



«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 № 7

Название лабораторной работы:

Критерий согласия для проверки простой
непараметрической гипотезы

Вариант № 9

Дисциплина:

Теория вероятности и математическая статистика

Студент группы ФН11-52Б

(Подпись, дата)

Очкин Н.В.

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

Облакова Т.В.

(И.О. Фамилия)

Задание

Постройте с помощью стохастического эксперимента на основе указанной метрики приближенный критерий для проверки основной гипотезы. Найдите критические значения $D_{\text{кр}}$ для трех уровней значимости $\alpha = 0.1, 0.05$ и 0.01 .

Протестируйте критерий на трех-четырех примерах и сформулируйте выводы.

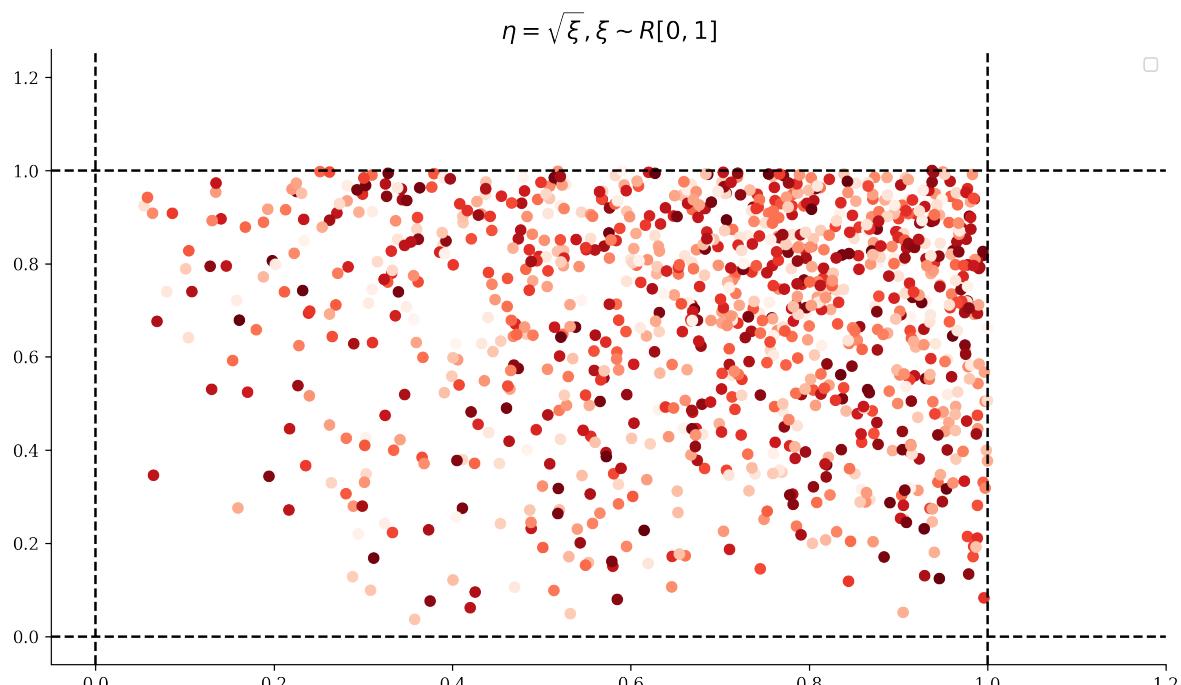
Исходные данные

$$A = 1 \quad D = 4 \quad n = 1000$$

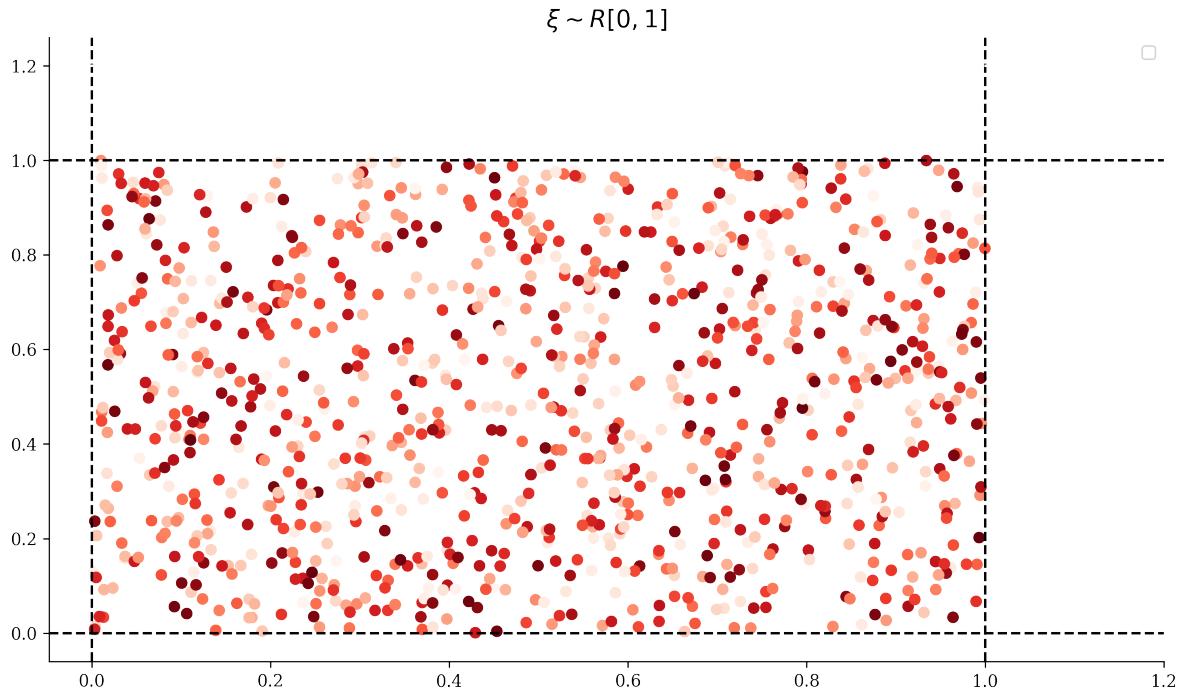
$$\eta = \sqrt{\xi}, \quad \xi \sim R[0, 1] \quad D = \sum_j \frac{|\nu_j - np_j|}{np_j}$$

Ход выполнения работы

Для начала рассмотрим распределения, с которыми работаем. Для этого сгенерируем $1e3$ случайных величин для осей x и y для заданного и равномерного распределений (для сравнения).



(Цвет не несет информации о каких-либо свойствах распределения и используется исключительно для улучшения визуального восприятия.)



(Цвет не несет информации о каких-либо свойствах распределения и используется исключительно для улучшения визуального восприятия.)

Также посмотрим на функции плотности вероятности (далее pdf) и функции распределения (далее cdf).

pdf и cdf для непрерывного равномерного закона на отрезке $[0, 1]$ известны и равны:

$$f_X(x) = \begin{cases} 1, & x \in [0, 1] \\ 0, & x \notin [0, 1] \end{cases} \quad F_X(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x < 1 \\ 1, & x \geq 1 \end{cases}$$

Для нахождения pdf и cdf для случайной величины $\eta = \sqrt{\xi}$, $\xi \sim R[0, 1]$, воспользуемся формулами[1]:

$$p_\eta(x) = \frac{1}{g'(g^{-1}(x))} p_\xi(g^{-1}(x)) \quad (1)$$

$$F_\eta(x) = P\{\eta \leq x\} = P\{g(\xi) \leq x\} = P\{\xi \leq g^{-1}(x)\} = F_\xi(g^{-1}(x)) \quad (2)$$

Программно реализуем данные формулы:

```
def g(x):
    match A_:
        case 1:
            return np.sqrt(x)
        case 2:
```

```

        return 1 - np.sqrt(x)
    case _:
        raise ValueError("Invalid value for A")

def inverseFunction(y):
    def equation_to_solve(x):
        return g(x) - y

sol = None
guesses = np.arange(a_, b_ + (b_ - a_)/10, (b_ - a_)/10)
for guess in guesses:
    try:
        sol, = sp.optimize.fsolve(equation_to_solve, guess)
        break
    except:
        continue

if sol is not None:
    return sol
else:
    raise Exception('solution was not found')

def pdf_(x):
    if x < a_ or x > b_:
        return 0

def numericalDerivative(f, x):
    return cdm_.diff(f=f, x=x)

inverse_x = inverseFunction(x)
return 1 / numericalDerivative(g, inverse_x) * sp.stats.uniform.pdf(inverse_x)

def cdf_(x):
    return sp.stats.uniform.cdf(inverseFunction(x))

```

Где для дифференцирования используется метод центральных разностей:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Погрешность определяется как $O(h)$, h примем равной 1e-6.

```

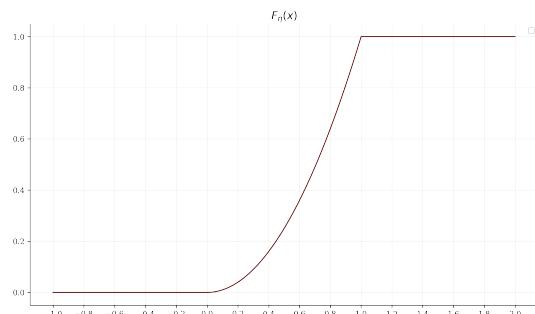
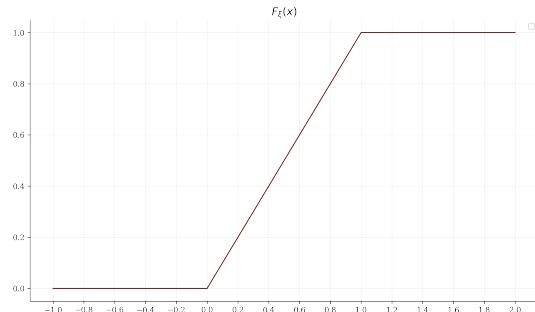
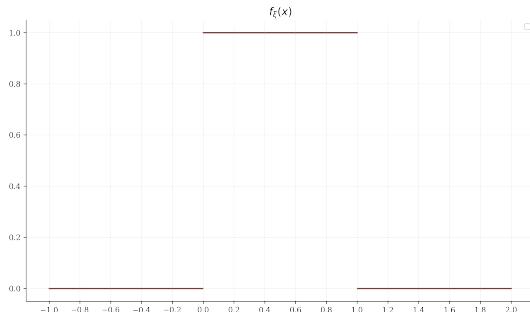
class CDM:
    def __init__(self, h):
        self.h = h

    def diff(self, f, x):
        numerator = f(x + self.h) - f(x - self.h)
        denominator = 2 * self.h

        return numerator / denominator

```

Наконец построим графики найденных pdf и cdf:



Перейдем к стохастическому эксперименту.

Напишем вспомогательную функцию для генерации выборки, в соответствии с вариантом:

```
def generateValue(n=1):
    match A_:
        case 0:
            return sp.stats.uniform.rvs(size=n)
        case 1:
            return np.sqrt(sp.stats.uniform.rvs(size=n))
        case 2:
            return 1 - np.sqrt(sp.stats.uniform.rvs(size=n))
```

Теперь смоделируем выборку, подчиняющуюся основной гипотезе $m = 1e4$ раз:

```
| data_ = [generateValue(n_) for _ in range(m_)]
```

Сгруппируем каждую из полученных выборок.

Для этого создадим вспомогательный класс SAMPLE, который будет хранить следующие параметры как поля класса:

размер выборки ($_n$), крайние члены ($_min$ и $_max$), размах выборки ($_range$), количество интервалов ($_l$), шаг интервала ($_h$), границы интервалов ($_int_boundaries$), интервалы ($_intervals$), середины интервалов ($_mid_ranges$), частоты ($_freqs$), относительные частоты ($_rel_freqs$) и плотность относительных

частот (*_rel_freqs_density*).

Следующие поля сделаем статическими, тк они остаются постоянными вне зависимости от выборки:

_n, _min, _max, _range, _l, _h, _int_boundaries, _intervals, _mid_ranges.

Которые инициализируем в отдельном методе:

```
@classmethod
def _initializeStaticFields(cls):
    cls._n      = n_
    cls._min    = a_
    cls._max    = b_
    cls._range = cls._max - cls._min
    cls._l     = 1 + int(np.log2(cls._n))
    cls._h     = cls._range / cls._l

    cls._int_boundaries = np.array(
        [cls._min + i * cls._h for i in range(0, cls._l + 1, 1)])
    )
    cls._intervals = np.array(
        [(cls._int_boundaries[i],
          cls._int_boundaries[i+1]) for i in range(0, cls._l, 1)])
    )
    cls._mid_ranges = np.array(
        [sum(interval)/2 for interval in cls._intervals]
    )
```

Также отдельно напишем обработку частот, так как они уже будут разниться у каждой выборки:

```
def _countFrequencies(self, data):
    present = lambda el, int_ : int_[0] <= el < int_[1]
    freqs_ = np.zeros(Sample._l)
    for el in data:
        for j in range(0, Sample._l, 1):
            if present(el, Sample._intervals[j]):
                freqs_[j] += 1
    freqs_[-1] += np.count_nonzero(data == Sample._max)

    self._freqs = freqs_
    self._postCountFrequencies()

def _postCountFrequencies(self):
    if self._freqs is not None:
        rel_freqs_ = self._freqs / Sample._n

        self._rel_freqs = rel_freqs_

        rel_freqs_density_ = self._rel_freqs / Sample._h
        self._rel_freqs_density = rel_freqs_density_
```

Наконец сгруппируем каждую из m сгенерированных выборок:

```
| groupedSamples = [Sample(sample) for sample in data_]
```

Выведем информацию о полученной группировке для одной из выборок.

Данная информация присуща всем выборкам:

```
n : 1000      min : 0      max : 1      range : 1      l : 10      h : 0.1  
границы интервалов : [0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1]  
интервалы : [0  0.1),  [0.1  0.2),  [0.2  0.3),  [0.3  0.4),  [0.4  0.5),  
[0.5  0.6),  [0.6  0.7),  [0.7  0.8),  [0.8  0.9),  [0.9  0.1]  
середины интервалов : [0.05  0.15  0.25  0.35  0.45  0.55  0.65  0.75  0.85  0.95]
```

Данная информация присуща только конкретной (выбранной для демонстрации) выборке:

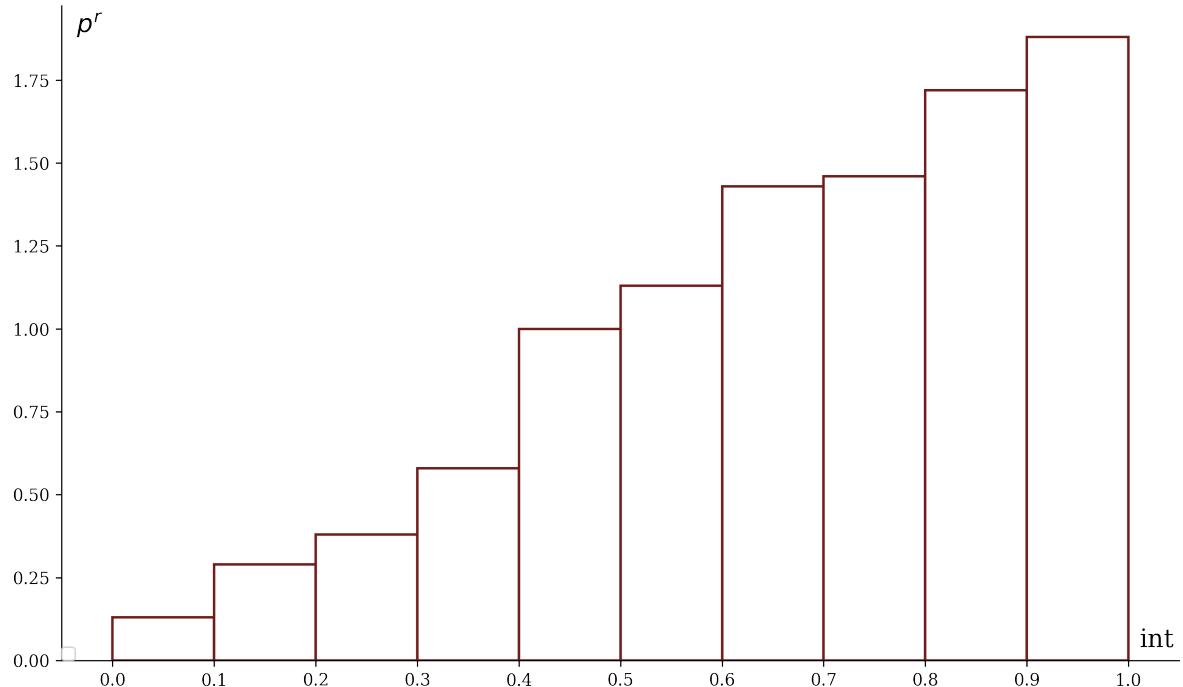
```
частоты : [11  36  49  73  92  116  117  146  161  199]  
относительные частоты : [0.011  0.036  0.049  0.073  0.092  0.116  0.117  0.146  0.161  0.199]  
плотность относительных частот : [0.11  0.36  0.49  0.73  0.92  1.16  1.17  1.46  1.61  1.99]
```

Добавим еще один метод в класс SAMPLE для отрисовки гистограммы относительных частот:

```
def _hist(self, filename):  
    RED = '#6F1D1B'  
  
    _, ax = plt.subplots(figsize=(10, 6))  
  
    x_values = Sample._mid_ranges  
    y_values = self._rel_freqs_density  
  
    ax.bar(x_values,  
            y_values,  
            width=Sample._h,  
            color='white',  
            edgecolor=RED,  
            linestyle='-',  
            linewidth=1.5,  
            align='center')  
  
    decorate_plot(ax, Sample._int_boundaries, 'int', '$p^r$', loc=(0, 0))  
  
    if SAVE_PLOTS:  
        plt.savefig(f'{filename}.png', dpi=300, transparent=True)
```

```
plt.show()
```

Гистограмма, выбранной для демонстрации выборки:



Посмотрим также гистограммы первых 3бти выборок после демонстрационной:

Теперь перейдем к вычислению статистики $D_k(n, l)$.

Напишем вспомогательную функцию для выбора корректной метрики, в соответствии с вариантом:

```
def metric(nu, p):
    nu = np.array(nu)
    p = np.array(p)
    match D_:
        case 1:
            return 1/n_ * np.max(np.abs(nu - n_ * p))
        case 2:
            return 1/n_ * np.sum(np.abs(nu - n_ * p))
        case 3:
            return 1/n_ * np.sqrt(np.sum((nu - n_ * p)**2))
        case 4:
            return np.sum((np.abs(nu - n_ * p))/(n_ * p))
        case 5:
            return np.sum((nu - n_ * p)**2/(n_ * p * (n_ - n_ * p)))
```

Количество значений, попавших в j -ый интервал группировки (ν) нам уже известно и хранится в поле $_freqs$ для каждого объекта класса SAMPLE.

Теоретическую вероятность попадания в j -ый интервал группировки найдем в соответствии с (2):

```
theorIntHitProbs_ = [] # p_j
theorIntHitProbsN_ = [] # n*p_j

for interval in Sample._intervals:
    beg = interval[0]
    end = interval[1]

    theorIntHitProb = cdf_(end) - cdf_(beg)
    theorIntHitProbs_.append(theorIntHitProb)

theorIntHitProbsN_.append(Sample._n * theorIntHitProb)
```

```
p : [0.01 0.03 0.05 0.07 0.09 0.11 0.13 0.15 0.17 0.19]
n · p : [10.0 30.0 50.0 70.0 90.0 110.0 130.0 150.0 170.0 190.0]
```

Наконец вычислим статистики:

```

D_arr = []
for groupedSample in groupedSamples:
    nu = groupedSample._freqs
    D_curr = metric(nu, theorIntHitProbs_)

    D_arr.append(D_curr)

```

Выведем первые и последние две статистики в отсортированном массиве статистик всех выборок:

```
[0.235165    0.237748    ...    2.278528    2.2851]
```

Теперь приближенно оценим квантили уровней $\alpha = [0.1 \ 0.05 \ 0.01]$:

```

alphas_ = [0.1, 0.05, 0.01]
quantiles_ = np.quantile(D_arr, [1 - alpha for alpha in alphas_],
                         method='inverted_cdf')
quantiles_

```

Получим:

α	0.1	0.05	0.01
D_{cr}	1.33	1.46	1.7

Напишем еще одну вспомогательную функцию для тестирования построенного критерия:

```

def testSample(sample):
    groupedSample = Sample(sample)
    nu = groupedSample._freqs
    D_curr = metric(nu, theorIntHitProbs_)

    accepted = 0
    accepted_sent = []
    for i, quantile in enumerate(quantiles_):
        if D_curr < quantile:
            accepted_text = f'{D_curr} < {quantile} for alpha: {alphas_[i]} => accept'
            accepted_sent.append(accepted_text)
            accepted += 1

    if accepted == 3:
        print(f'{D_curr} < D for all alphas => accept')
    elif accepted > 0:
        print(accepted_sent)
    else:
        print(f'{D_curr} > D for all alphas => decline')

```

И запустим на нескольких законах распределения:

- Uniform: $D \approx 14.226 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- $\eta = \sqrt{\xi}, \ \xi \sim R[0, 1]: D \approx 1.203 < D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза принимается во всех случаях
- $\eta = 1 - \sqrt{\xi}, \ \xi \sim R[0, 1]: D \approx 30.992 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- Beta: $D \approx 26.130 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- Triangular: $D \approx 7.152 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- Exponential (truncated to $[0, 1]$): $D \approx 21.886 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- Lognormal (truncated to $[0, 1]$): $D \approx 6.101 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях
- Weibull (truncated to $[0, 1]$): $D \approx 6.590 > D_{cr} \forall \alpha \in [0.1 \ 0.05 \ 0.01] \Rightarrow$ гипотеза отклоняется во всех случаях

Вывод

На основе стохастического эксперимента был построен критерий согласия для проверки простой гипотезы. Критерий был проверен на семи выборках распределений не в соответствие с основной гипотезой, и для каждой из них основную гипотезу отклонил, что говорит в пользу критерия. Кроме того, критерий был проверен на выборке, распределенной в соответствие с основной гипотезой. Для этого случая была получена статистика, меньшая каждого из уровней доверия, и основная гипотеза была принята, что также говорит в пользу критерия.

Приложение

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

```
import numpy as np
import scipy as sp
from dataclasses import dataclass, field
from typing import List, ClassVar
from IPython.display import Math, display
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

EXTRA_SAMPLE_SIZE = 1000
SAVE_PLOTS = False
FINITE_DIFFERENCE_APPROXIMATION_STEP_SIZE = 1e-6

A_ = 1
D_ = 4
n_ = 1000
m_ = 1e4

def generateValue(n=1):
    match A_:
        case 0:
            return sp.stats.uniform.rvs(size=n)
        case 1:
            return np.sqrt(sp.stats.uniform.rvs(size=n))
        case 2:
            return 1 - np.sqrt(sp.stats.uniform.rvs(size=n))

def metric(nu, p):
    nu = np.array(nu)
    p = np.array(p)
    match D_:
        case 1:
            return 1/n_ * np.max(np.abs(nu - n_ * p))
        case 2:
            return 1/n_ * np.sum(np.abs(nu - n_ * p))
        case 3:
            return 1/n_ * np.sqrt(np.sum((nu - n_ * p)**2))
        case 4:
            return np.sum((np.abs(nu - n_ * p))/(n_ * p))
        case 5:
            return np.sum((nu - n_ * p)**2/(n_ * p * (n_ - n_ * p)))

m_ = int(m_)

a_ = 0
b_ = 1

def decorate_plot(ax, x_ticks, xname, yname, loc=(-0.025, -0.3)):
    SIZE_TICKS = 10
```

```

# Eliminate upper and right axes
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

# Show ticks in the left and lower axes only
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

# axis names
ax.set_xlabel(xname, fontsize=15)
ax.xaxis.set_label_coords(0.98, 0.05)

ax.set_ylabel(yname, rotation=0, fontsize=15)
ax.yaxis.set_label_coords(0.025, 0.95)

ax.set_xticks(x_ticks)

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

plt.legend(fontsize=10, loc=loc)

# Update font settings
plt.rcParams.update({'font.family': 'serif', 'font.size': 12})

# Adjust layout
plt.tight_layout()

def clean(data):
    res = []
    for el in data:
        res.append(round(el, 3))
    return res

@dataclass
class Sample:
    # static fields
    _n: ClassVar[int] = None
    _min: ClassVar[float] = None
    _max: ClassVar[float] = None
    _range: ClassVar[float] = None
    _l: ClassVar[float] = None
    _h: ClassVar[float] = None
    _int_boundaries: ClassVar[List[float]] = None
    _intervals: ClassVar[List[float]] = None
    _mid_ranges: ClassVar[List[float]] = None

    # instance fields
    _freqs: List[float] = field(default_factory=list)
    _rel_freqs: List[float] = field(default_factory=list)
    _rel_freqs_density: List[float] = field(default_factory=list)

    @classmethod
    def _initializeStaticFields(cls):

```

```

cls._n      = n_
cls._min    = a_
cls._max    = b_
cls._range  = cls._max - cls._min
cls._l      = 1 + int(np.log2(cls._n))
cls._h      = cls._range / cls._l

cls._int_boundaries = np.array(
    [cls._min + i * cls._h for i in range(0, cls._l + 1, 1)])
)
cls._intervals = np.array(
    [(cls._int_boundaries[i], cls._int_boundaries[i+1]) for i in range(0, cls._l)])
)
cls._mid_ranges = np.array(
    [sum(interval)/2 for interval in cls._intervals])
)

def _countFrequencies(self, data):
    present = lambda el, int_ : int_[0] <= el < int_[1]
    freqs_ = np.zeros(Sample._l)
    for el in data:
        for j in range(0, Sample._l, 1):
            if present(el, Sample._intervals[j]):
                freqs_[j] += 1
    freqs_[-1] += np.count_nonzero(data == Sample._max)

    self._freqs = freqs_
    self._postCountFrequencies()

def _postCountFrequencies(self):
    if self._freqs is not None:
        rel_freqs_ = self._freqs / Sample._n

        self._rel_freqs = rel_freqs_

        rel_freqs_density_ = self._rel_freqs / Sample._h
        self._rel_freqs_density = rel_freqs_density_

def _hist(self, filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    x_values = Sample._mid_ranges
    y_values = self._rel_freqs_density

    ax.bar(x_values,
           y_values,
           width=Sample._h,
           color='white',
           edgecolor=RED,
           linestyle='--',
           linewidth=1.5,
           align='center')

```

```

    decorate_plot(ax, Sample._int_boundaries, 'int', '$p^r$', loc=(0, 0))

# if SAVE_PLOTS:
plt.savefig(f'{filename}.png', dpi=300, transparent=True)

plt.show()

def __init__(self, data):
    if self._n is None:
        self._initializeStaticFields()

    self._countFrequencies(data)

def __str__(self) -> str:
    str_ = f'n: {self._n}\n'
    \
        f'min: {self._min}      max: {self._max}\n'
    \
        f'range: {self._range}\n'
    \
        f'l: {self._l}\n'
    \
        f'h: {self._h}\n'
    \
        f'interval boundaries: {self._int_boundaries}\n'
    \
        f'intervals: {self._intervals}\n'
    \
        f'intervals\' midpoints: {self._mid_ranges}\n'
    \
        f'frequencies: {self._freqs}\n'
    \
        f'relative frequencies: {self._rel_freqs}\n'
    \
        f'relative frequencies\' density: {self._rel_freqs_density}\n'

    return str_

class CDM:
    def __init__(self, h):
        self.h = h

    def diff(self, f, x):
        numerator = f(x + self.h) - f(x - self.h)
        denominator = 2 * self.h

        return numerator / denominator

cdm_ = CDM(h=FINITE_DIFFERENCE_APPROXIMATION_STEP_SIZE)

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

```

```

interval_size = b_ - a_
step = interval_size/5

x_values = [sp.stats.uniform.rvs() for _ in range(EXTRA_SAMPLE_SIZE)]
y_values = [sp.stats.uniform.rvs() for _ in range(EXTRA_SAMPLE_SIZE)]

colors = [sp.stats.uniform.rvs(0, 100) for _ in range(EXTRA_SAMPLE_SIZE)]

ax.scatter(x_values,
           y_values,
           c=colors,
           cmap='Reds',
           label='')

ax.axvline(x=b_, color='black', linestyle='--')
ax.axhline(y=b_, color='black', linestyle='--')

ax.axvline(x=a_, color='black', linestyle='--')
ax.axhline(y=a_, color='black', linestyle='--')

ax.axhline(y=b_ + step, color='white', linestyle='--')

ax.set_title(f'$\\xi \\sim R[{a_}, {b_}]$')

decorate_plot(ax, np.arange(a_, b_ + step + step, step), '', '',
              loc='best')

if SAVE_PLOTS:
    plt.savefig(f'{filename}.png', dpi=300, transparent=True)

plt.show()

buildBar('uniform_EXTRA_SAMPLE_SIZE')

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    interval_size = b_ - a_
    step = interval_size/5

    x_values = generateValue(n=EXTRA_SAMPLE_SIZE)
    y_values = generateValue(n=EXTRA_SAMPLE_SIZE)

    colors = [sp.stats.uniform.rvs(0, 100) for _ in range(EXTRA_SAMPLE_SIZE)]

    plotTitle = ''
    match A_:
        case 1:
            plotTitle = '$\\eta = \\sqrt{\\xi}, \\xi \\sim R[0, 1]$'
        case 2:
            plotTitle = '$1 - \\eta = \\sqrt{\\xi}, \\xi \\sim R[0, 1]$'

    ax.scatter(x_values,
               y_values,

```

```

        c=colors,
        cmap='Reds',
        label='')

ax.axvline(x=b_, color='black', linestyle='--')
ax.axhline(y=b_, color='black', linestyle='--')

ax.axvline(x=a_, color='black', linestyle='--')
ax.axhline(y=a_, color='black', linestyle='--')

ax.axhline(y=b_ + step, color='white', linestyle='--')

ax.set_title(plotTitle)

decorate_plot(ax, np.arange(a_, b_ + step + step, step), ' ', ' ', loc='best')

if SAVE_PLOTS:
    plt.savefig(f'{filename}.png', dpi=300, transparent=True)

plt.show()

if A_ != 0:
    buildBar('modified_uniform_EXTRA_SAMPLE_SIZE')

def g(x):
match A_:
    case 1:
        return np.sqrt(x)
    case 2:
        return 1 - np.sqrt(x)
    case _:
        raise ValueError("Invalid value for A")

def inverseFunction(y):
    def equation_to_solve(x):
        return g(x) - y

    sol = None
    guesses = np.arange(a_, b_ + (b_ - a_)/10, (b_ - a_)/10)
    for guess in guesses:
        try:
            sol, = sp.optimize.fsolve(equation_to_solve, guess)
            break
        except:
            continue

    if sol is not None:
        return sol
    else:
        raise Exception('solution was not found')

def pdf_(x):
    if x < a_ or x > b_:
        return 0

```

```

def numericalDerivative(f, x):
    return cdm_.diff(f=f, x=x)

inverse_x = inverseFunction(x)
if A_ == 2: # ???
    return np.abs(1/numericalDerivative(g, inverse_x) * sp.stats.uniform.pdf(inverse_x))
else:
    return 1 / numericalDerivative(g, inverse_x) * sp.stats.uniform.pdf(inverse_x)

def cdf_(x):
    if A_ == 2: # ???
        return 1 - sp.stats.uniform.cdf(inverseFunction(x))
    else:
        return sp.stats.uniform.cdf(inverseFunction(x))

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    interval_size = b_ - a_
    beg = a_ - (b_ - a_)
    end = b_ + (b_ - a_)
    step = interval_size/5

    x_values = np.linspace(beg, end, EXTRA_SAMPLE_SIZE)
    y_values = sp.stats.uniform.pdf(x_values)

    ax.scatter(x_values,
               y_values,
               color=RED,
               label='',
               s=1)

    plt.grid(linestyle='--', linewidth=0.25)
    ax.set_title('$f_\xi(x)$')

    decorate_plot(ax, np.arange(beg, end + step, step), ' ', ' ', loc='best')

    if SAVE_PLOTS:
        plt.savefig(f'{filename}.png', dpi=300, transparent=True)

    plt.show()

buildBar('uniform_pdf')

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    interval_size = b_ - a_
    beg = a_ - (b_ - a_)
    end = b_ + (b_ - a_)
```

```

step = interval_size/5

x_values = np.linspace(beg, end, EXTRA_SAMPLE_SIZE)
y_values = sp.stats.uniform.cdf(x_values)

ax.plot(x_values,
         y_values,
         color=RED,
         label='')

plt.grid(linestyle='--', linewidth=0.25)

ax.set_title('$F_{\xi}(x)$')

decorate_plot(ax, np.arange(beg, end + step, step), ' ', ' ', loc='best')

if SAVE_PLOTS:
    plt.savefig(f'{filename}.png', dpi=300, transparent=True)

plt.show()

buildBar('uniform_cdf')

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    interval_size = b_ - a_
    beg = a_ - (b_ - a_)
    end = b_ + (b_ - a_)
    step = interval_size/5

    x_values = np.linspace(beg, end, EXTRA_SAMPLE_SIZE * 10)
    y_values = [pdf_(x) for x in x_values]

    ax.scatter(x_values,
               y_values,
               color=RED,
               label='',
               s=1)

    plt.grid(linestyle='--', linewidth=0.25)

    ax.set_title('$f_{\eta}(x)$')

    decorate_plot(ax, np.arange(beg, end + step, step), ' ', ' ', loc='best')

    if SAVE_PLOTS:
        plt.savefig(f'{filename}.png', dpi=300, transparent=True)

    plt.show()

if A_ != 0:
    buildBar('modified_uniform_pdf')

```

```

def buildBar(filename):
    RED = '#6F1D1B'

    _, ax = plt.subplots(figsize=(10, 6))

    interval_size = b_ - a_
    beg = a_ - (b_ - a_)
    end = b_ + (b_ - a_)
    step = interval_size/5

    x_values = np.linspace(beg, end, EXTRA_SAMPLE_SIZE)
    y_values = [cdf_(x) for x in x_values]

    ax.plot(x_values,
             y_values,
             color=RED,
             label='')

    plt.grid(linestyle='--', linewidth=0.25)

    ax.set_title('$F_\eta(x)$')

    decorate_plot(ax, np.arange(beg, end + step, step), '', '',
                  loc='best')

    if SAVE_PLOTS:
        plt.savefig(f'{filename}.png', dpi=300, transparent=True)

    plt.show()

if A_ != 0:
    buildBar('modified_uniform_cdf')

data_ = [generateValue(n_) for _ in range(m_)]
groupedSamples = [Sample(sample) for sample in data_]

for sample in groupedSamples:
    print(sample)
    print('-'*200, end='\n\n')

for i, grouped_sample in enumerate(groupedSamples):
    grouped_sample._hist(f'sample{i}_hist')

theorIntHitProbs_ = [] # p_j
theorIntHitProbsN_ = [] # n*p_j

for interval in Sample._intervals:
    beg = interval[0]
    end = interval[1]

    theorIntHitProb = cdf_(end) - cdf_(beg)
    theorIntHitProbs_.append(theorIntHitProb)

    theorIntHitProbsN_.append(Sample._n * theorIntHitProb)

```

```

print(f'p_i: {clean(theorIntHitProbs_)}')
print(f'n * p_i: {clean(theorIntHitProbsN_)}')

D_arr = []
for groupedSample in groupedSamples:
    nu = groupedSample._freqs
    D_curr = metric(nu, theorIntHitProbs_)

    D_arr.append(D_curr)

sorted_D_arr = sorted(D_arr)
print(sorted_D_arr[0], sorted_D_arr[1], sorted_D_arr[-2], sorted_D_arr[-1])

alphas_ = [0.1, 0.05, 0.01]
quantiles_ = np.quantile(D_arr, [1 - alpha for alpha in alphas_], method='inverted_cdf')
quantiles_

def testSample(sample):
    groupedSample = Sample(sample)
    nu = groupedSample._freqs
    D_curr = metric(nu, theorIntHitProbs_)

    accepted = 0
    accepted_sent = []
    for i, quantile in enumerate(quantiles_):
        if D_curr < quantile:
            accepted_text = f'{D_curr} < {quantile} for alpha: {alphas_[i]} => accept'
            accepted_sent.append(accepted_text)
            accepted += 1

    if accepted == 3:
        print(f'{D_curr} < D for all alphas => accept')
    elif accepted > 0:
        print(accepted_sent)
    else:
        print(f'{D_curr} > D for all alphas => decline')

sample_uniform = sp.stats.uniform.rvs(size=n_)
display(Math(f'$\\begin{equation} \\text{{Uniform}} \\end{equation}$'))
testSample(sample_uniform)

sample_modified_uniform1 = np.sqrt(sp.stats.uniform.rvs(size=n_))
display(Math(f'$\\begin{equation} \\eta = \\sqrt{\\xi}, \\xi \\sim R[0, 1] \\end{equation}$'))
testSample(sample_modified_uniform1)

sample_modified_uniform2 = 1 - np.sqrt(sp.stats.uniform.rvs(size=n_))
display(Math(f'$\\begin{equation} \\eta = 1 - \\sqrt{\\xi}, \\xi \\sim R[0, 1] \\end{equation}$'))
testSample(sample_modified_uniform2)

alpha = 2
beta_param = 5

```

```

sample_beta = sp.stats.beta.rvs(alpha, beta_param, size=n_)
display(Math(f'${\\begin{equation}} \\text{{Beta}} {\\end{equation}}$'))
testSample(sample_beta)

c = 0.5

sample_triangular = sp.stats.triang.rvs(c, size=n_)
display(Math(f'${\\begin{equation}} \\text{{Triangular}} {\\end{equation}}$'))
testSample(sample_triangular)

lambda_param = 1.0
b = 1.0

sample_trunc_exponential = sp.stats.truncexon.rvs(b, scale=1/lambda_param, size=n_)
display(Math(f'${\\begin{equation}} \\text{{Exponential (truncated to [0, 1])}} {\\end{equation}}$'))
testSample(sample_trunc_exponential)

s = 0.5
scale = np.exp(0)

samples_lognormal = sp.stats.lognorm.rvs(s, scale=scale, size=n_)
samples_lognormal_truncated = [x for x in samples_lognormal if a_ <= x <= b_]
display(Math(f'${\\begin{equation}} \\text{{Lognormal (truncated to [0, 1])}} {\\end{equation}}$'))
testSample(samples_lognormal_truncated)

shape = 1.5
scale = 1.0

samples_weibull = sp.stats.weibull_min.rvs(shape, scale=scale, size=n_)
samples_weibull_truncated = [x for x in samples_weibull if a_ <= x <= b_]
display(Math(f'${\\begin{equation}} \\text{{Weibull (truncated to [0, 1])}} {\\end{equation}}$'))
testSample(samples_weibull_truncated)

```

Список использованных источников

1. Севастьянов Б.А. Курс теории вероятностей и математической статистики.
–Москва-Ижевск, 2019. - 95с