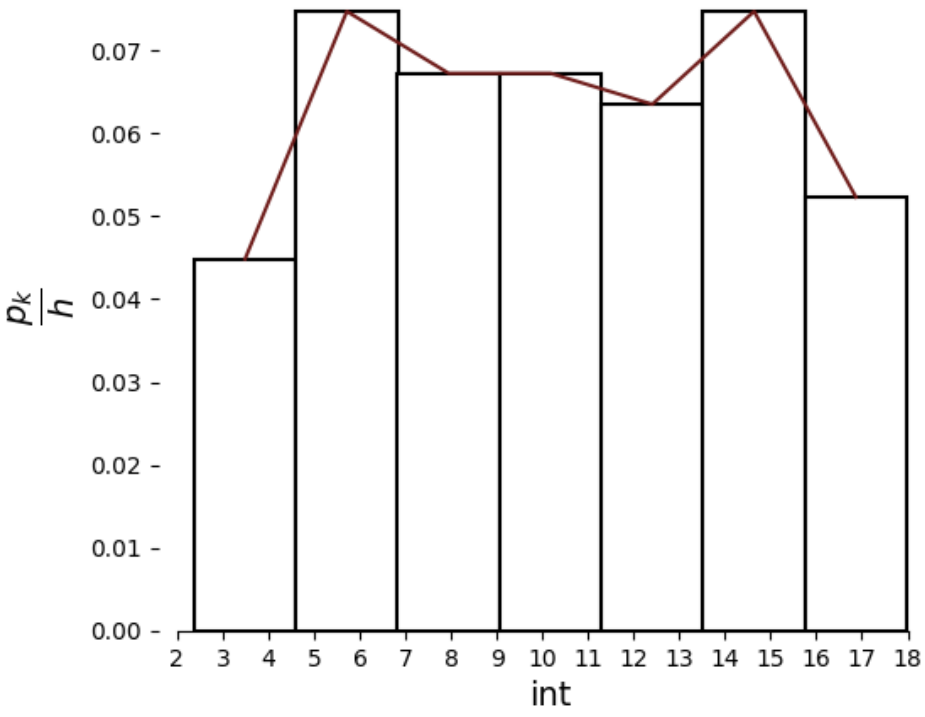
Таблица 2: Сгруппированные данные

А также найдем границы интервалов (листинг 8):

$$[2.375 \quad 4.605 \quad 6.835 \quad 9.065 \quad 11.295 \quad 13.525 \quad 15.755 \quad 17.985]$$

3.3 По сгруппированным данным постройте гистограмму относительных частот

Воспользуемся библиотекой MATPLOTLIB для построения гистограммы (листинг 9)



3.4 Вычислите выборочное среднее и выборочную дисперсию

По следующим формулам определим выборочное среднее и выборочную дисперсию соответственно:

$$\overline{X} = \frac{1}{n}\sum_{k=1}^{n} X_k \qquad\qquad S^2 = \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \overline{X})^2$$

Обратимся к листингу 10 для вычислений:

Выборочное среднее: 10.2681

Выборочная дисперсия: 19.611

3.5 По виду гистограммы определите возможный закон распределения, оцените параметры этого закона по методу моментов, постройте совмещенные графики гистограммы и плотности предполагаемого закона

Поскольку по виду гистограммы затруднительно дать точный ответ к какому закону распределения относится заданная выборка данных, воспользуемся критерием Колмогорова-Смирнова. В качестве гипотез выберем следующие часто встречающиеся законы распределения:

- Нормальный
- Пуассоновский
- Логнормальный
- Хи-квадрат
- Равномерный
- Бета
- Экспоненциальный
- t-распределение Стьюдента
- Распределение Вейбулла
- Гамма

Применять критерий Колмогорова-Смирнова будем при помощи функции KSTEST модуля STATS библиотеки SCIPY, которая на выходе дает 4 параметра:

1. STATISTIC **(или D)** - это основная статистика теста Колмогорова-Смирнова:
   - Она представляет собой максимальную разницу между эмпирической функцией распределения (наблюдаемые данные) и теоретической функцией распределения.
   - Значение лежит в диапазоне от 0 до 1. Чем больше значение, тем сильнее различие между данными и теоретическим распределением.
2. PVALUE - это значение p-уровня значимости:

- Оно показывает вероятность того, что наблюдаемое различие между эмпирическим и теоретическим распределениями могло возникнуть случайно при условии, что нулевая гипотеза (о соответствии распределению) верна.
- Если PVALUE меньше заранее установленного уровня значимости (в нашем случае, 0.05), то нулевая гипотеза отвергается, что означает, что данные не соответствуют предполагаемому распределению.

3. STATISTIC-LOCATION - это точка, в которой наблюдается максимальная разница между эмпирической и теоретической функциями распределения:
   - Это помогает понять, на каком участке распределения данные сильнее всего отличаются от теоретической модели.
4. STATISTIC-SIGN - это знак отклонения в точке максимальной разницы:
   - Может принимать значения 1 или -1.
   - Если STATISTIC-SIGN = -1, это означает, что в точке максимального отклонения эмпирическая функция распределения ниже, чем теоретическая.
   - Если бы значение было 1, это указывало бы на то, что эмпирическая функция распределения выше, чем теоретическая в этой точке.

Первоначальный подбор параметров для каждого из закона осуществим при помощи следующей функции: SCIPY.STATS.<ЗАКОН>.FIT. Далее вручную методом подгонки, смотря на график, подберем параметр еще лучше.

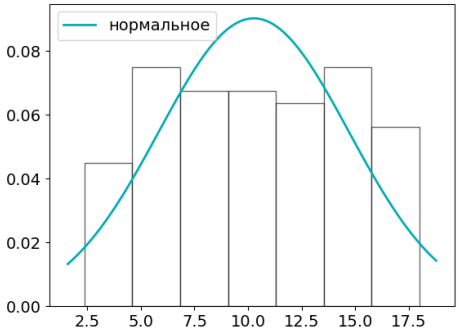
Для каждого проверяемого закона также построим совмещенный график с гистограммой исходных данных.
Код предоставлен в листингах 11 и 12

Нормальный

Среднее, μ :	10.2681
Стандратное отклонение, σ :	4.4284

Результаты теста Колмогорова-Смирнова:

statistic:	0.08872
pvalue:	0.28428
statistic-location:	12.767
statistic-sign:	-1

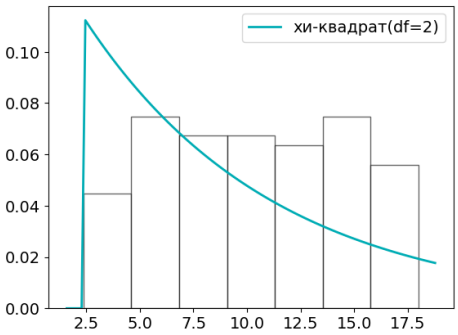


Хи-квадрат

Количество степеней свободы:	2
Сдвиг :	2.375
Параметр масштаба :	4.4099

Результаты теста Колмогорова-Смирнова:

statistic:	0.17036
pvalue:	0.00163
statistic-location:	17.985
statistic-sign:	1

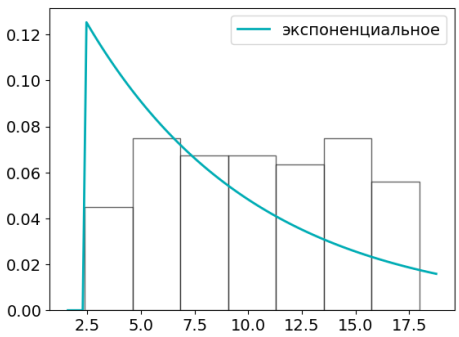


Экспоненциальный

Сдвиг:	2.375
Параметр масштаба, $\frac{1}{\lambda}$:	7.8931

Результаты теста Колмогорова-Смирнова:

statistic:	0.18895
pvalue:	0.00032
statistic-location:	8.206
statistic-sign:	-1

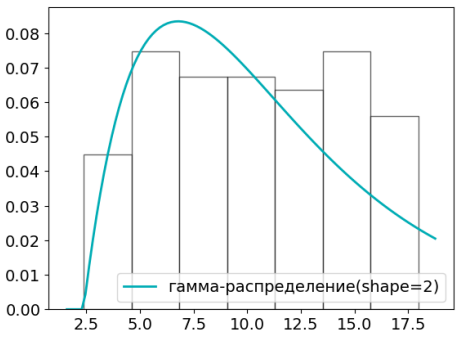


Гамма

Параметр формы:	2
Сдвиг:	2.375
Параметр масштаба:	4.4099

Результаты теста Колмогорова-Смирнова:

statistic:	0.13175
pvalue:	0.02818
statistic-location:	17.985
statistic-sign:	1

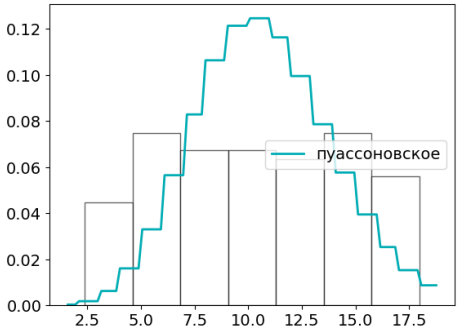


Пуассоновский

Параметр интенсивности: 10.2681

Результаты теста Колмогорова-Смирнова:

statistic:	0.19049
pvalue:	0.00027
statistic-location:	12.04
statistic-sign:	−1



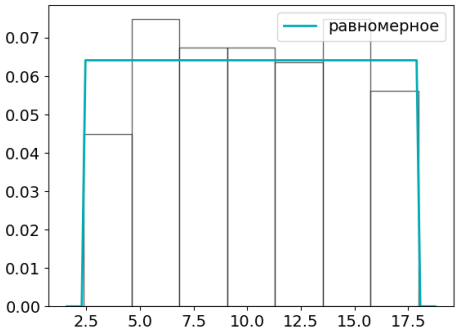
Равномерный

Нижняя граница: 2.375

Размах: 15.61

Результаты теста Колмогорова-Смирнова:

statistic:	0.048
pvalue:	0.93243
statistic-location:	16.195
statistic-sign:	1



t-распределение Стьюдента

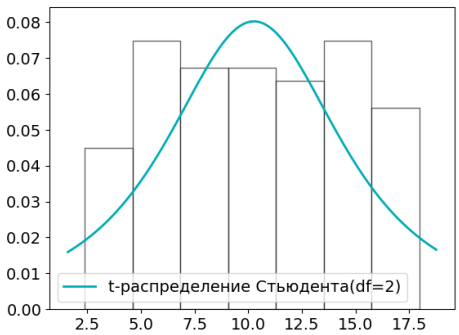
Кол-во степеней свободы: 2

Сдвиг: 10.2681

Масштаб: 4.4099

Результаты теста Колмогорова-Смирнова:

statistic:	0.11112
pvalue:	0.09562
statistic-location:	17.985
statistic-sign:	1



Логнормальный

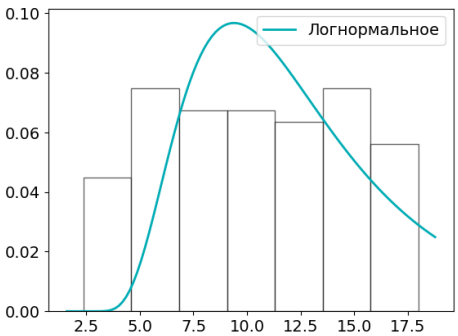
Параметр формы: 0.515

Сдвиг: 2.375

Параметр масштаба: 9.1469

Результаты теста Колмогорова-Смирнова:

statistic:	0.19857
pvalue:	0.00013
statistic-location:	6.883
statistic-sign:	1



Бета

Параметр формы 1:	1.2407
Параметр формы 2:	0.9441
Сдвиг:	0.924
Масштаб:	17.0606

Вейбулловский

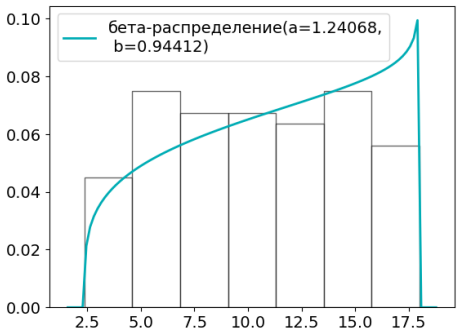
Параметр формы:	2.4993
Сдвиг:	2.375
Параметр масштаба:	4.4099

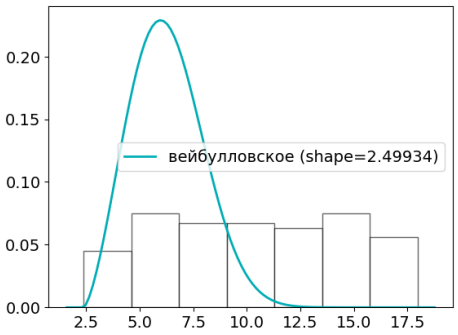
Результаты теста Колмогорова-Смирнова:

statistic:	0.0842
pvalue:	0.34331
statistic-location:	9.927
statistic-sign:	1

Результаты теста Колмогорова-Смирнова:

statistic:	0.5389
pvalue:	0.0
statistic-location:	8.905
statistic-sign:	−1





Запишем все данные в таблицу

Index	Name	Statistic	P-Value	Statistic Location	Statistic Sign
0	norm	0.089	0.284	12.767	-1
1	chi2	0.170	0.002	17.985	1
2	expon	0.189	0.000	8.206	-1
3	gamma	0.132	0.028	17.985	1
4	poisson	0.190	0.000	12.040	-1
5	uniform	0.048	0.932	16.195	1
6	t	0.111	0.096	17.985	1
7	lognorm	0.199	0.000	6.883	1
8	beta	0.084	0.343	9.927	1
9	weibull_min	0.539	0.000	8.905	-1

Таблица 3: Результаты критерия Колмогорова-Смирнова

Избавимся от строк таблицы со значениями P-VALUE < 0.05

Index	Name	Statistic	P-Value	Statistic Location	Statistic Sign
0	norm	0.089	0.284	12.767	-1
5	uniform	0.048	0.932	16.195	1
6	t	0.111	0.096	17.985	1
8	beta	0.084	0.343	9.927	1

Таблица 4: Отфильтрованные результаты критерия Колмогорова-Смирнова

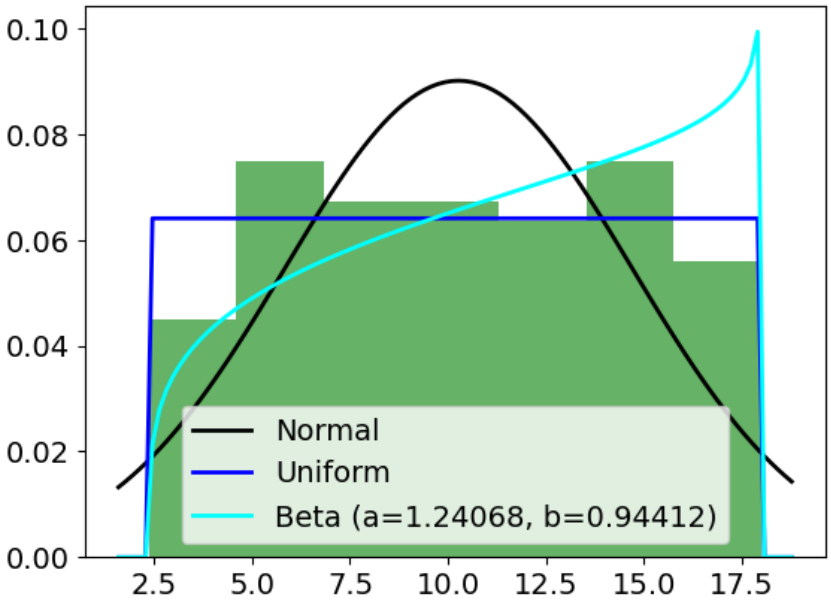
Отсортируем по значению STATISTIC в порядке убывания

Index	Name	Statistic	P-Value	Statistic Location	Statistic Sign
5	uniform	0.048	0.932	16.195	1
8	beta	0.084	0.343	9.927	1
0	norm	0.089	0.284	12.767	-1
6	t	0.111	0.096	17.985	1

Таблица 5: Отсортированные результаты критерия Колмогорова-Смирнова

Исходя из полученных данных, можно сделать вывод, что закон распределения выборки с наибольшей вероятностью является **равномерным**, далее **бета-распределением**, **нормальным** и, наконец, **t-распределением Стьюдента** (с указанными ранее параметрами).

Изобразим плотности первых трех законов на совмещенном графике с гистограммой, построенной на заданной выборке данных (листинг 13)



Оценим параметры равномерного закона распределения методом моментов

$$M_\xi = \int_{-\inf}^{+\inf} x p_\xi(x) dx = \int_a^b x \cdot \frac{1}{b-a} dx = \frac{a+b}{2}$$

$$M_\xi = \overline{X} \Longrightarrow \frac{a+b}{2} = \overline{X} \Longrightarrow a + b = 2\overline{X} \approx 20.5362$$

$$D_\xi = M_\xi^2 - (M_\xi)^2$$

$$M_\xi^2 = \int_{-\inf}^{+\inf} x^2 p_\xi(x) dx = \int_a^b x^2 \cdot \frac{1}{b-a} dx = \frac{a^2 + ab + b^2}{3}$$

$$(M_\xi)^2 = \frac{(a+b)^2}{4}$$

$$D_\xi = \frac{(b-a)^2}{12}$$

$$D_\xi = S^2 \Longrightarrow \frac{(b-a)^2}{12} = S^2 \Longrightarrow (b-a)^2 = 12S^2 \Longrightarrow b - a = 2\sqrt{3}\sqrt{S^2} \approx 15.3405$$

$$\begin{cases} b - a = 15.3405 \\ b + a = 20.5362 \end{cases}$$

$$\begin{cases} a = 2.59785 \\ b = 17.93835 \end{cases}$$

(Можно заметить, что найденные параметры не сильно отличаются от тех, что были найдены программно)

3.6 Найдите эмпирическую функцию распределения и постройте совмещенные графики эмпирической и теоретической функций распределения

Запишем реализацию эмпирической функции распределения femp(x) в виде псевдокода:

Псевдокод функции femp

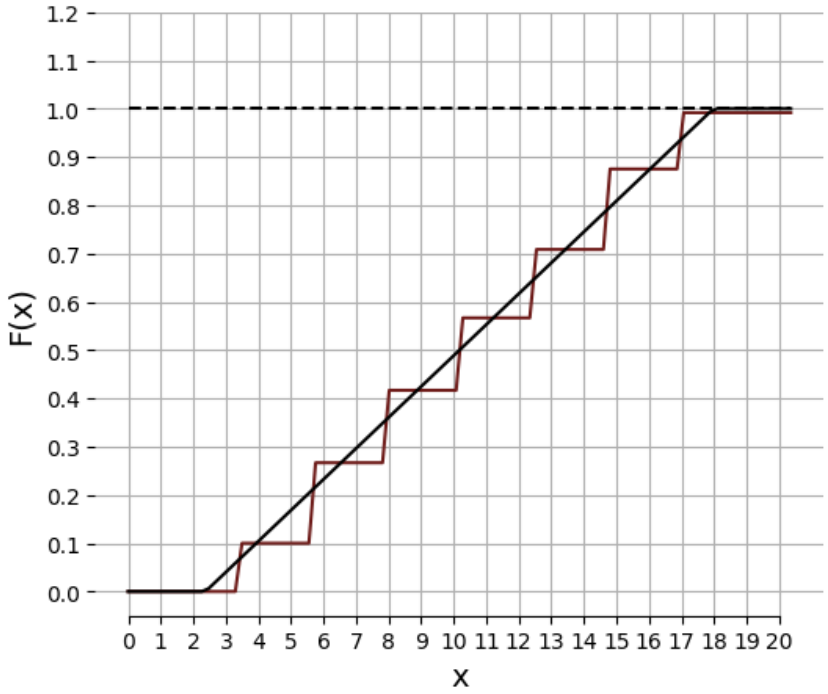
```
функция femp(x) {
    функция ind(x) {
        возвращает 1 если x > 0, иначе возвращает 0
    }

    сумма = 0
    для i в диапазоне (количества интервалов) {
        сумма += i-й элемент вектора относительных частот * \
                    ind(x минус i-й элемент вектора с \
                                середединами интервалов)
    }

    вернуть сумму
}
```

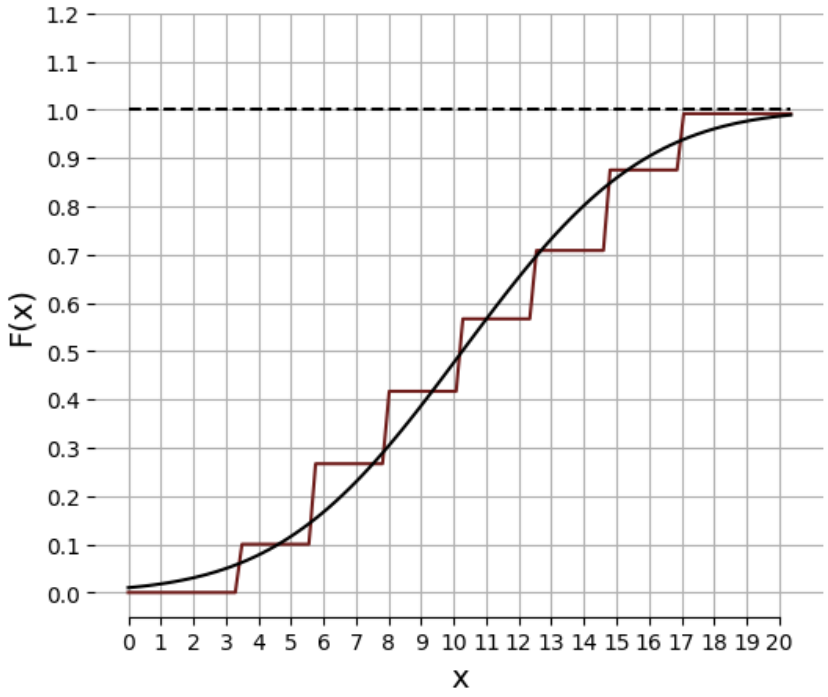
Сам же код реализации функции предоставлен в листинге 14.

Построим совмещенные графики эмпирической функции распределения и функции распределения для равномерного закона (a = 2.59785, b = 17.93835), воспользовавшись листингами 15 и 16.



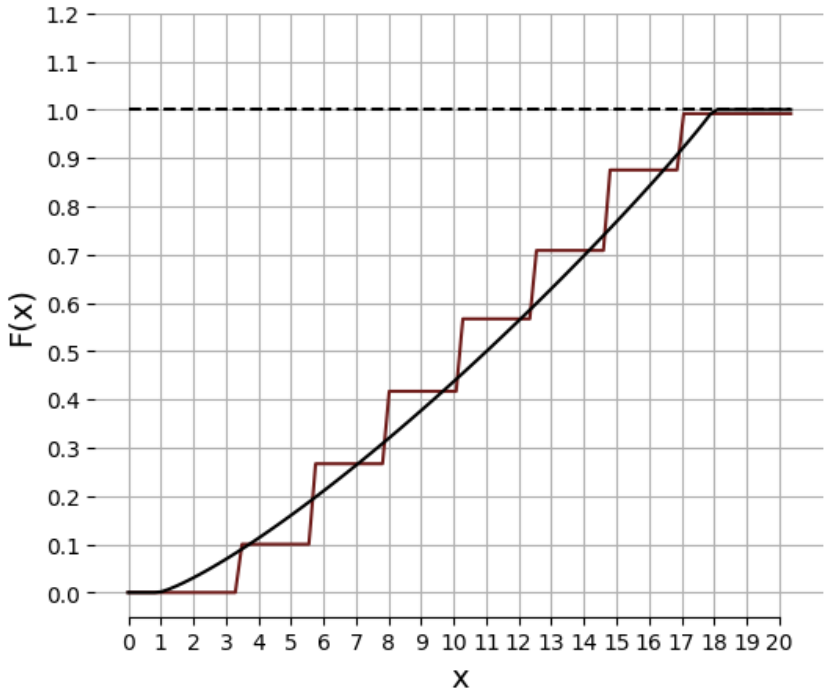
Теперь тоже самое, но для нормального закона ($\mu = 10.2681$, $\sigma^2 = 19.6109$) (листинги 15 и 17).

Примечание: параметры были найдены с помощью библиотеки SCIPY.



И для бета-распределения ($\alpha = 1.2407$, $\beta = 0.94412$, сдвиг (loc) $= 2.375$, масштаб (scale) $= 4.4099$) (листинги 15 и 18).

Примечание: параметры были найдены с помощью библиотеки SCIPY.



3.7 Вывод

Первоначальная обработка позволяет предварительно отнести выборку (с наибольшей вероятностью) к равномерному закону с параметрами:

$$a = 2.59785 \qquad b = 17.93835$$

4 Приложение

Программный код, с помощью которого была выполнена данная лабораторная работа.

Примечание.

Так как отчет был написан с использованием дистрибутива TeX и следующий код был отформатирован с использованием окружения LSTLISTING, в некоторых местах текст, написанный на русском языке, может иметь проблемы с выравниванием, пробелами, шрифтом и т.д. Это связано с тем, что библиотека LISTINGS, из которой мы берем окружение для форматирования кода, плохо работает с Unicode.

Листинг 1: Подключение библиотек

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
import math
from IPython.display import display, Math

PRECISION = 5
pd.set_option('display.float_format', '{:.3f}'.format)
np.set_printoptions(precision=PRECISION)
```

Листинг 2: Ввод данных

```
_data = [
    14.495, 4.715,  7.175,  8.428,  11.093, 3.375
    12.906, 8.415,  8.916,  13.48,  5.343,  17.985
    15.992, 13.89,  9.838,  13.924, 9.012,  9.458
    17.69,  6.542,  14.396, 8.592,  8.206,  14.237
    7.357,  10.821, 12.767, 16.058, 12.959, 4.354
    12.888, 10.268, 9.182,  5.647,  8.282,  2.903
    15.988, 12.959, 14.919, 6.339,  2.375,  17.921
    9.097,  15.85,  11.449, 11.095, 9.493,  12.175
    7.479,  13.535, 9.234,  6.078,  4.964,  6.355
    13.957, 12.911, 15.694, 14.286, 9.869,  5.175
    5.811,  7.241,  5.814,  3.086,  6.875,  3.878
    5.333,  15.134, 12.924, 9.159,  4.727,  4.646
    15.535, 9.919,  17.117, 10.351, 16.892, 12.423
    10.511, 4.942,  4.843,  9.927,  15.864, 3.635
    17.963, 8.25,   5.14,   6.734,  12.622, 13.325
    3.377,  16.195, 12.04,  12.768, 2.744,  14.186
    9.354,  15.439, 14.612, 15.649, 8.681,  5.006
    3.608,  2.867,  12.177, 15.506, 7.683,  14.022
    17.103, 8.905,  12.173, 17.757, 6.883,  2.666
    9.861,  5.743,  16.175, 15.308, 7.039,  15.238
]

\begin{lstlisting}
# Variation series
```

```
_parsedData = [_data[i:i+10] for i in range(0, len(_data), 10)]

_parsedData
```

1. Крайние члены вариационного ряда и размах выборки

Листинг 3: Поиск крайних членов и размаха выборки

```
# Extreme members
_maxEl = np.max(_parsedData)
_minEl = np.min(_parsedData)

print(f"Крайние члены: {_maxEl}, {_minEl}")

# Sample range
_range = np.round(_maxEl - _minEl, PRECISION)

print(f"Размах выборки: {_range}")
```

2. Осуществите группировку данных

Листинг 4: Поиск интервалов

```
# Let's find the number of intervals using Sturges' rule
_n = len(_data)
_k = 1 + np.trunc(np.log2(_n))

print(f"Количество интервалов: {_k}")

# Interval step
_h = _range / _k

print(f"Шаг интервала: {_h}")
```

Листинг 5: Функция для нахождения частот

```
# function for finding frequencies
def findFrequency(minim, maxim, data):
    count = 0

    for el in data:
        if minim <= el < maxim:
            count += 1

    return count
```

Листинг 6: Группировка данных

```
intervals           = [i for i in range(1, int(_k) + 1)] # interval number
tableRanges         = [] # interval
frequencies         = [] # frequency
relativeFrequencies = [] # relative frequency
midRanges           = [] # middle of the interval

curr = _minEl
while (math.floor(curr + _h) <= _maxEl):
    firstEl  = curr                         # first element of the current interval
```

```
    secondEl = round(curr + _h, PRECISION) # second element of the current interval
    tableRanges.append((firstEl, secondEl))

    midRange = round((firstEl + secondEl)/2, PRECISION) # middle of the interval
    midRanges.append(midRange)

    frequency = findFrequency(firstEl, secondEl, _data) # frequency
    frequencies.append(frequency)

    relativeFrequency = round(frequency / _n, PRECISION) # relative frequency
    relativeFrequencies.append(relativeFrequency)

    curr = round(curr + _h, PRECISION)
```

Листинг 7: Оформление данных в виде таблицы

```
_intervalRange = pd.DataFrame({'номер интервала':       intervals,
                               'интервал':              tableRanges,
                               'середина интервала':    midRanges,
                               'частота':               frequencies,
                               'относительная частота': relativeFrequencies})

_intervalRange
```

Листинг 8: Поиск границ интервалов

```
# interval boundaries
_int1 = midRanges - _h/2
_int1 = np.append(_int1, _maxEl)

print(f'границы интервалов: {_int1}')
```

3. По сгруппированным данным постройте гистограмму относительных частот

```
def buildLine(x, y, color):
    plt.plot(x, y, color=color, linestyle='-', linewidth=1.5)
```

Листинг 9: Построение гистограммы

```
def buildBar(x, y, overlay = None, showAxis = True):
    RED   = '#6F1D1B'
    BLUE  = '#00ADB5'
    WHITE = '#EEEEEE'
    BLACK = '#393E46' # #253031 #0D1B1E #1C2321

    # Define font sizes
    SIZE_DEFAULT = 14
    SIZE_LARGE   = 16
    SIZE_TICKS   = 10
    plt.rc("font", weight="normal")        # controls default font
    plt.rc("font", size=SIZE_DEFAULT)      # controls default text sizes
    plt.rc("axes", titlesize=SIZE_LARGE)   # fontsize of the axes title
    plt.rc("axes", labelsize=SIZE_DEFAULT) # fontsize of the x and y labels
    plt.rc("xtick", labelsize=SIZE_DEFAULT)# fontsize of the tick labels
    plt.rc("ytick", labelsize=SIZE_DEFAULT)# fontsize of the tick labels

    fig, ax = plt.subplots(
```

```
        figsize=(6, 5)
    )

    xticks = [i for i in range(math.floor(min(x)) - 1, math.ceil(max(x)) + 2)]

    # Hide the all but the bottom spines (axis lines)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_visible(False)
    ax.spines["top"].set_visible(False)

    if not showAxis:
        ax.spines["bottom"].set_visible(False)

    # Only show ticks on the left and bottom spines
    ax.yaxis.set_ticks_position("left")
    ax.xaxis.set_ticks_position("bottom")
    ax.spines["bottom"].set_bounds(min(xticks), max(xticks))

    # histogtamm
    plt.bar(x, y, width=2.25, color='white', edgecolor='black',
            linewidth=1.5, align='center')

    # line
    if overlay is None:
        buildLine(x, y, RED)
    else:
        xmin, xmax = plt.xlim()
        x = np.linspace(xmin, xmax, 100)
        calculate_y = overlay(x)
        plt.plot(x, calculate_y, color=BLUE, linestyle='-', linewidth=1.5)

    # axis names
    if showAxis:
        plt.xlabel('int')
        plt.ylabel('$\\frac{p_k}{h}$', fontsize=20)
        plt.xticks(xticks)
    else:
        ax.xaxis.set_ticks_position('none')
        ax.yaxis.set_ticks_position('none')
        ax.set_xticks([])
        ax.set_yticks([])

    # Adjust the font size of the tick labels
    plt.tick_params(axis='both', which='major', labelsize=SIZE_TICKS)

    plt.show()

def overlayPlot(overlay):
    BLUE  = '#00ADB5'

    plt.hist(_data, bins=int(_k), color='white', edgecolor='black',
             density=True, alpha=0.6)

    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)

    calculate_y, label = overlay(x)
    plt.plot(x, calculate_y, color=BLUE, linestyle='-', linewidth=2, label=label)

    plt.legend()
    plt.show()
```

```
# is indicated along the ordinate axis when constructing a histogram
_histogrammOrdinateAxis = relativeFrequencies / _h

print(f'x: {midRanges}')
print(f'y: {_histogrammOrdinateAxis}')

buildBar(midRanges, _histogrammOrdinateAxis)

plt.hist(_data, bins=int(_k))
plt.show()
```

4. Вычислите выборочное среднее и выборочную дисперсию

Листинг 10: поиск выборочного среднего и выборочной дисперсии

```
# sample average
_overlineX = (1 / _n) * np.sum(_data)

# sample variance
_S2 = 1 / (_n - 1) * np.sum((_data - _overlineX)**2)

print(f'выборочное среднее: {_overlineX}')
print(f'выборочная дисперсия: {_S2}')
```

5. По виду гистограммы определите возможный закон распределения, оцените параметры этого закона по методу моментов, постройте совмещенные графики гистограммы и плотности предполагаемого закона

Листинг 11: Анализ и вывод полученных данных в ходе теста Колмогорова-Смирнова

```
def align(name, enName, kstestRes, *params):
    def line(n):
        display('-'*n)

    heading: str = f"\\begin{{equation}} " \
                   f"   \\text{{{name}}} " \
                   f"\\end{{equation}}   " \

    dataDisplay: str = f"\\begin{{align}}\n"

    NAME   = 0
    SYMBOL = 1
    VALUE  = 2
    for param in params:
        dataDisplay += f"& \\text{{{param[NAME]}}}, "\
                       f"{param[SYMBOL]}: & \\quad {param[VALUE]} \\\\\n"

    dataDisplay += f"\\end{{align}}"

    kstestHeading = \
    f"\\begin{{equation}} "                                  \
    f"   \\textрезультаты{{ тестаКолмогороваСмирнова -}}: " \
    f"\\end{{equation}} "
```

```
statistic          = kstestRes.statistic
pvalue             = kstestRes.pvalue
statistic_location = kstestRes.statistic_location
statistic_sign     = kstestRes.statistic_sign

statisticDisplay = f'максимальное различиемеждуэмпирическойи     '       \
                   f'теоретической функциямираспределениясоставляет    ' \
                   f'примерно: {int(np.round(statistic, 2) * 100)}\\%'

pvalueFlag = True if pvalue < 0.05 else False

pvalueDisplay = f'значение уровня- значимости: {pvalue} '
pvalueDisplay += '<' if pvalueFlag else '>'
pvalueDisplay += f' 0.05'

if pvalueFlag:
    pvalueDisplay += \
    f' данные( несоответствуютпредполагаемомураспределению    .)'
else:
    pvalueDisplay += \
    f' на( уровнезначимости  0.05 нетоснованийотвергнутьгипотезу     ' \
    f'о соответствииданныхвыбранномураспределению    )'

statistic_locationDisplay = \
f'максимальное отклонениенаблюдаетсяпризначенииданных     , ' \
f'равном {statistic_location}'

statistic_signDisplay = \
f'в точкемаксимальногоотклоненияэмпирическаяфункцияраспределения       '
statistic_signDisplay += 'ниже' if statistic_sign < 0 else 'выше'
statistic_signDisplay += ', чемтеоретическая '

kstestResDisplay = [statisticDisplay,
                    pvalueDisplay,
                    statistic_locationDisplay,
                    statistic_signDisplay]

kstestDisplay  = f"\\begin{{align}}\n"

for res in kstestResDisplay:
    kstestDisplay += f"& \\text{{{res}}} \\\\\n"

kstestDisplay += f"\\end{{align}}"

line(200)

display(Math(heading))
display(Math(dataDisplay))
display(Math(kstestHeading))
display(Math(kstestDisplay))

match enName:
    case 'norm':
        def buildLaw(x):
            return sp.stats.norm.pdf(
                        x,
                        np.mean(_data),
                        np.std(_data, ddof=1)
                    ), name
```

```
case 'chi2':
    def buildLaw(x):
        df_chi2 = 2
        return sp.stats.chi2.pdf(
                    x,
                    df_chi2,
                    loc=np.min(_data),
                    scale=np.std(_data)
                ), name + f'(df={df_chi2})'

case 'expon':
    def buildLaw(x):
        return sp.stats.expon.pdf(
                    x,
                    loc=np.min(_data),
                    scale=np.mean(_data)-np.min(_data)
                ), name

case 'gamma':
    def buildLaw(x):
        shape_gamma = 2
        return sp.stats.gamma.pdf(
                    x,
                    shape_gamma,
                    loc=np.min(_data),
                    scale=np.std(_data)
                ), name + f'(shape={shape_gamma})'

case 'poisson':
    def buildLaw(x):
        lambda_poisson = np.mean(_data)
        return sp.stats.poisson.pmf(
                    np.floor(x),
                    lambda_poisson
                ), name

case 'uniform':
    def buildLaw(x):
        return sp.stats.uniform.pdf(
                    x,
                    loc=np.min(_data),
                    scale=np.ptp(_data)
                ), name

case 't':
    def buildLaw(x):
        df_t = 2
        return sp.stats.t.pdf(
                    x,
                    df_t,
                    loc=np.mean(_data),
                    scale=np.std(_data)
                ), name + f'(df={np.round(df_t, PRECISION)})'

case 'lognorm':
    def buildLaw(x):
        shape_lognorm = np.std(np.log(_data))
        return sp.stats.lognorm.pdf(
                    x,
                    shape_lognorm,
```

```
                            loc=np.min(_data),
                            scale=np.exp(np.mean(np.log(_data)))
                        ), name

            case 'beta':
                def buildLaw(x):
                    a_beta = params[0][VALUE]
                    b_beta = params[1][VALUE]
                    return sp.stats.beta.pdf(
                            (x - np.min(_data)) / np.ptp(_data),
                            a_beta,
                            b_beta
                        ) / np.ptp(_data), \
                        name + f'(a={np.round(a_beta, PRECISION)},'\
                               f'\n b={np.round(b_beta, PRECISION)})'

            case 'weibull_min':
                def buildLaw(x):
                    shape_weibull = params[0][VALUE]
                    return sp.stats.weibull_min.pdf(
                            x,
                            shape_weibull,
                            loc=np.min(_data),
                            scale=np.std(_data)
                        ), \
                        name + f'(shape={np.round(shape_weibull, PRECISION)})'

            case default:
                pass

        overlayPlot(buildLaw)

    def fillArrays(*args):
        allNames               = args[2]
        allStatistic           = args[3]
        allPvalue              = args[4]
        allStatistic_location  = args[5]
        allStatistic_sign      = args[6]

        name = args[1]

        statistic          = args[0].statistic
        pvalue             = args[0].pvalue
        statistic_location = args[0].statistic_location
        statistic_sign     = args[0].statistic_sign

        arrays = [allNames,
                  allStatistic,
                  allPvalue,
                  allStatistic_location,
                  allStatistic_sign]
        data   = [name,
                  statistic,
                  pvalue,
                  statistic_location,
                  statistic_sign]

        for ind, el in enumerate(data):
            arrays[ind].append(el)
```

Листинг 12: Проведение теста Колмогорова-Смирнова на различных законах

```
data = np.array(_data)

allNames              = []
allStatistic          = []
allPvalue             = []
allStatistic_location = []
allStatistic_sign     = []

# Normal distribution (norm)
mean = np.mean(_data)
std = np.std(_data, ddof=1)
result = sp.stats.kstest(data, 'norm', args=(mean, std))
fillArrays(result,
           'norm',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('нормальное', 'norm', result, ['среднее', '\\mu', mean],
                                    ['стандратное отклонение', '\\sigma', std])

# Chi-square distribution (chi2)
df = 2
loc = np.min(_data)
scale = np.std(_data)
result = sp.stats.kstest(data, 'chi2', args=(df, loc, scale))
fillArrays(result,
           'chi2',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('хиквадрат-', 'chi2', result, ['количество степенейсвободы ', '', df],
                                    ['сдвиг ', '', loc],
                                    ['параметр масштаба', '', scale])

# Exponential distribution (expon)
loc = np.min(_data)
scale = np.mean(_data) - np.min(_data)
result = sp.stats.kstest(data, 'expon', args=(loc, scale))
fillArrays(result,
           'expon',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('экспоненциальное', 'expon', result, ['сдвиг', '', loc],
                                           ['параметр масштаба',
                                           '\\frac{1}{\\lambda}',
                                           scale])

# Gamma distribution (gamma)
shape = 2
loc = np.min(_data)
scale = np.std(_data)
result = sp.stats.kstest(data, 'gamma', args=(shape, loc, scale))
```

```
fillArrays(result,
           'gamma',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('гаммараспределение-', 'gamma', result, ['параметр формы', '', shape],
                                                ['сдвиг ', '', loc],
                                                ['параметр масштаба', '', scale])

# Poisson distribution (poisson)
lambda_ = np.mean(data)
result = sp.stats.kstest(data, 'poisson', args=(lambda_,))
fillArrays(result,
           'poisson',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('пуассоновское', 'poisson', result, ['параметр интенсивности', '\\mu', lambda_])

# Uniform distribution (uniform)
loc = np.min(_data)
scale = np.ptp(_data)
result = sp.stats.kstest(data, 'uniform', args=(loc, scale))
fillArrays(result,
           'uniform',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('равномерное', 'uniform', result, ['нижняя граница', '', loc],
                                        ['размах', '', scale])

# Student's t-distribution (t)
df = 2
loc = np.mean(_data)
scale = np.std(_data)
result = sp.stats.kstest(data, 't', args=(df, loc, scale))
fillArrays(result,
           't',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('tраспределение- Стьюдента', 't', result, ['количество степенейсвободы
', '', df],
                                                 ['сдвиг', '', loc],
                                                 ['масштаб', '', scale])

# Lognormal distribution (lognorm)
shape = np.std(np.log(_data))
loc = np.min(_data)
scale = np.exp(np.mean(np.log(_data)))
result = sp.stats.kstest(data, 'lognorm', args=(shape, loc, scale))
fillArrays(result,
```

```
            'lognorm',
            allNames,
            allStatistic,
            allPvalue,
            allStatistic_location,
            allStatistic_sign)
align('Логнормальное', 'lognorm', result, ['параметр формы', '', shape],
                                          ['сдвиг', '', loc],
                                          ['параметр масштаба', '', scale])

# Beta distribution (beta)
a, b, loc, scale = sp.stats.beta.fit(data)
result = sp.stats.kstest(data, 'beta', args=(a, b, loc, scale))
fillArrays(result,
            'beta',
            allNames,
            allStatistic,
            allPvalue,
            allStatistic_location,
            allStatistic_sign)
align('бетараспределение-', 'beta', result, ['параметр формы 1', '', a],
                                            ['параметр формы 2', '', b],
                                            ['сдвиг', '', loc],
                                            ['масштаб', '', scale])

# Weibull distribution (weibull_min)
c, loc, scale = sp.stats.weibull_min.fit(data)
loc = np.min(_data)
scale = np.std(_data)
result = sp.stats.kstest(data, 'weibull_min', args=(c, loc, scale))
fillArrays(result,
            'weibull_min',
            allNames,
            allStatistic,
            allPvalue,
            allStatistic_location,
            allStatistic_sign)
align('вейбулловское ', 'weibull_min', result, ['параметр формы', '', c],
                                              ['сдвиг', '', loc],
                                              ['масштаб', '', scale])


_kstestData = pd.DataFrame({
    'name': allNames,
    'statistic': allStatistic,
    'pvalue': allPvalue,
    'statistic_location': allStatistic_location,
    'statistic_sign': allStatistic_sign
})

_kstestData

# pvalue - If the pvalue is less than a predetermined significance level
# (for example, 0.05), then the null hypothesis is rejected, which means
# that the data does not fit the expected distribution.
_kstestData_filtered = _kstestData[_kstestData['pvalue'] > 0.05]
_kstestData_filtered

# statistic - The larger the value, the greater the difference between the
# data and the theoretical distribution.
_kstestData_filtered.sort_values(by='statistic')
```

Листинг 13: Построение графиков плотностей наиболее подходящих законов

```
# Calculate sampling parameters
mean = np.mean(_data)
std = np.std(_data, ddof=1)

# Sample histogram
plt.hist(_data, bins=7, density=True, alpha=0.6, color='g')

xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)

# Check for normal distribution
p = sp.stats.norm.pdf(x, mean, std)
plt.plot(x, p, 'k', linewidth=2, label='Normal')

# Check for uniform distribution
uniform_fit = sp.stats.uniform.pdf(
                    x,
                    loc=np.min(_data),
                    scale=np.ptp(_data)
                )
plt.plot(x, uniform_fit, 'b', linewidth=2, label='Uniform')

# Check for beta distribution
a_beta, b_beta = 1.2406801559765772, 0.9441200264826078  # Параметрыформы
beta_fit = sp.stats.beta.pdf(
                (x - np.min(_data)) / np.ptp(_data),
                a_beta,
                b_beta
            ) / np.ptp(_data)
plt.plot(x, beta_fit, 'cyan', linewidth=2,
                              label='Beta (a=1.24068, b=0.94412)')

# Add a legend and display the graph
plt.legend()
plt.show()

_a = _overlineX - np.sqrt(3 * _S2)
_b = _overlineX + np.sqrt(3 * _S2)

print(f'a: {_a}')
print(f'b: {_b}')
```

6. Найдите эмпирическую функцию распределения и постройте совмещенные графики эмпирической и теоретической функций распределения

```
# accumulated frequency column
_kum = np.zeros(np.size(relativeFrequencies) + 1)
ind = 1
for relativeFrequency in relativeFrequencies:
    _kum[ind] = _kum[ind-1] + relativeFrequencies[ind-1]
    ind += 1
print(f'kum: {_kum}')
```

Листинг 14: эмпирическая функция распределения femp(x)

```
def femp(x):
```

```
    def ind(x):
        return 1 if x > 0 else 0

    sumy = 0
    for i in range(int(_k)):
        sumy += relativeFrequencies[i] * ind(x - midRanges[i])

    return sumy
```

Листинг 15: функция построения совмещенных femp(x)

```
def buildFemp(cdf_y_values):
    RED   = '#6F1D1B'
    BLUE  = '#00ADB5'
    WHITE = '#EEEEEE'
    BLACK = '#393E46' # #253031 #0D1B1E #1C2321

    # Define font sizes
    SIZE_DEFAULT = 14
    SIZE_LARGE   = 16
    SIZE_TICKS   = 10
    plt.rc("font", weight="normal")        # controls default font
    plt.rc("font", size=SIZE_DEFAULT)      # controls default text sizes
    plt.rc("axes", titlesize=SIZE_LARGE)   # fontsize of the axes title
    plt.rc("axes", labelsize=SIZE_DEFAULT) # fontsize of the x and y labels
    plt.rc("xtick", labelsize=SIZE_DEFAULT)# fontsize of the tick labels
    plt.rc("ytick", labelsize=SIZE_DEFAULT)# fontsize of the tick labels

    fig, ax = plt.subplots(
        figsize=(6, 5)
    )

    # Generate a range of x values
    x_values = np.linspace(0, np.max(_data) + np.min(_data), 100)

    # Evaluate the function for each x value
    femp_y_values = [femp(x) for x in x_values]

    cdf_y_values = cdf_y_values(x_values)

    xticks = [i for i in range(0, int(np.max(_data) + np.min(_data)) + 1)]
    yticks = np.arange(0, 1.2 + 0.1, 0.1)

    # Hide the all but the bottom spines (axis lines)
    ax.spines["right"].set_visible(False)
    ax.spines["left"].set_visible(False)
    ax.spines["top"].set_visible(False)

    # Only show ticks on the left and bottom spines
    ax.yaxis.set_ticks_position("left")
    ax.xaxis.set_ticks_position("bottom")
    ax.spines["bottom"].set_bounds(min(xticks), max(xticks))

    # Plot femp(x)
    plt.plot(x_values, femp_y_values, label='femp(x)', color=RED)

    # Plot the cumulative distribution function
    plt.plot(x_values, cdf_y_values, label='CDF(x)', color='black')

    # plot y = 1 line
    plt.plot(x_values, np.full_like(x, 1), label='y = 1',
```

```
            linestyle='--', color='black')

    # axis names
    plt.xlabel('x')
    plt.ylabel('F(x)')

    plt.xticks(xticks)
    plt.yticks(yticks)

    # Adjust the font size of the tick labels
    plt.tick_params(axis='both', which='major', labelsize=SIZE_TICKS)

    plt.grid(True)

    plt.show()
```

Листинг 16: нахождение значений y для uniform

```
# uniform
def uniform_y_values(x_values):
    return [sp.stats.uniform.cdf(
                x,
                loc=_a,
                scale=_b - _a
            ) for x in x_values]

buildFemp(uniform_y_values)
```

Листинг 17: нахождение значений y для norm

```
# norm
def norm_y_values(x_values):
    mean, std = sp.stats.norm.fit(data)
    return [sp.stats.norm.cdf(x, mean, std) for x in x_values]

buildFemp(norm_y_values)

display(Math(f'\\mu: {mean}'))
display(Math(f'\\sigma^2: {std**2}'))
```

Листинг 18: нахождение значений y для beta

```
# beta
def beta_y_values(x_values):
    a, b, loc, scale = sp.stats.beta.fit(data)
    return [sp.stats.beta.cdf(x, a, b, loc, scale) for x in x_values]

buildFemp(beta_y_values)

display(Math(f'\\alpha: {a}'))
display(Math(f'\\beta: {b}'))
print(f'сдвиг (loc): {loc}')
print(f'масштаб (scale): {scale}')
```