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 **1**

2 Исходные данные **1**

3 Решение **2**

3.1 Найдите крайние члены вариационного ряда и размах выборки 2

4 Приложение **2**

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

3.1 Найдите крайние члены вариационного ряда и размах выборки

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.

```
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)

_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
]

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

_parsedData
```

1. Крайние члены вариационного ряда и размах 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

```
# 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}")

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

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

    return count
```

```
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) \
        if not (abs(curr + _h - _maxEl) < 0.1)          \
        else findFrequency(firstEl, secondEl, _data, True) # frequency
    frequencies.append(frequency)

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

    curr = round(curr + _h, PRECISION)

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

_intervalRange

# 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)

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

```
# 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

```
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 руровня- 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 отклонениенаблюдаетсяпризначениидanных     , ' \
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 'pareto':
                def buildLaw(x):
                    b_pareto = 0.1
                    return sp.stats.pareto.pdf(
                            x - np.min(_data) + 1,
                            b_pareto
                        ), name + f'(shape={b_pareto})'

            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)

data = np.array(_data)

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

# Normal distribution (norm)
mean, std = sp.stats.norm.fit(data)
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, loc, scale = sp.stats.chi2.fit(data)
result = sp.stats.kstest(data, 'chi2', args=(df,))
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, scale = sp.stats.expon.fit(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, loc, scale = sp.stats.gamma.fit(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, scale = sp.stats.uniform.fit(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, loc, scale = sp.stats.t.fit(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, loc, scale = sp.stats.lognorm.fit(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)
fillArrays(result,
           'weibull_min',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('вейбулловское ', 'weibull_min', result, ['параметр формы', '', c],
                                                ['сдвиг', '', loc],
                                                ['масштаб', '', scale])

# Pareto distribution (pareto)
b, loc, scale = sp.stats.pareto.fit(data)
result = sp.stats.kstest(data, 'pareto', args=(b, loc, scale))
fillArrays(result,
           'pareto',
           allNames,
           allStatistic,
           allPvalue,
           allStatistic_location,
           allStatistic_sign)
align('парето ', 'pareto', result, ['параметр формы', '', b],
                                   ['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')

# 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()

_param = 2 * _overlineX

_param
```

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}')

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

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()

# uniform
def uniform_y_values(x_values):
    return [sp.stats.uniform.cdf(
                x,
                loc=np.min(_data),
                scale=np.ptp(_data)
            ) for x in x_values]

buildFemp(uniform_y_values)

# 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)

# 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)
```