**Цель работы**

Изучить понятие объекта, средства его создания и уничтожения, а также отношения между объектами.

**Постановка задачи**

1. Ознакомиться с вопросами использования конструкторов и деструкторов для создания и уничтожения объектов и с отношениями между объектами.

2. Модифицировать программу, разработанную в лабораторных работах №1, 2, так чтобы в ней были определены классы, реализующие смесь распределений и эмпирическое распределение. Основное распределение: **симметричное гиперболическое распределение** описывает класс MainDist.

**Путеводитель по работе**

*Реализация*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Распределение | | Основные функции | | |
| Плотность | Характеристики *M*ξ, *D*ξ, γ1, γ2 | Моделирование |
| Основное | Стандартное | Формула (1.1) | *M*ξ= γ1=0, *D*ξ = , γ2 = Формула (1.5) | Формула (1.4) |
| Сдвигмасштаб |  | *M*ξ=μ, *D*ξ=σ2λ2, где σ2 = , γ1 =0, γ2 = Формула (1.5) | Формула (1.4) |
| Смесь | |  | , *D*ξ = Формула (2.3), γ1 =Формула (2.4), γ2 = Формула (2.5) | Формула (2.6) |
| Эмпирическое | |  | , | Формула (3.7) |

*Тестирование (подпункты пункта 3 содержания работы)*

|  |  |  |  |
| --- | --- | --- | --- |
| Распределение | Основные функции | | |
| Плотность | Характеристики *M*ξ, *D*ξ, γ1, γ2 | Модели-рование |
| Смесь | 3.2 | 3.2 | 3.3.1 |
| Эмпирическое | 3.3.1 | 3.3.1, 3.3.2 | 3.3.2 |

**Описание разработанного класса**

**Смесь распределений – MixtureDist**:

* Закрытые (приватные) атрибуты:
  + P – параметр смеси (вещественное число) всегда в диапазоне [0, 1]
  + Dist1 – первый компонент смеси (MainDist)
  + Dist2 – второй компонент смеси (MainDist)
* Конструкторы:
  + Стандартный (без аргументов)
  + С аргументами p, dist1, dist2 – объекты основного распределения передаются по константной ссылке и инициализируются конструктором копирования класса MainDist
  + Из файла file\_name – чтение ранее сохраненного в файл объекта MixtureDist
  + Конструктор копирования и оператор присваивания копированием
  + Конструктор перемещения и оператор присваивания перемещения
* Деструктор – так как все атрибуты класса имеют собственные деструкторы, классу достаточно стандартного деструктора
* Доступ к состоянию объекта:
  + Component1/2 – получить ссылку на первый/второй компонент смеси (MainDist), так же методы перегружены получать константную ссылку, в случае вызова из константного объекта
  + getP/setP – получить/изменить параметр смеси
  + save – сохранить состояние объекта в файл
* Функции из интерфейса:
  + density(x) – возвращает плотность распределения в точке/точках x
  + M() – возвращает матожидание распределения
  + D() – возвращает дисперсию распределения
  + G1() – возвращает коэффициент асимметрии распределения
  + G2() – возвращает коэффициент эксцесса распределения
  + Xi() – возвращает смоделированную случайную величину/величины

**Эмпирическое распределение – EmpiricDist**:

Так как объект класса подразумевает динамическое использование массивов / списков, для удобства еще в прошлой лабораторной работе был создан вспомогательный класс nstu::vector, являющийся собственной реализацией динамического массива для вещественных чисел vector из STL.

* Закрытые (приватные) атрибуты:
  + InitSample – начальная выборка, по которой строится распределение (nstu::vector), не должна быть пустой
  + K – количество интервалов с относительной частотой (целое число), всегда больше 0, значение по умолчанию рассчитывается формулой Стрерджеса
  + IntervalFreq – относительные частоты на интервалах (nstu::vector)
  + Min – самое малое значение выборки (нижняя граница) (вещественное число)
  + Max – самое большое значение выборки (верхняя граница) (вещественное число)
  + Delta – длина интервала (вещественное число)
* Конструкторы:
  + Стандартный (без аргументов)
  + С аргументами init\_sample – начальная выборка, k – количество интервалов (по умолчанию вычисляется по формуле Стерджеса)
  + С аргументами n – размер выборки, dist – распределение (MainDist/MixtureDist/EmpiricDist), k – количество интервалов (по умолчанию вычисляется по формуле Стерджеса)
  + Из файла file\_name – чтение ранее сохраненного в файл объекта EmpiricDist
  + Конструктор копирования и оператор присваивания копированием
  + Конструктор перемещения и оператор присваивания перемещения
* Деструктор – так как все атрибуты класса имеют собственные деструкторы, классу достаточно стандартного деструктора
* Доступ к состоянию объекта:
  + getInitSample – получить копию начальной выборки
  + setInitSample – изменить начальную выборку
  + getK/setK – получить/изменить количество интервалов
  + getMin/Max – получить минимальное/максимальное значение выборки
* Функции из интерфейса:
  + density(x) – возвращает плотность распределения в точке/точках x
  + M() – возвращает матожидание распределения
  + D() – возвращает дисперсию распределения
  + G1() – возвращает коэффициент асимметрии распределения
  + G2() – возвращает коэффициент эксцесса распределения
  + Xi() – возвращает смоделированную случайную величину/величины

**Алгоритм**

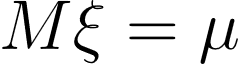
Расчет **основного** распределения:

Расчет плотности симметрического гиперболического распределения осуществляется по формуле:

 (1.1)

С учетом сдвига и масштаба:

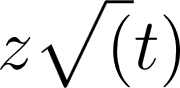
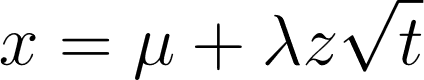
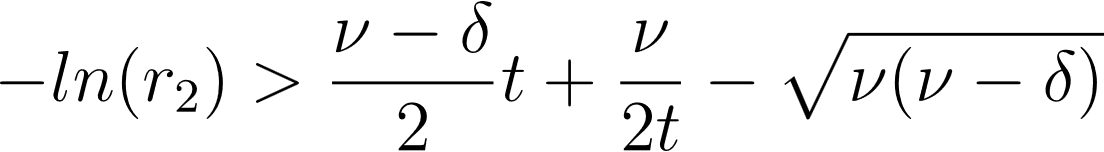
(1.2)

Математическое ожидание:  (1.3)

Дисперсия: *D*ξ = λ2 (1.4)

Коэффициент асимметрии: γ1 =0

Коэффициент эксцесса: γ2 =  (1.5)

Случайная величина: x =  без сдвига-масштаба и  со сдвигом-масштабом соответственно, где , ,  при , r1, r2, r3, r4 – случайные величины, равномерно распределенные на интервале (0, 1). (1.6)

Расчет **смеси** распределений:

Расчет плотности смеси двух распределений осуществляется по формуле:

 (2.1)

Математическое ожидание:  (2.2)

Дисперсия:  (2.3)

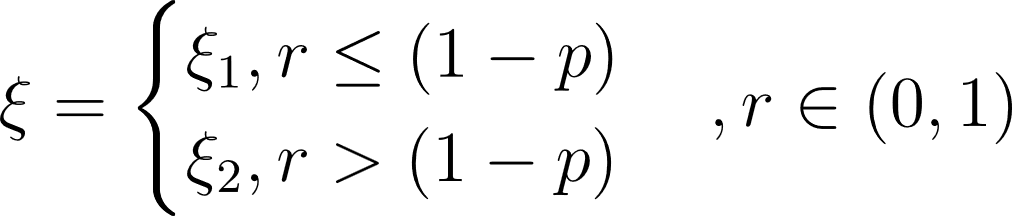
Коэффициент асимметрии:

 (2.4)

Коэффициент эксцесса:

 (2.5)

Случайная величина:

 (2.6)

Расчет **эмпирического** распределения:

Расчет эмпирической плотности распределения:

 (3.1)

Здесь предполагается, что промежуток  разбит на  непересекающихся промежутков , , длины , при этом каждый промежуток содержит свой левый конец, но лишь последний промежуток содержит и свой правый конец,  – количество элементов выборки, содержащихся в промежутке . Таким образом, имеем промежутки

, ,

.

 (3.2)

Математическое ожидание:

 (3.3)

Дисперсия:

 (3.4)

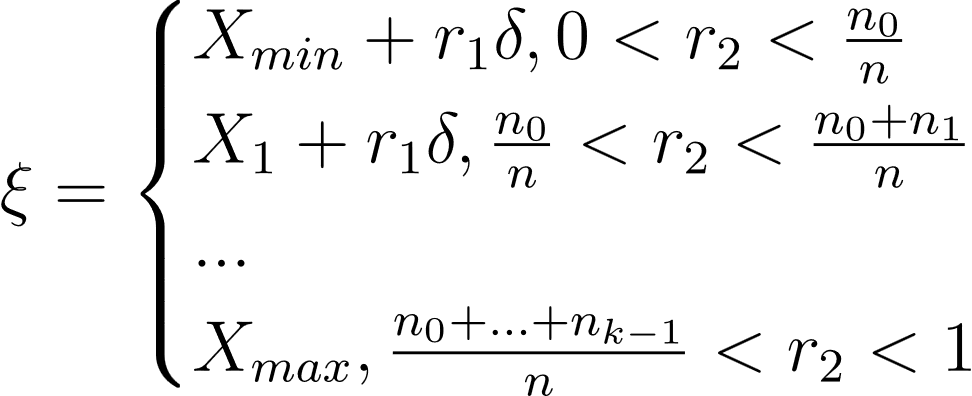
Коэффициент эксцесса:

 (3.5)

Коэффициент эксцесса:

 (3.6)

Случайная величина:

 (3.7)

**Тестирование**

3.2. Минимальный набор тестов для смеси распределений (см. пример 1.2):

3.2.1) тест для тривиального случая: μ1=μ2=10, λ1=λ2=2, ν1=ν2 = 1, *p* = 0.5;

3.2.2) тест для сдвиговых преобразований: μ1=0, μ2=2, λ1=λ2=1, ν1=ν2 = 1, *p*=0.75 (*M*ξ=1.5, *D*ξ=+0.75, γ1= –0.75/);

3.2.3) тест для масштабных преобразований: μ1=μ2=0, λ1=1, λ2=3, ν1=ν2 = 1, *p*=0.5 (*M*ξ=0, *D*ξ=5, γ1=0, γ2=1.64(γ2*i*+3)–3);

3.2.4) тест с неравными параметрами формы: μ1=μ2=0, λ1=λ2=1, ν1=0.1, ν2 = 30, *p*=0.5 (*M*ξ=0, *D*ξ=()/2, γ1=0, γ2=0.5((γ21+3)+ (γ22+3))/(*D*ξ)2 –3).

3.3. Тестирование эмпирического распределения и функций моделирования случайных величин для всех распределений:

3.3.1) для (нестандартных) основного распределения и смеси при некоторых значениях их параметров.

3.3.2) в соответствии с эмпирической плотностью, построенной по одной из выборок, сгенерировать новую выборку того же объема, вычислить ее эмпирические характеристики, сравнить их с эмпирическими характеристиками исходной выборки и теоретическими характеристиками.

Тесты методов класса MainDist:

1. Тестирование конструкторов и геттеров
2. Тестирование операторов присваивания
3. Тестирование сеттеров
4. Тестирование аналитических методов
5. Тестирование генерации случайных чисел
6. Тестирование обработки исключений

**Результаты тестирования:**

-----Тест 3.2.1-----

Тест распределения смеси:

nu1 = 1mu1 = 10, lambda1 = 2

nu2 = 1mu2 = 10, lambda2 = 2

p = 0.5, x = 0

Плотность вероятности f(x): s0.00253475, c0.00253475

M = s10, c10; D = s10.7979, c10.7979; G1 = s0, c0; G2 = s1.85697, c1.85697

-----Тест 3.2.2-----

Тест распределения смеси:

nu1 = 1mu1 = 0, lambda1 = 1

nu2 = 1mu2 = 2, lambda2 = 1

p = 0.75, x = 0

Плотность вероятности f(x): s0.142986, c0.142986

M = s1.5, c1.5; D = s3.44948, c3.44948; G1 = s-0.117066, c-0.117066; G2 = s1.10574, c1.10574

-----Тест 3.2.3-----

Тест распределения смеси:

nu1 = 1mu1 = 0, lambda1 = 1

nu2 = 1mu2 = 0, lambda2 = 3

p = 0.5, x = 0

Плотность вероятности f(x): s0.20373, c0.20373

M = s0, c0; D = s13.4974, c13.4974; G1 = s0, c0; G2 = s4.96544, c4.96544

-----Тест 3.2.4-----

Тест распределения смеси:

nu1 = 0.1mu1 = 0, lambda1 = 1

nu2 = 30mu2 = 0, lambda2 = 1

p = 0.5, x = 0

Плотность вероятности f(x): s0.269628, c0.269628

M = s0, c0; D = s10.6484, c10.6484; G1 = s0, c0; G2 = s7.74183, c7.74183

-----Тест 3.3.1.1-----

Тест эмпирического распределения:

Распределение построено на основе выборки 10000 элементов из основного распределения со следующими параметрами:

nu = 1, mu = 0, lambda = 1

Выборка:

f(0) = 0.305595, M = 0, D = 2.69948, G1 = 0, G2 = 1.85697

Эмпирическое распределение:

f(0) = s0.234022, c0.234022, M = s0.000471055, c0.000471055; D = s2.75624, c2.75624; G1 = s-0.0618734, c-0.0618734; G2 = s2.06422, c2.06422

-----Тест 3.3.1.2-----

Тест эмпирического распределения:

Распределение построено на основе выборки 10000 элементов из основного распределения со следующими параметрами:

nu = 3, mu = 10, lambda = 2

Выборка:

f(0) = 0.000376053, M = 10, D = 6.12708, G1 = 0, G2 = 0.889953

Эмпирическое распределение:

f(0) = s0.000577659, c0.000577717, M = s9.98603, c9.98603; D = s5.97474, c5.97474; G1 = s0.00643799, c0.00643799; G2 = s0.870862, c0.870862

-----Тест 3.3.1.3-----

Тест эмпирического распределения:

Распределение построено на основе выборки 10000 элементов из распределения смеси со следующими параметрами:

nu1 = 1mu1 = 0, lambda1 = 1

nu2 = 1mu2 = 0, lambda2 = 2

p = 0.5

Выборка:

f(0) = 0.229196, M = 0, D = 6.74871, G1 = 0, G2 = 3.60548

Эмпирическое распределение:

f(0) = s0.183205, c0.183215, M = s-0.000301592, c-0.000301592; D = s6.76417, c6.76417; G1 = s0.0952404, c0.0952404; G2 = s4.41119, c4.41119

-----Тест 3.3.1.4-----

Тест эмпирического распределения:

Распределение построено на основе выборки 10000 элементов из распределения смеси со следующими параметрами:

nu1 = 1mu1 = 0, lambda1 = 1

nu2 = 3mu2 = 3, lambda2 = 2

p = 0.4

Выборка:

f(0) = 0.210529, M = 1.2, D = 6.23052, G1 = 0.559388, G2 = 0.779838

Эмпирическое распределение:

f(0) = s0.182012, c0.182039, M = s1.15531, c1.15531; D = s6.15644, c6.15644; G1 = s0.525294, c0.525294; G2 = s0.882556, c0.882556

-----Тест 3.3.2-----

Тест эмпирического распределения:

Распределение построено на основе выборки 10000 элементов из основного распределения со следующими параметрами:

nu = 1, mu = 0, lambda = 1

Выборка:

f(0) = 0.305595, M = 0, D = 2.69948, G1 = 0, G2 = 1.85697

Эмпирическое распределение1:

f(0) = s0.285359, c0.285375, M = s0.00749292, c0.00749292; D = s2.67739, c2.67739; G1 = s-0.0387297, c-0.0387297; G2 = s1.77188, c1.77188

Эмпирическое распределение2:

f(0) = 0.255098, M = 0.0142977, D = 2.98521, G1 = -0.181769, G2 = 1.48567

--- НАЧАЛО ПОЛНОГО ТЕСТИРОВАНИЯ КЛАССА MIXTUREDIST ---

1. Тестирование конструкторов

--------------------------------

1.1. Конструктор по умолчанию

- Состояние mix\_default: p=0.500000

- Comp1 (nu, mu, lambda): 1.000000, 0.000000, 1.000000

- Comp2 (nu, mu, lambda): 1.000000, 0.000000, 1.000000

1.2. Конструктор с параметрами

- Состояние mix\_params: p=0.300000

- Comp1 (nu, mu, lambda): 5.000000, -2.000000, 1.000000

- Comp2 (nu, mu, lambda): 3.000000, 2.000000, 1.500000

1.3. Конструктор копирования

- Состояние mix\_copy: p=0.300000

- Comp1 (nu, mu, lambda): 5.000000, -2.000000, 1.000000

- Comp2 (nu, mu, lambda): 3.000000, 2.000000, 1.500000

1.4. Конструктор перемещения

- Состояние mix\_move: p=0.800000

- Comp1 (nu, mu, lambda): 5.000000, -2.000000, 1.000000

- Comp2 (nu, mu, lambda): 3.000000, 2.000000, 1.500000

2. Тестирование сохранения и загрузки (save/load)

------------------------------------------------

- Сохраняем mix\_params в файл 'mixture\_params.txt'...

- Загружаем из файла в новый объект...

- Состояние mix\_loaded: p=0.300000

- Comp1 (nu, mu, lambda): 5.000000, -2.000000, 1.000000

- Comp2 (nu, mu, lambda): 3.000000, 2.000000, 1.500000

>>> ОШИБКА: Данные после сохранения и загрузки не совпадают!

3. Тестирование геттеров и сеттеров

------------------------------------

- Начальное значение p: 0.500000

- Новое значение p после setP(0.9): 0.900000

- Проверка неконстантного геттера component1():

- Значение mu у component1 после изменения через ссылку: 99.000000

>>> ПОДТВЕРЖДЕНО: Неконстантный геттер работает корректно.

4. Тестирование аналитических методов

-------------------------------------

- Состояние mix\_params: p=0.300000

- Comp1 (nu, mu, lambda): 5.000000, -2.000000, 1.000000

- Comp2 (nu, mu, lambda): 3.000000, 2.000000, 1.500000

- Математическое ожидание (M): -0.800000

- Дисперсия (D): 5.312763

- Асимметрия (G1): 0.878143

- Эксцесс (G2): 0.352772

- Плотность в точке x=1.000000: 0.066757 (ожидалось: 0.066757)

5. Тестирование генерации случайных чисел (Xi)

----------------------------------------------

- Запуск 1 с seed = 12345

Сгенерированные числа: -0.276677 -2.269039 0.330590 -0.605092 -1.859422

- Запуск 2 с тем же seed = 12345

Сгенерированные числа: -0.276677 -2.269039 0.330590 -0.605092 -1.859422

>>> ПОДТВЕРЖДЕНО: Воспроизводимость генерации для MixtureDist работает!

6. Тестирование обработки исключений

------------------------------------

- Успех (p < 0 в конструкторе): Параметр смеси должен быть в диапазоне [0, 1]

- Успех (p > 1 в конструкторе): Параметр смеси должен быть в диапазоне [0, 1]

- Успех (p < 0 в сеттере): Параметр смеси должен быть в диапазоне [0, 1]

- Успех (p > 1 в сеттере): Параметр смеси должен быть в диапазоне [0, 1]

- Успех (невалидные параметры для MainDist в файле): Лямбда не может быть 0

--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---

--- НАЧАЛО ПОЛНОГО ТЕСТИРОВАНИЯ КЛАССА EMPIRICDIST ---

1. Тестирование конструкторов

--------------------------------

1.1. Конструктор из готовой выборки nstu::vector

- Состояние ed\_from\_vec:

- k=4.000000, min=10.000000, max=50.000000, sample\_size=10

>>> ПОДТВЕРЖДЕНО: Параметры рассчитаны корректно.

1.2. Конструктор из другого распределения (MainDist)

- Состояние ed\_from\_main:

- k=7.000000, min=-5.925562, max=5.649925, sample\_size=100

1.3. Конструктор из другого распределения (MixtureDist)

- Состояние ed\_from\_mix:

- k=7.000000, min=-8.173454, max=7.068633, sample\_size=100

2. Тестирование сохранения и загрузки (save/load)

------------------------------------------------

- Состояние ed\_loaded:

- k=1.000000, min=10.000000, max=50.000000, sample\_size=10

>>> ПОДТВЕРЖДЕНО: Сохраненная и загруженная выборки совпадают!

3. Тестирование сеттеров

------------------------

- Начальное k=4.000000

- Новое k после setK(5): 1.000000

- Установлена новая выборка размером 5

- Состояние ed\_set (после setInitSample):

- k=1.000000, min=1.000000, max=5.000000, sample\_size=5

>>> ПОДТВЕРЖДЕНО: Объект корректно пересчитан.

4. Тестирование аналитических методов

-------------------------------------

- Состояние ed\_analysis:

- k=3.000000, min=1.000000, max=5.000000, sample\_size=5

- Математическое ожидание (M): 3.000000 (ожидалось: 3.0)

- Дисперсия (D): 2.000000 (ожидалось: 2.0)

- Асимметрия (G1): 0.000000 (ожидалось: 0.0)

- Плотность в точке x=2.500000: 0.150000

5. Тестирование генерации случайных чисел (Xi)

----------------------------------------------

- Запуск 1: 23.996775 22.121219 10.541498 20.477572 32.727337

- Запуск 2: 23.996775 22.121219 10.541498 20.477572 32.727337

>>> ПОДТВЕРЖДЕНО: Воспроизводимость генерации для EmpiricDist работает!

6. Тестирование обработки исключений

------------------------------------

- Успех (пустая выборка в конструкторе): Начальная выборка не может быть пустой

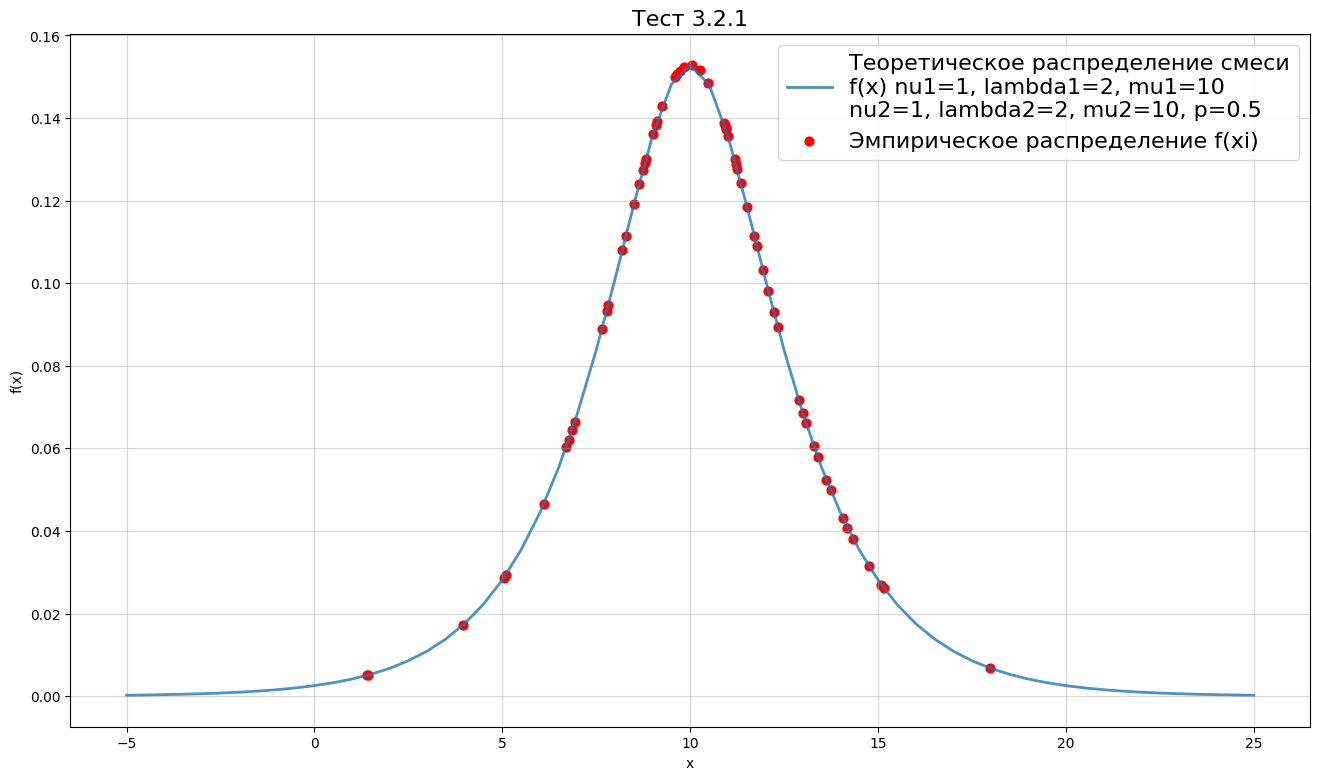
- Успех (несуществующий файл): Не удалось открыть файл non\_existent\_file.txt

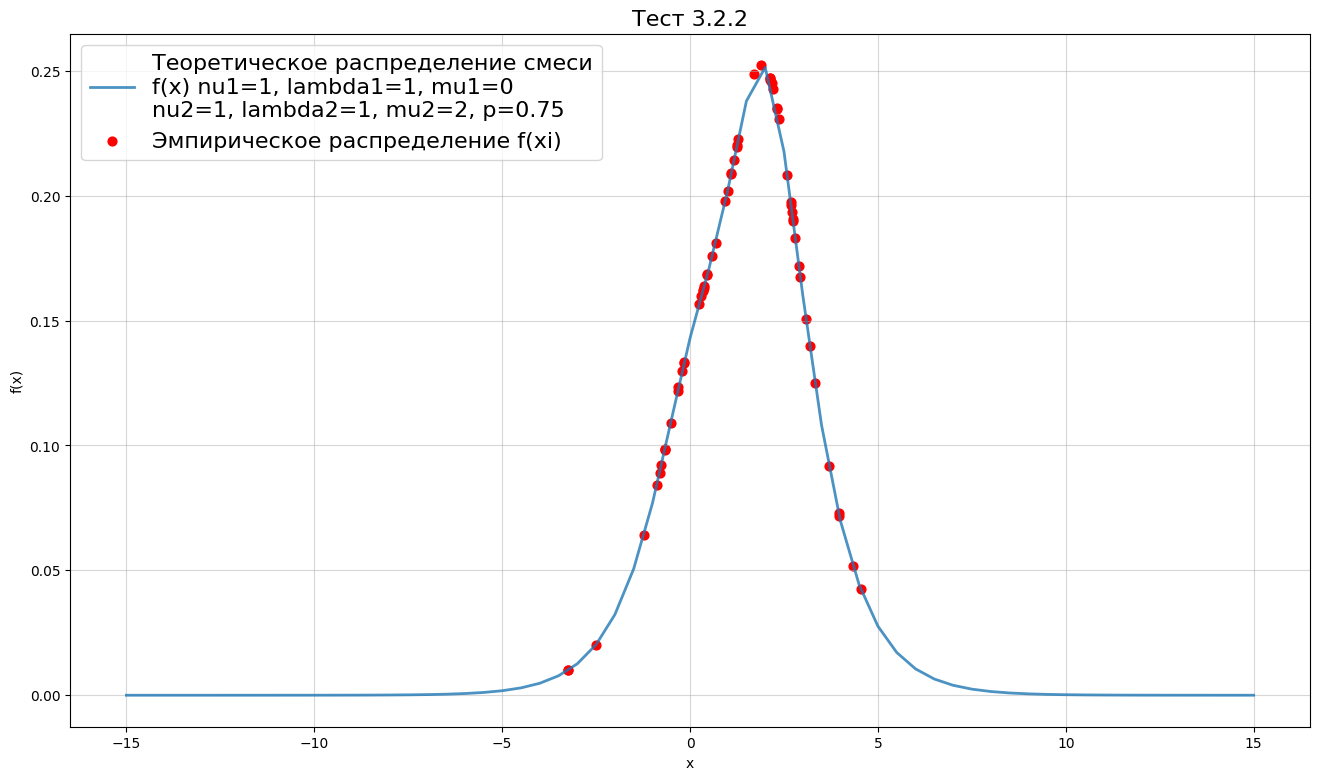
--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---

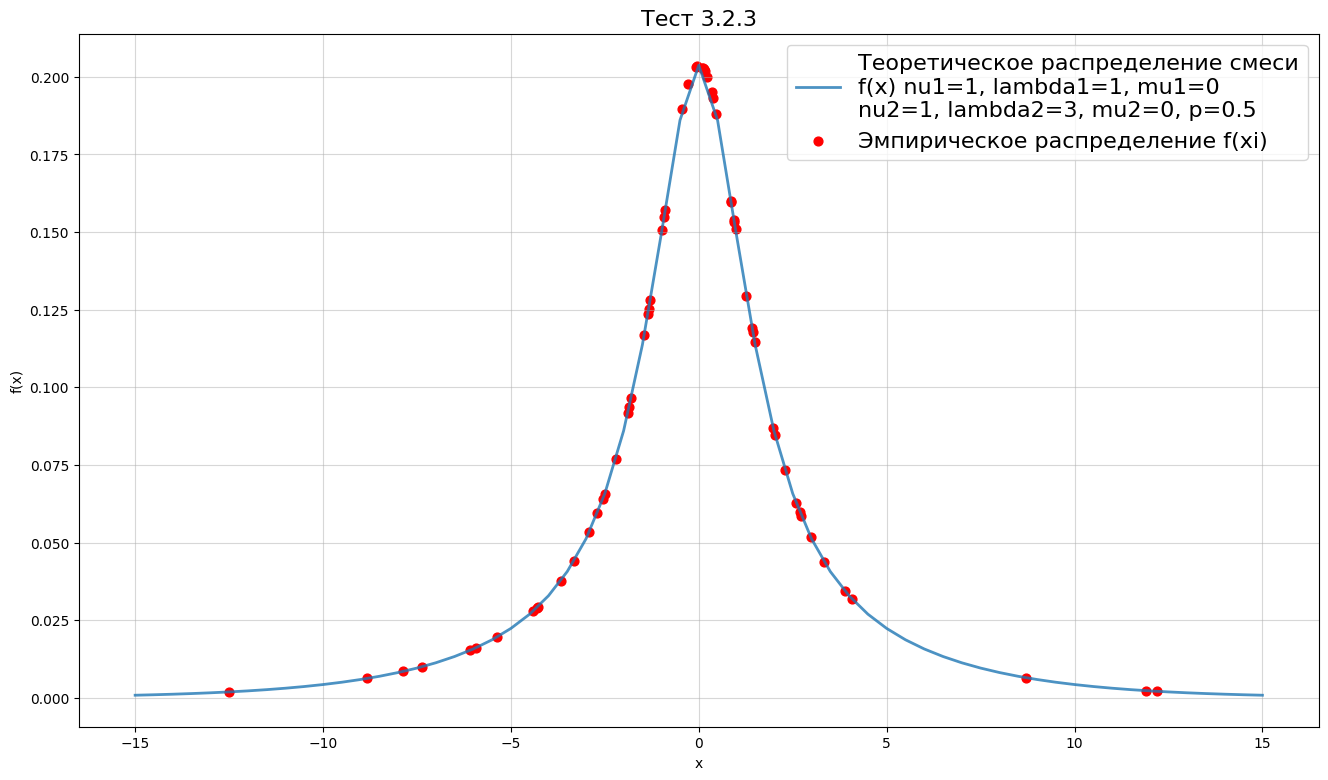
**Графики распределений**

Все графики построены на выборке из 10000 элементов с помощью python.

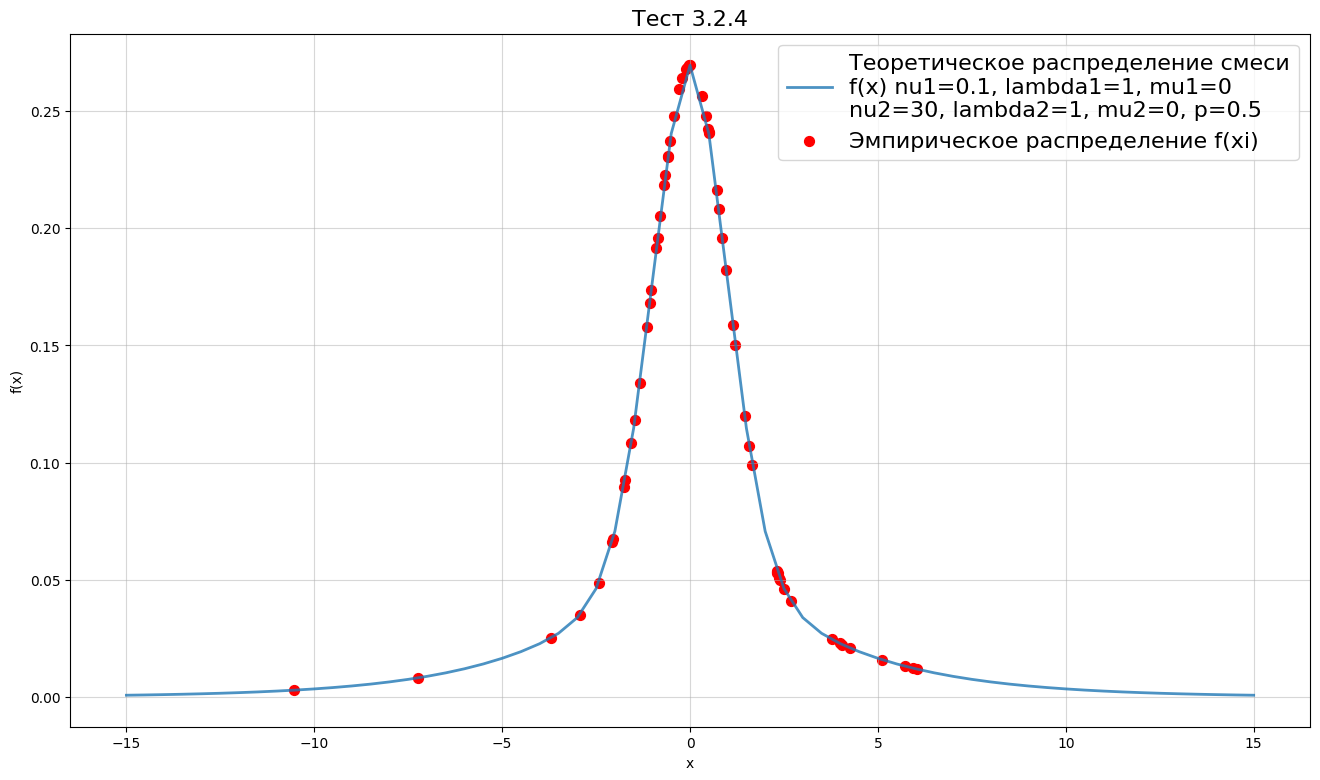
Графики тестов:

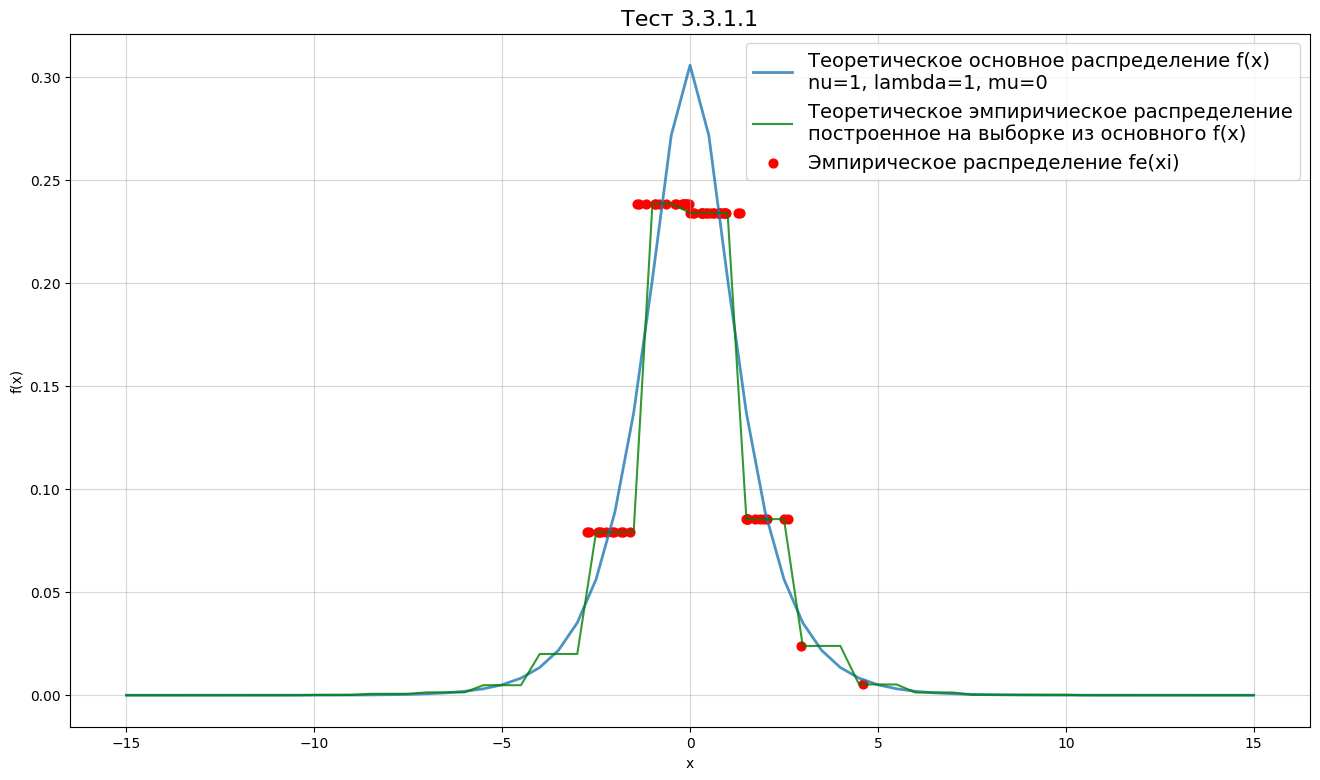
3.2.1

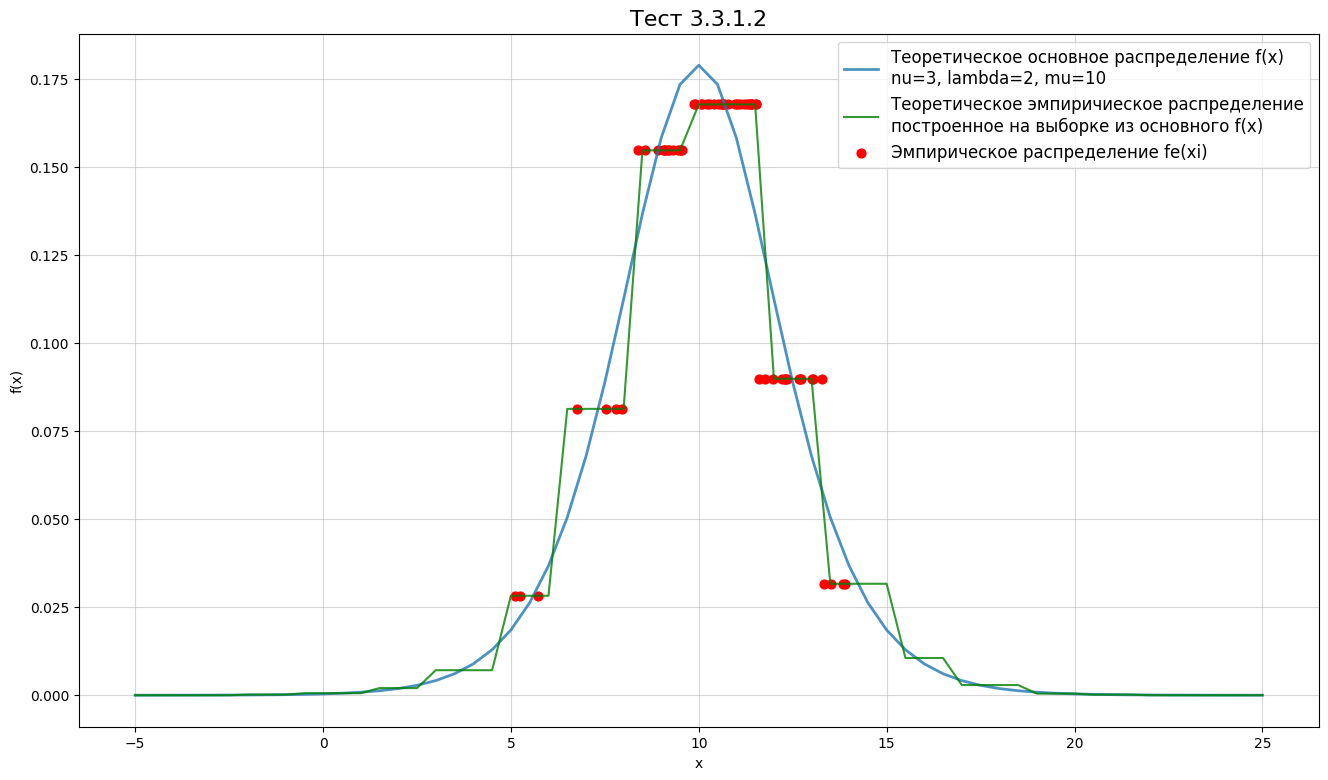
3.2.2

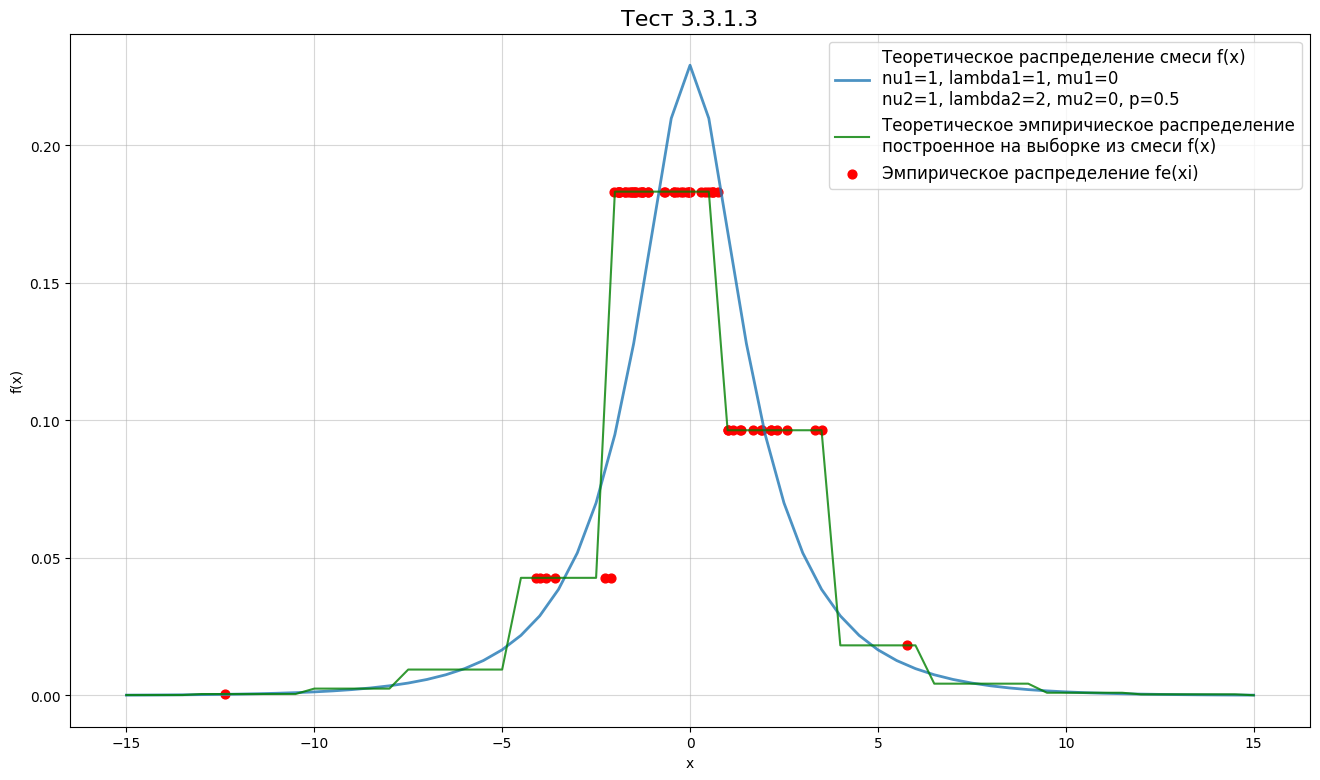
3.2.3

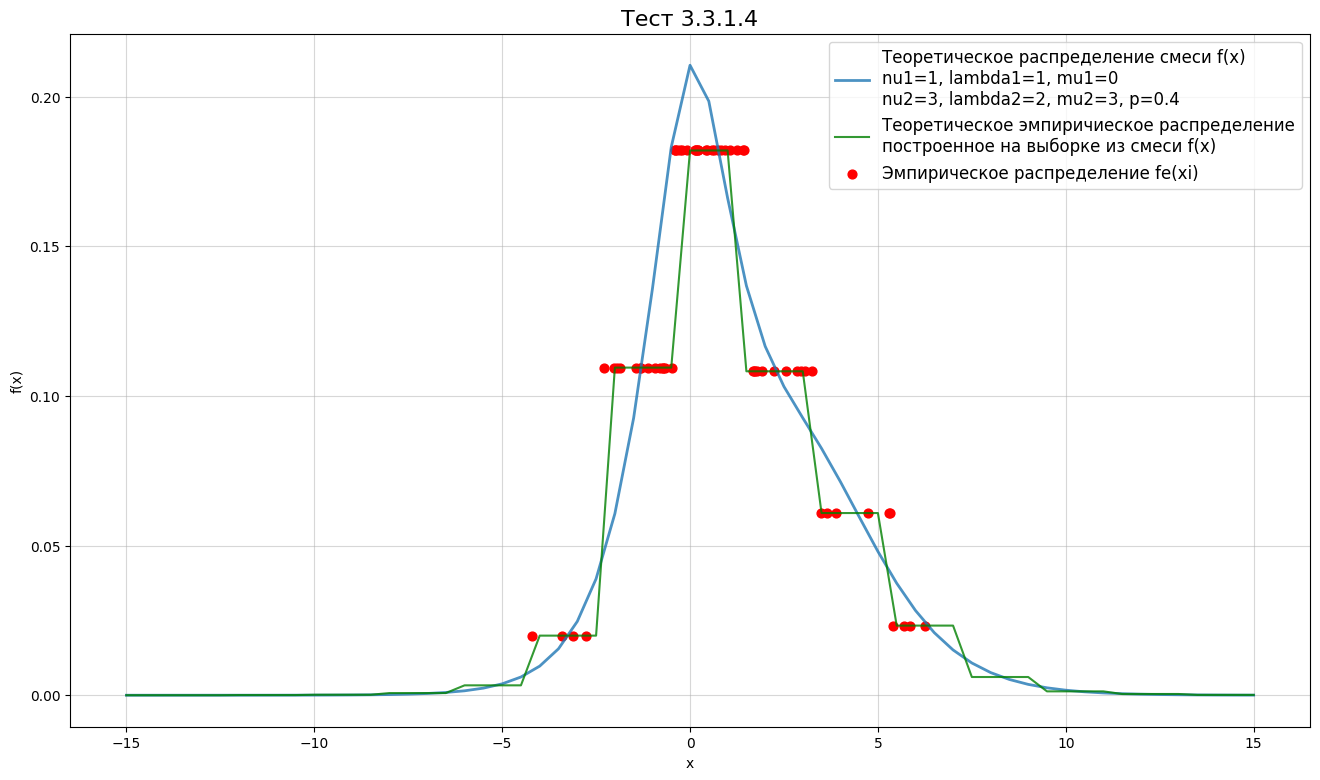
3.2.4

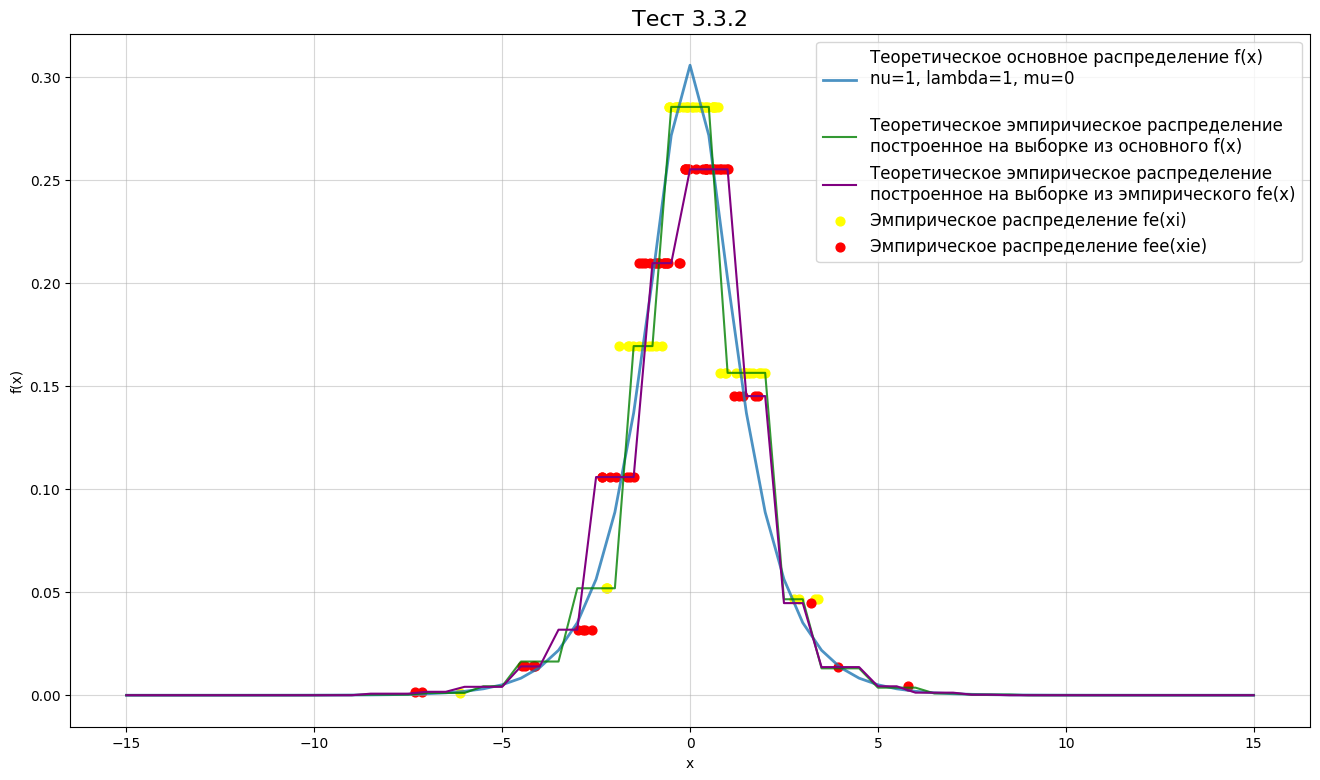


3.3.1.1

3.3.1.2

3.3.1.3

3.3.1.4

3.3.2

**Вывод**

В ходе лабораторной работы мы создали классы смеси распределений и эмпирического распределения, изучили понятие объекта, средства его создания и уничтожения, а также отношения между объектами.

**Приложение**

Код программы

**Объявление класса vector и функции randNum (../lab2/header.h):**

#pragma once

#include <memory>

#include <stdexcept>

using ld = long double;

namespace nstu {

const ld pi = 3.14159265358979;

inline ld randNum() {

  ld r;

  do {

    r = static\_cast<ld>(rand()) / RAND\_MAX;

  } while (r == 0 || r == 1);

  return r;

}

class vector {

 private:

  std::unique\_ptr<ld[]> data;

  uint32\_t length;

  uint32\_t full\_length;

  // ========== Vector Iterator =========

  class VectorIterator {

   private:

    ld\* ptr;

   public:

    using iterator\_category = std::random\_access\_iterator\_tag;

    explicit VectorIterator(ld\* p) : ptr(p) {}

    ld& operator\*() const { return \*ptr; }

    ld\* operator->() { return ptr; }

    VectorIterator& operator++() {

      ++ptr;

      return \*this;

    }

    VectorIterator operator++(int) {

      VectorIterator tmp = \*this;

      ++(\*this);

      return tmp;

    }

    VectorIterator& operator--() {

      --ptr;

      return \*this;

    }

    VectorIterator operator--(int) {

      VectorIterator tmp = \*this;

      --(\*this);

      return tmp;

    }

    VectorIterator operator+(int i) const { return VectorIterator(ptr + i); }

    VectorIterator operator-(int i) const { return VectorIterator(ptr - i); }

    friend bool operator==(const VectorIterator& a, const VectorIterator& b) {

      return a.ptr == b.ptr;

    }

    friend bool operator!=(const VectorIterator& a, const VectorIterator& b) {

      return a.ptr != b.ptr;

    }

  };

  class ConstVectorIterator {

   private:

    ld\* ptr;

   public:

    using iterator\_category = std::random\_access\_iterator\_tag;

    explicit ConstVectorIterator(ld\* p) : ptr(p) {}

    const ld operator\*() const { return \*ptr; }

    const ld\* operator->() { return ptr; }

    ConstVectorIterator& operator++() {

      ++ptr;

      return \*this;

    }

    ConstVectorIterator operator++(int) {

      ConstVectorIterator tmp = \*this;

      ++(\*this);

      return tmp;

    }

    ConstVectorIterator& operator--() {

      --ptr;

      return \*this;

    }

    ConstVectorIterator operator--(int) {

      ConstVectorIterator tmp = \*this;

      --(\*this);

      return tmp;

    }

    ConstVectorIterator operator+(int i) const {

      return ConstVectorIterator(ptr + i);

    }

    ConstVectorIterator operator-(int i) const {

      return ConstVectorIterator(ptr - i);

    }

    friend bool operator==(const ConstVectorIterator& a,

                           const ConstVectorIterator& b) {

      return a.ptr == b.ptr;

    }

    friend bool operator!=(const ConstVectorIterator& a,

                           const ConstVectorIterator& b) {

      return a.ptr != b.ptr;

    }

  };

  // ===================================

 public:

  using iterator = VectorIterator;

  using const\_iterator = ConstVectorIterator;

  explicit vector() : length(0), full\_length(1) {

    data = std::unique\_ptr<ld[]>(new ld[full\_length]);

    if (!data) throw std::bad\_alloc();

  }

  explicit vector(uint32\_t length, ld value = 0) : full\_length(length \* 2) {

    this->length = length;

    data = std::unique\_ptr<ld[]>(new ld[full\_length]);

    if (!data) throw std::bad\_alloc();

    if (value != 0) std::fill(data.get(), data.get() + length, value);

  }

  vector(std::initializer\_list<ld> list)

      : length(list.size()), full\_length(list.size() \* 2) {

    data = std::unique\_ptr<ld[]>(new ld[full\_length]);

    if (!data) throw std::bad\_alloc();

    std::copy(list.begin(), list.end(), data.get());

  }

  vector(const vector& other)

      : length(other.length), full\_length(other.full\_length) {

    data = std::unique\_ptr<ld[]>(new ld[full\_length]);

    if (!data) throw std::bad\_alloc();

    std::copy(other.data.get(), other.data.get() + other.length, data.get());

  }

  vector& operator=(const vector& other) {

    if (this != &other) {

      length = other.length;

      full\_length = other.full\_length;

      data = std::unique\_ptr<ld[]>(new ld[full\_length]);

      if (!data) throw std::bad\_alloc();

      std::copy(other.data.get(), other.data.get() + other.length, data.get());

    }

    return \*this;

  }

  vector(vector&& other) noexcept

      : data(std::move(other.data)),

        length(other.length),

        full\_length(other.full\_length) {

    other.length = 0;

    other.full\_length = 0;

  }

  vector& operator=(vector&& other) noexcept {

    if (this != &other) {

      data = std::move(other.data);

      length = other.length;

      full\_length = other.full\_length;

      other.length = 0;

      other.full\_length = 0;

    }

    return \*this;

  }

  uint32\_t size() const noexcept { return length; }

  uint32\_t capacity() const noexcept { return full\_length; }

  bool empty() const noexcept { return length == 0; }

  const ld operator[](uint32\_t i) const {

    if (i >= length) throw std::out\_of\_range("Выход за границы массива");

    return data.get()[i];

  }

  ld& operator[](uint32\_t i) {

    if (i >= length) throw std::out\_of\_range("Выход за границы массива");

    return data.get()[i];

  }

  void push\_back(ld value) {

    if (length >= full\_length) {

      full\_length \*= 2;

      std::unique\_ptr<ld[]> new\_data(new ld[full\_length]);

      if (!new\_data) throw std::bad\_alloc();

      std::copy(data.get(), data.get() + length, new\_data.get());

      data = std::move(new\_data);

    }

    data.get()[length++] = value;

  }

  void pop\_back() {

    if (length == 0) throw std::out\_of\_range("Вектор пустой");

    --length;

  }

  ld pop() {

    if (length == 0) throw std::out\_of\_range("Вектор пустой");

    return data.get()[--length];

  }

  iterator begin() noexcept { return iterator(data.get()); }

  iterator end() noexcept { return iterator(data.get() + length); }

  const\_iterator begin() const noexcept { return const\_iterator(data.get()); }

  const\_iterator end() const noexcept {

    return const\_iterator(data.get() + length);

  }

  ~vector() {}

};

}  // namespace nstu

**Описание класса MainDist (../lab2/MainDist.hpp):**

#pragma once

#include <string>

#include "header.h"

class MainDist {

 private:

  ld m\_nu;      // параметр формы

  ld m\_mu;      // сдвиг

  ld m\_lambda;  // масштаб

 public:

  // Конструкторы

  // Стандартный конструктор без параметров

  MainDist();

  /\*\*

   \* Конструктор с параметрами распределения

   \* @param nu Параметр формы распределения (не может быть равен 0).

   \* @param mu Параметр сдвига распределения.

   \* @param lambda Параметр масштаба распределения (не может быть равен 0).

   \*/

  explicit MainDist(ld nu, ld mu, ld lambda);

  /\*\*

   \* Конструктор чтения из файла

   \* @param file\_name Имя файла, из которого читается распределение.

   \*/

  explicit MainDist(std::string file\_name);

  /\*\*

   \* Сохранение распределения в файл

   \* @param file\_name Имя файла, в который сохраняется распределение.

   \*/

  void save(const std::string& file\_name) const;

  /\*\*

   \* Конструктор с параметрами распределения из списка инициализации

   \* @param list Список инициализации из трех элементов: nu, mu, lambda - форма,

   \* сдвиг и масштаб соответственно.

   \*/

  explicit MainDist(std::initializer\_list<ld> list);

  // Конструктор копирования

  MainDist(const MainDist& other);

  // Оператор присваивания копированием

  MainDist& operator=(const MainDist& other);

  // Конструктор перемещения

  MainDist(MainDist&& other) noexcept;

  // Оператор присваивания перемещением

  MainDist& operator=(MainDist&& other) noexcept;

  // Деструктор

  ~MainDist() = default;

  // Геттеры

  /\*\*

   \* Получение параметра формы nu

   \* @return Значение параметра nu

   \*/

  ld getNu() const noexcept;

  /\*\*

   \* Получение параметра сдвига mu

   \* @return Значение параметра mu

   \*/

  ld getMu() const noexcept;

  /\*\*

   \* Получение параметра масштаба lambda

   \* @return Значение параметра lambda

   \*/

  ld getLambda() const noexcept;

  // Сеттеры

  /\*\*

   \* Изменение параметра формы nu (не может быть 0)

   \* @param nu Новое значение параметра nu

   \*/

  void setNu(ld nu);

  /\*\*

   \* Изменение параметра сдвига mu

   \* @param mu Новое значение параметра mu

   \*/

  void setMu(ld mu) noexcept;

  /\*\*

   \* Изменение параметра масштаба lambda (не может быть 0)

   \* @param lambda Новое значение параметра lambda

   \*/

  void setLambda(ld lambda);

  // Функции интерфейса

  /\*\*

   \* Вычисление плотности вероятности

   \* @param x Значение x

   \* @return Значение плотности вероятности

   \*/

  ld density(ld x) const noexcept;

  /\*\*

   \* Вычисление плотности вероятности для списка значений

   \* @param x Список значений

   \* @return Список значений плотности вероятности

   \*/

  nstu::vector density(const nstu::vector& x) const;

  /\*\*

   \* Вычисление математического ожидания

   \* @return Значение математического ожидания

   \*/

  ld M() const noexcept;

  /\*\*

   \* Вычисление дисперсии

   \* @return Значение дисперсии

   \*/

  ld D() const noexcept;

  /\*\*

   \* Вычисление коэффициента ассиметрии

   \* @return Значение коэффициента ассиметрии

   \*/

  ld G1() const noexcept;

  /\*\*

   \* Вычисление коэффициента эксцесса

   \* @return Значение коэффициента эксцесса

   \*/

  ld G2() const noexcept;

  /\*\*

   \* Моделирование случайной величины

   \* @return Значение случайной величины

   \*/

  ld Xi() const noexcept;

  /\*\*

   \* Моделирование выборки случайных величин

   \* @param size Размер выборки

   \* @return Выборка случайных величин

   \*/

  nstu::vector Xi(uint32\_t size) const;

};

**Реализация класса MainDist (../lab2/MainDist.cpp):**

#include "MainDist.hpp"

#include <boost/math/special\_functions/bessel.hpp>

#include <cmath>

#include <fstream>

MainDist::MainDist() : m\_nu(1.0), m\_mu(0.0), m\_lambda(1.0) {}

MainDist::MainDist(ld nu, ld mu, ld lambda)

    : m\_nu(nu), m\_mu(mu), m\_lambda(lambda) {

  if (lambda == 0) throw std::invalid\_argument("Лямбда не может быть 0");

  if (nu == 0) throw std::invalid\_argument("Нулевое значение nu");

}

MainDist::MainDist(std::string file\_name) {

  std::ifstream file(file\_name);

  if (!file.is\_open()) throw std::runtime\_error("Не удалось открыть файл");

  file >> m\_nu >> m\_mu >> m\_lambda;

  if (m\_lambda == 0) throw std::invalid\_argument("Лямбда не может быть 0");

  if (m\_nu == 0) throw std::invalid\_argument("Нулевое значение nu");

  file.close();

}

void MainDist::save(const std::string& file\_name) const {

  std::ofstream file(file\_name);

  if (!file.is\_open()) throw std::runtime\_error("Не удалось открыть файл");

  file << m\_nu << " " << m\_mu << " " << m\_lambda;

  file.close();

}

MainDist::MainDist(std::initializer\_list<ld> list) {

  if (list.size() != 3)

    throw std::invalid\_argument("Неверное количество аргументов");

  auto i = list.begin();

  m\_nu = \*(i++);

  m\_mu = \*(i++);

  m\_lambda = \*i;

  if (m\_lambda == 0) throw std::invalid\_argument("Лямбда не может быть 0");

  if (m\_nu == 0) throw std::invalid\_argument("Нулевое значение nu");

}

MainDist::MainDist(const MainDist& other)

    : m\_nu(other.m\_nu), m\_mu(other.m\_mu), m\_lambda(other.m\_lambda) {}

MainDist& MainDist::operator=(const MainDist& other) {

  if (this != &other) {

    m\_nu = other.m\_nu;

    m\_mu = other.m\_mu;

    m\_lambda = other.m\_lambda;

  }

  return \*this;

}

MainDist::MainDist(MainDist&& other) noexcept

    : m\_nu(other.m\_nu), m\_mu(other.m\_mu), m\_lambda(other.m\_lambda) {

  other.m\_nu = 0;

  other.m\_mu = 0;

  other.m\_lambda = 0;

}

MainDist& MainDist::operator=(MainDist&& other) noexcept {

  if (this != &other) {

    m\_nu = other.m\_nu;

    m\_mu = other.m\_mu;

    m\_lambda = other.m\_lambda;

    other.m\_nu = 0;

    other.m\_mu = 0;

    other.m\_lambda = 0;

  }

  return \*this;

}

ld MainDist::getNu() const noexcept { return m\_nu; }

ld MainDist::getMu() const noexcept { return m\_mu; }

ld MainDist::getLambda() const noexcept { return m\_lambda; }

void MainDist::setNu(ld nu) {

  if (nu == 0)

    throw std::invalid\_argument("Нулевое значение nu");

  else

    m\_nu = nu;

}

void MainDist::setMu(ld mu) noexcept { m\_mu = mu; }

void MainDist::setLambda(ld lambda) {

  if (lambda == 0)

    throw std::invalid\_argument("Лямбда не может быть 0");

  else

    m\_lambda = lambda;

}

ld MainDist::density(ld x) const noexcept {

  ld coeff = 2 \* m\_lambda \* sqrt(m\_nu) \* boost::math::cyl\_bessel\_k(1, m\_nu);

  ld exponent = exp(-m\_nu \* sqrt(1 + pow((x - m\_mu) / m\_lambda, 2) / m\_nu));

  return exponent / coeff;

}

nstu::vector MainDist::density(const nstu::vector& x) const {

  nstu::vector result(x.size());

  for (uint32\_t i = 0; i < x.size(); ++i) {

    result[i] = density(x[i]);

  }

  return result;

}

ld MainDist::M() const noexcept { return m\_mu; }

ld MainDist::D() const noexcept {

  return pow(m\_lambda, 2) \* boost::math::cyl\_bessel\_k(2, m\_nu) /

         boost::math::cyl\_bessel\_k(1, m\_nu);

}

ld MainDist::G1() const noexcept { return 0; }

ld MainDist::G2() const noexcept {

  return 3 \* boost::math::cyl\_bessel\_k(3, m\_nu) \*

             boost::math::cyl\_bessel\_k(1, m\_nu) /

             pow(boost::math::cyl\_bessel\_k(2, m\_nu), 2) -

         3;

}

ld MainDist::Xi() const noexcept {

  ld r1, r2, delta, t;

  do {

    r1 = nstu::randNum();

    r2 = nstu::randNum();

    delta = 2 \* (sqrt(1 + pow(m\_nu, 2)) - 1) / m\_nu;

    t = -2 \* log(r1) / delta;

  } while (-log(r2) <= (m\_nu - delta) \* t / 2 + m\_nu / (2 \* t) -

                           sqrt(m\_nu \* (m\_nu - delta)));

  ld r3 = nstu::randNum(), r4 = nstu::randNum();

  ld z = sqrt(-2 \* log(r3)) \* cos(2 \* nstu::pi \* r4);

  // z = sqrt(-2 \* log(r3)) \* sin(2 \* pi \* r4);

  return z \* sqrt(t) \* m\_lambda + m\_mu;

}

nstu::vector MainDist::Xi(uint32\_t size) const {

  nstu::vector result(size);

  for (auto& i : result) i = Xi();

  return result;

}

**Описание класса MixtureDist (MixtureDist.hpp):**

#pragma once

// подключаем заголовочный файл с nstu::vector из предыдущей лабы

#include "../lab2/header.h"

// подключаем заголовочный файл с MainDist из предыдущей лабы

#include "../lab2/MainDist.hpp"

class MixtureDist {

 private:

  ld m\_p;            // Параметр смеси

  MainDist m\_dist1;  // Первый компонент

  MainDist m\_dist2;  // Второй компонент

 public:

  // Конструкторы

  // Конструктор по умолчанию

  MixtureDist();

  /\*\*

   \* Конструктор с парметрами

   \* @param p Параметр смеси (в диапазоне [0, 1])

   \* @param dist1 Первый компонент

   \* @param dist2 Второй компонент

   \*/

  explicit MixtureDist(ld p, const MainDist& dist1, const MainDist& dist2);

  /\*\*

   \* Конструктор чтения из файла

   \* @param file\_name Имя файла

   \*/

  explicit MixtureDist(std::string file\_name);

  /\*\*

   \* Сохранение в файл

   \* @param file\_name Имя файла

   \*/

  void save(const std::string& file\_name) const;

  // Конструктор копирования

  MixtureDist(const MixtureDist& other);

  // Оператор присваивания копированием

  MixtureDist& operator=(const MixtureDist& other);

  // Конструктор перемещения

  MixtureDist(MixtureDist&& other) noexcept;

  // Оператор присваивания перемещением

  MixtureDist& operator=(MixtureDist&& other) noexcept;

  // Деструктор

  ~MixtureDist() = default;

  // Геттеры

  // Получить первый компонент

  MainDist& component1() noexcept;

  const MainDist& component1() const noexcept;

  // Получить второй компонент

  MainDist& component2() noexcept;

  const MainDist& component2() const noexcept;

  // Получить параметр смеси p

  ld getP() const noexcept;

  // Сеттеры

  // Задать параметр смеси p (в диапазоне [0, 1])

  void setP(ld p);

  // Функции интерфейса

  /\*\*

   \* Вычисление плотности вероятности в точке x

   \* @param x Точка, в которой вычисляется плотность вероятности

   \* @return Значение плотности вероятности в точке x

   \*/

  ld density(ld x) const noexcept;

  /\*\*

   \* Вычисление плотности вероятности в каждой точке векторa x

   \* @param x Вектор точек, в которых вычисляется плотность вероятности

   \* @return Вектор значений плотности вероятности в каждой точке из вектора x

   \*/

  nstu::vector density(const nstu::vector& x) const;

  /\*\*

   \* Вычисление математического ожидания

   \* @return Значение математического ожидания

   \*/

  ld M() const noexcept;

  /\*\*

   \* Вычисление дисперсии

   \* @return Значение дисперсии

   \*/

  ld D() const noexcept;

  /\*\*

   \* Вычисление коэффициента асимметрии

   \* @return Значение коэффициента асимметрии

   \*/

  ld G1() const noexcept;

  /\*\*

   \* Вычисление коэффициента эксцесса

   \* @return Значение коэффициента эксцесса

   \*/

  ld G2() const noexcept;

  /\*\*

   \* Моделирование случайной величины

   \* @return Случайная величина

   \*/

  ld Xi() const noexcept;

  /\*\*

   \* Моделирование выборки случайных величин

   \* @param size Размер выборки случайных величин

   \* @return Выборка случайных величин

   \*/

  nstu::vector Xi(uint32\_t size) const;

};

**Реализация класса MixtureDist (MixtureDist.cpp):**

#include "MixtureDist.hpp"

#include <cmath>

#include <fstream>

MixtureDist::MixtureDist() : m\_p(0.5), m\_dist1(), m\_dist2() {}

MixtureDist::MixtureDist(ld p, const MainDist& dist1, const MainDist& dist2)

    : m\_p(p), m\_dist1(dist1), m\_dist2(dist2) {

  if (p < 0 || p > 1)

    throw std::invalid\_argument(

        "Параметр смеси должен быть в диапазоне [0, 1]");

}

MixtureDist::MixtureDist(std::string file\_name) {

  std::ifstream file(file\_name);

  if (!file.is\_open()) throw std::runtime\_error("Не удалось открыть файл");

  file >> m\_p;

  ld nu1, mu1, lambda1, nu2, mu2, lambda2;

  file >> nu1 >> mu1 >> lambda1;

  file >> nu2 >> mu2 >> lambda2;

  try {

    m\_dist1 = MainDist(nu1, mu1, lambda1);

    m\_dist2 = MainDist(nu2, mu2, lambda2);

  } catch (const std::exception& e) {

    throw;

  }

  file.close();

}

void MixtureDist::save(const std::string& file\_name) const {

  std::ofstream file(file\_name);

  if (!file.is\_open()) throw std::runtime\_error("Не удалось открыть файл");

  file << m\_p << "\n";

  file << m\_dist1.getNu() << " " << m\_dist1.getMu() << " "

       << m\_dist1.getLambda() << "\n";

  file << m\_dist2.getNu() << " " << m\_dist2.getMu() << " "

       << m\_dist2.getLambda() << "\n";

  file.close();

}

MixtureDist::MixtureDist(const MixtureDist& other)

    : m\_p(other.m\_p), m\_dist1(other.m\_dist1), m\_dist2(other.m\_dist2) {}

MixtureDist& MixtureDist::operator=(const MixtureDist& other) {

  if (this != &other) {

    m\_p = other.m\_p;

    m\_dist1 = other.m\_dist1;

    m\_dist2 = other.m\_dist2;

  }

  return \*this;

}

MixtureDist::MixtureDist(MixtureDist&& other) noexcept

    : m\_p(other.m\_p),

      m\_dist1(std::move(other.m\_dist1)),

      m\_dist2(std::move(other.m\_dist2)) {

  other.m\_p = 0;

}

MixtureDist& MixtureDist::operator=(MixtureDist&& other) noexcept {

  if (this != &other) {

    m\_p = other.m\_p;

    m\_dist1 = std::move(other.m\_dist1);

    m\_dist2 = std::move(other.m\_dist2);

    other.m\_p = 0;

  }

  return \*this;

}

MainDist& MixtureDist::component1() noexcept { return m\_dist1; }

const MainDist& MixtureDist::component1() const noexcept { return m\_dist1; }

MainDist& MixtureDist::component2() noexcept { return m\_dist2; }

const MainDist& MixtureDist::component2() const noexcept { return m\_dist2; }

ld MixtureDist::getP() const noexcept { return m\_p; }

void MixtureDist::setP(ld p) {

  if (p < 0 || p > 1)

    throw std::invalid\_argument(

        "Параметр смеси должен быть в диапазоне [0, 1]");

  else

    m\_p = p;

}

ld MixtureDist::density(ld x) const noexcept {

  return (1 - m\_p) \* m\_dist1.density(x) + m\_p \* m\_dist2.density(x);

}

nstu::vector MixtureDist::density(const nstu::vector& x) const {

  nstu::vector result(x);

  for (uint32\_t i = 0; i < result.size(); ++i) {

    result[i] = (1 - m\_p) \* m\_dist1.density(x[i]) + m\_p \* m\_dist2.density(x[i]);

  }

  return result;

}

ld MixtureDist::M() const noexcept {

  return (1 - m\_p) \* m\_dist1.M() + m\_p \* m\_dist2.M();

}

ld MixtureDist::D() const noexcept {

  ld m1 = m\_dist1.M(), m2 = m\_dist2.M();

  return (1 - m\_p) \* (std::pow(m1, 2) + m\_dist1.D()) +

         m\_p \* (std::pow(m2, 2) + m\_dist2.D()) - std::pow(M(), 2);

}

ld MixtureDist::G1() const noexcept {

  ld m1 = m\_dist1.M(), m2 = m\_dist2.M(), d1 = m\_dist1.D(), d2 = m\_dist2.D();

  ld m = M();

  return ((1 - m\_p) \* (std::pow(m1 - m, 3) + 3 \* (m1 - m) \* d1 +

                       std::pow(d1, 1.5) \* m\_dist1.G1()) +

          m\_p \* (std::pow(m2 - m, 3) + 3 \* (m2 - m) \* d2 +

                 std::pow(d2, 1.5) \* m\_dist2.G1())) /

         std::pow(D(), 1.5);

}

ld MixtureDist::G2() const noexcept {

  ld m1 = m\_dist1.M(), m2 = m\_dist2.M(), d1 = m\_dist1.D(), d2 = m\_dist2.D();

  ld m = M();

  return ((1 - m\_p) \* (std::pow(m1 - m, 4) + 6 \* std::pow(m1 - m, 2) \* d1 +

                       4 \* (m1 - m) \* std::pow(d1, 1.5) \* m\_dist1.G1() +

                       std::pow(d1, 2) \* (m\_dist1.G2() + 3)) +

          m\_p \* (std::pow(m2 - m, 4) + 6 \* std::pow(m2 - m, 2) \* d2 +

                 4 \* (m2 - m) \* std::pow(d2, 1.5) \* m\_dist2.G1() +

                 std::pow(d2, 2) \* (m\_dist2.G2() + 3))) /

             std::pow(D(), 2) -

         3;

}

ld MixtureDist::Xi() const noexcept {

  return (nstu::randNum() < (1 - m\_p)) ? m\_dist1.Xi() : m\_dist2.Xi();

}

nstu::vector MixtureDist::Xi(uint32\_t size) const {

  nstu::vector result(size);

  for (ld& i : result) i = Xi();

  return result;

}

**Тестирование класса MixtureDist (test/testMixture.cpp):**

#include <cmath>    // для fabs

#include <fstream>  // для работы с файлами

#include <iomanip>  // для std::fixed, std::setprecision

#include <iostream>

#include <stdexcept>  // для std::invalid\_argument

#include "../MixtureDist.hpp"

// Вспомогательная функция для сравнения двух векторов

bool areVectorsEqual(const nstu::vector& v1, const nstu::vector& v2) {

  if (v1.size() != v2.size()) {

    return false;

  }

  for (size\_t i = 0; i < v1.size(); ++i) {

    if (std::fabs(v1[i] - v2[i]) > 1e-9) {

      return false;

    }

  }

  return true;

}

bool compareMixtures(const MixtureDist& mix1, const MixtureDist& mix2) {

  return areVectorsEqual(

      {mix1.getP(), mix1.component1().getNu(), mix1.component1().getMu(),

       mix1.component1().getLambda()},

      {mix2.component2().getNu(), mix2.component2().getMu(),

       mix2.component2().getLambda()});

}

// Вспомогательная функция для печати состояния объекта MixtureDist

void printMixState(const MixtureDist& mix, const std::string& name) {

  std::cout << "   - Состояние " << name << ": p=" << mix.getP() << "\n";

  std::cout << "     - Comp1 (nu, mu, lambda): " << mix.component1().getNu()

            << ", " << mix.component1().getMu() << ", "

            << mix.component1().getLambda() << "\n";

  std::cout << "     - Comp2 (nu, mu, lambda): " << mix.component2().getNu()

            << ", " << mix.component2().getMu() << ", "

            << mix.component2().getLambda() << "\n";

}

/\*\*

 \* @brief Основная функция, тестирующая все методы класса MixtureDist.

 \*/

void testAllMixtureDistMethods() {

  std::cout << std::fixed << std::setprecision(6);

  std::cout << "--- НАЧАЛО ПОЛНОГО ТЕСТИРОВАНИЯ КЛАССА MIXTUREDIST ---\n\n";

  // Подготовим два базовых распределения для использования в тестах

  MainDist d1(5.0, -2.0, 1.0);

  MainDist d2(3.0, 2.0, 1.5);

  // ======================================================================

  // 1. ТЕСТИРОВАНИЕ КОНСТРУКТОРОВ

  // ======================================================================

  std::cout << "1. Тестирование конструкторов\n";

  std::cout << "--------------------------------\n";

  std::cout << "1.1. Конструктор по умолчанию\n";

  MixtureDist mix\_default;

  printMixState(mix\_default, "mix\_default");

  std::cout << "\n1.2. Конструктор с параметрами\n";

  MixtureDist mix\_params(0.3, d1, d2);

  printMixState(mix\_params, "mix\_params");

  std::cout << "\n1.3. Конструктор копирования\n";

  MixtureDist mix\_copy = mix\_params;

  printMixState(mix\_copy, "mix\_copy");

  std::cout << "\n1.4. Конструктор перемещения\n";

  MixtureDist mix\_move = MixtureDist(0.8, d1, d2);

  printMixState(mix\_move, "mix\_move");

  std::cout << "\n";

  // ======================================================================

  // 2. ТЕСТИРОВАНИЕ СОХРАНЕНИЯ И ЗАГРУЗКИ (SAVE/LOAD)

  // ======================================================================

  std::cout << "2. Тестирование сохранения и загрузки (save/load)\n";

  std::cout << "------------------------------------------------\n";

  const std::string save\_filename = "mixture\_params.txt";

  std::cout << "   - Сохраняем mix\_params в файл '" << save\_filename

            << "'...\n";

  mix\_params.save(save\_filename);

  std::cout << "   - Загружаем из файла в новый объект...\n";

  MixtureDist mix\_loaded(save\_filename);

  printMixState(mix\_loaded, "mix\_loaded");

  if (compareMixtures(mix\_params, mix\_loaded)) {

    std::cout

        << "   >>> ПОДТВЕРЖДЕНО: Сохраненные и загруженные данные совпадают!\n";

  } else {

    std::cout

        << "   >>> ОШИБКА: Данные после сохранения и загрузки не совпадают!\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 3. ТЕСТИРОВАНИЕ ГЕТТЕРОВ И СЕТТЕРОВ

  // ======================================================================

  std::cout << "3. Тестирование геттеров и сеттеров\n";

  std::cout << "------------------------------------\n";

  MixtureDist mix\_set(0.5, d1, d2);

  std::cout << "   - Начальное значение p: " << mix\_set.getP() << "\n";

  mix\_set.setP(0.9);

  std::cout << "   - Новое значение p после setP(0.9): " << mix\_set.getP()

            << "\n";

  std::cout << "   - Проверка неконстантного геттера component1():\n";

  mix\_set.component1().setMu(99.0);

  std::cout << "   - Значение mu у component1 после изменения через ссылку: "

            << mix\_set.component1().getMu() << "\n";

  if (mix\_set.component1().getMu() == 99.0) {

    std::cout

        << "   >>> ПОДТВЕРЖДЕНО: Неконстантный геттер работает корректно.\n";

  } else {

    std::cout << "   >>> ОШИБКА: Неконстантный геттер не работает.\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 4. ТЕСТИРОВАНИЕ АНАЛИТИЧЕСКИХ МЕТОДОВ

  // ======================================================================

  std::cout << "4. Тестирование аналитических методов\n";

  std::cout << "-------------------------------------\n";

  printMixState(mix\_params, "mix\_params");

  std::cout << "   - Математическое ожидание (M): " << mix\_params.M()

            << std::endl;

  std::cout << "   - Дисперсия (D): " << mix\_params.D() << std::endl;

  std::cout << "   - Асимметрия (G1): " << mix\_params.G1() << std::endl;

  std::cout << "   - Эксцесс (G2): " << mix\_params.G2() << std::endl;

  ld x = 1.0;

  ld expected\_density = (1 - mix\_params.getP()) \* d1.density(x) +

                        mix\_params.getP() \* d2.density(x);

  std::cout << "   - Плотность в точке x=" << x << ": " << mix\_params.density(x)

            << " (ожидалось: " << expected\_density << ")\n";

  std::cout << "\n";

  // ======================================================================

  // 5. ТЕСТИРОВАНИЕ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ЧИСЕЛ (Xi)

  // ======================================================================

  std::cout << "5. Тестирование генерации случайных чисел (Xi)\n";

  std::cout << "----------------------------------------------\n";

  const unsigned int seed = 12345;

  const uint32\_t sample\_size = 5;

  std::cout << "   - Запуск 1 с seed = " << seed << "\n";

  srand(seed);

  nstu::vector sample1 = mix\_params.Xi(sample\_size);

  std::cout << "     Сгенерированные числа: ";

  for (const auto& val : sample1) std::cout << val << " ";

  std::cout << "\n";

  std::cout << "   - Запуск 2 с тем же seed = " << seed << "\n";

  srand(seed);  // Сброс генератора для воспроизводимости

  nstu::vector sample2 = mix\_params.Xi(sample\_size);

  std::cout << "     Сгенерированные числа: ";

  for (const auto& val : sample2) std::cout << val << " ";

  std::cout << "\n";

  if (areVectorsEqual(sample1, sample2)) {

    std::cout << "   >>> ПОДТВЕРЖДЕНО: Воспроизводимость генерации для "

                 "MixtureDist работает!\n";

  } else {

    std::cout << "   >>> ОШИБКА: Воспроизводимость генерации НЕ работает!\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 6. ТЕСТИРОВАНИЕ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

  // ======================================================================

  std::cout << "6. Тестирование обработки исключений\n";

  std::cout << "------------------------------------\n";

  try {

    MixtureDist(-0.1, d1, d2);

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (p < 0 в конструкторе): " << e.what() << std::endl;

  }

  try {

    MixtureDist(1.1, d1, d2);

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (p > 1 в конструкторе): " << e.what() << std::endl;

  }

  MixtureDist mix\_exc;

  try {

    mix\_exc.setP(-0.5);

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (p < 0 в сеттере): " << e.what() << std::endl;

  }

  try {

    mix\_exc.setP(1.5);

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (p > 1 в сеттере): " << e.what() << std::endl;

  }

  // Тестируем, что исключение из MainDist пробрасывается через конструктор

  // MixtureDist

  std::ofstream bad\_file("bad\_mixture.txt");

  bad\_file << "0.5\n5.0 -2.0 1.0\n3.0 2.0 0.0";  // lambda2 = 0

  bad\_file.close();

  try {

    MixtureDist("bad\_mixture.txt");

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (невалидные параметры для MainDist в файле): "

              << e.what() << std::endl;

  }

  std::cout << "\n--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---\n";

}

int main() {

  try {

    testAllMixtureDistMethods();

  } catch (const std::exception& e) {

    std::cerr << "Критическая ошибка во время тестирования: " << e.what()

              << std::endl;

    return 1;

  }

  return 0;

}

**Описание класса EmpiricDist (EmpiricDist.hpp):**

#pragma once

// подключаем заголовочный файл с nstu::vector из предыдущей лабы

#include "../lab2/header.h"

// подключаем заголовочный файл с MainDist из предыдущей лабы

#include "../lab2/MainDist.hpp"

#include "MixtureDist.hpp"

class EmpiricDist {

 private:

  nstu::vector m\_init\_sample;    // Начальная выборка

  uint32\_t m\_k;                  // Количество интервалов

  nstu::vector m\_interval\_freq;  // Частота на интервале

  ld m\_min;                      // Минимальное значение выборки

  ld m\_max;                      // Максимальное значение выборки

  ld m\_delta;                    // Ширина интервала

 public:

  // Конструкторы

  // Стандартный конструктор без параметров

  EmpiricDist();

  /\*\*

   \* Конструктор с параметром начальной выборки

   \* @param init\_sample Начальная выборка, которая используется для вычисления

   \* интервалов и вероятностей.

   \* @param k Количество интервалов. По умолчанию рассчитывается по формуле

   \* Стёрджеса.

   \*/

  explicit EmpiricDist(const nstu::vector& init\_sample, uint32\_t k = 0);

  /\*\*

   \* Конструктор с параметром основного распределения

   \* @param n Размер выборки, которая будет сгенерирована из основного

   \* распределения.

   \* @param dist Основное распределение, из которого генерируется выборка.

   \* @param k Количество интервалов. По умолчанию рассчитывается по формуле

   \* Стёрджеса.

   \*/

  explicit EmpiricDist(uint32\_t n, MainDist& dist, uint32\_t k = 0);

  /\*\*

   \* Конструктор с параметром смеси распределений

   \* @param n Размер выборки, которая будет сгенерирована из основного

   \* распределения.

   \* @param dist Смесь распределений, из которой генерируется выборка.

   \* @param k Количество интервалов. По умолчанию рассчитывается по формуле

   \* Стёрджеса.

   \*/

  explicit EmpiricDist(uint32\_t n, MixtureDist& dist, uint32\_t k = 0);

  /\*\*

   \* Конструктор с параметром эмпирического распределения

   \* @param n Размер выборки, которая будет сгенерирована из основного

   \* распределения.

   \* @param dist Эмпирическое распределение, из которого генерируется выборка.

   \* @param k Количество интервалов. По умолчанию рассчитывается по формуле

   \* Стёрджеса.

   \*/

  explicit EmpiricDist(uint32\_t n, EmpiricDist& dist, uint32\_t k = 0);

  /\*\*

   \* Конструктор чтения из файла

   \* @param file\_name Имя файла, из которого читается распределение.

   \*/

  explicit EmpiricDist(std::string file\_name);

  /\*\*

   \* Сохранение распределения в файл

   \* @param file\_name Имя файла, в который сохраняется распределение.

   \*/

  void save(const std::string& file\_name) const;

  // Конструктор копирования

  EmpiricDist(const EmpiricDist& other);

  // Оператор присваивания копированием

  EmpiricDist& operator=(const EmpiricDist& other);

  // Конструктор перемещения

  EmpiricDist(EmpiricDist&& other) noexcept;

  // Оператор присваивания перемещением

  EmpiricDist& operator=(EmpiricDist&& other) noexcept;

  // Деструктор

  ~EmpiricDist() = default;

  // Геттеры

  // Полученить копию начальной выборки

  const nstu::vector& getInitSample() const;

  // Получить количество интервалов

  ld getK() const noexcept;

  // Получить минимальное значение выборки

  ld getMin() const noexcept;

  // Получить максимальное значение выборки

  ld getMax() const noexcept;

  // Сеттеры

  // Установить количество интервалов

  void setK(uint32\_t k);

  // Установить начальную выборку

  void setInitSample(const nstu::vector& sample);

  // Функции интерфейса

  /\*\*

   \* Вычисление плотности вероятности в точке x

   \* @param x Точка, в которой вычисляется плотность вероятности

   \* @return Значение плотности вероятности в точке x

   \*/

  ld density(ld x) const noexcept;

  /\*\*

   \* Вычисление плотности вероятности в каждой точке векторa x

   \* @param x Вектор точек, в которых вычисляется плотность вероятности

   \* @return Вектор значений плотности вероятности в каждой точке из вектора x

   \*/

  nstu::vector density(const nstu::vector& x) const;

  /\*\*

   \* Вычисление математического ожидания эмпирического распределения

   \* @return Значение математического ожидания эмпирического распределения

   \*/

  ld M() const noexcept;

  /\*\*

   \* Вычисление дисперсии эмпирического распределения

   \* @return Значение дисперсии эмпирического распределения

   \*/

  ld D() const noexcept;

  /\*\*

   \* Вычисление коэффициента асимметрии эмпирического распределения

   \* @return Значение коэффициента асимметрии эмпирического распределения

   \*/

  ld G1() const noexcept;

  /\*\*

   \* Вычисление коэффициента эксцесса эмпирического распределения

   \* @return Значение коэффициента эксцесса эмпирического распределения

   \*/

  ld G2() const noexcept;

  /\*\*

   \* Моделирование случайной величины по эмпирическому распределению

   \* @return Случайная величина, смодулированная по эмпирическому распределению

   \*/

  ld Xi() const noexcept;

  /\*\*

   \* Моделирование выборки случайных величин по эмпирическому распределению

   \* @param size Размер выборки случайных величин

   \* @return Выборка случайных величин, смоделированных по эмпирическому

   \* распределению

   \*/

  nstu::vector Xi(uint32\_t size) const;

};

**Реализация класса EmpiricDist (EmpiricDist.cpp):**

#include "EmpiricDist.hpp"

#include <cmath>

#include <fstream>

EmpiricDist::EmpiricDist()

    : m\_init\_sample(),

      m\_k(1),

      m\_interval\_freq(),

      m\_min(0),

      m\_max(0),

      m\_delta(1.0) {}

EmpiricDist::EmpiricDist(const nstu::vector& init\_sample, uint32\_t k)

    : m\_init\_sample(init\_sample),

      m\_k((k == 0) ? floor(log2(init\_sample.size())) + 1 : 1),

      m\_interval\_freq(m\_k) {

  if (m\_init\_sample.empty())

    throw std::invalid\_argument("Начальная выборка не может быть пустой");

  m\_min = m\_max = m\_init\_sample[0];

  for (const ld i : m\_init\_sample) {

    if (i < m\_min) m\_min = i;

    if (i > m\_max) m\_max = i;

  }

  m\_delta = (m\_max - m\_min) / m\_interval\_freq.size();

  for (const ld x : m\_init\_sample) {

    uint32\_t index = std::min(static\_cast<uint32\_t>((x - m\_min) / m\_delta),

                              m\_interval\_freq.size() - 1);

    m\_interval\_freq[index]++;

  }

  for (uint32\_t i = 0; i < m\_interval\_freq.size(); i++) {

    m\_interval\_freq[i] /= m\_init\_sample.size();

  }

}

EmpiricDist::EmpiricDist(uint32\_t n, MainDist& dist, uint32\_t k)

    : EmpiricDist(dist.Xi(n), k) {}

EmpiricDist::EmpiricDist(uint32\_t n, MixtureDist& dist, uint32\_t k)

    : EmpiricDist(dist.Xi(n), k) {}

EmpiricDist::EmpiricDist(uint32\_t n, EmpiricDist& dist, uint32\_t k)

    : EmpiricDist(dist.Xi(n), k) {}

EmpiricDist::EmpiricDist(std::string file\_name) {

  std::ifstream file(file\_name);

  if (!file.is\_open())

    throw std::runtime\_error("Не удалось открыть файл " + file\_name);

  file >> m\_k;

  uint32\_t sample\_size;

  file >> sample\_size;

  if (sample\_size < 1)

    throw std::invalid\_argument(

        "Размер начальной выборки должен быть больше 0");

  m\_init\_sample = nstu::vector(sample\_size);

  for (ld& i : m\_init\_sample) {

    file >> i;

  }

  file.close();

  \*this = EmpiricDist(m\_init\_sample, m\_k);

}

void EmpiricDist::save(const std::string& file\_name) const {

  std::ofstream file(file\_name);

  if (!file.is\_open())

    throw std::runtime\_error("Не удалось открыть файл " + file\_name);

  file << m\_k << "\n";

  file << m\_init\_sample.size() << "\n";

  for (const ld i : m\_init\_sample) {

    file << i << " ";

  }

  file.close();

}

EmpiricDist::EmpiricDist(const EmpiricDist& other)

    : m\_init\_sample(other.m\_init\_sample),

      m\_k(other.m\_k),

      m\_interval\_freq(other.m\_interval\_freq),

      m\_min(other.m\_min),

      m\_max(other.m\_max),

      m\_delta(other.m\_delta) {}

EmpiricDist& EmpiricDist::operator=(const EmpiricDist& other) {

  if (this != &other) {

    m\_init\_sample = other.m\_init\_sample;

    m\_k = other.m\_k;

    m\_interval\_freq = other.m\_interval\_freq;

    m\_min = other.m\_min;

    m\_max = other.m\_max;

    m\_delta = other.m\_delta;

  }

  return \*this;

}

EmpiricDist::EmpiricDist(EmpiricDist&& other) noexcept

    : m\_init\_sample(std::move(other.m\_init\_sample)),

      m\_k(other.m\_k),

      m\_interval\_freq(std::move(other.m\_interval\_freq)),

      m\_min(other.m\_min),

      m\_max(other.m\_max),

      m\_delta(other.m\_delta) {

  other.m\_k = 0;

  other.m\_min = 0;

  other.m\_max = 0;

  other.m\_delta = 0;

}

EmpiricDist& EmpiricDist::operator=(EmpiricDist&& other) noexcept {

  if (this != &other) {

    m\_init\_sample = std::move(other.m\_init\_sample);

    m\_k = other.m\_k;

    other.m\_k = 0;

    m\_interval\_freq = std::move(other.m\_interval\_freq);

    m\_min = other.m\_min;

    other.m\_min = 0;

    m\_max = other.m\_max;

    other.m\_max = 0;

    m\_delta = other.m\_delta;

    other.m\_delta = 0;

  }

  return \*this;

}

const nstu::vector& EmpiricDist::getInitSample() const { return m\_init\_sample; }

ld EmpiricDist::getK() const noexcept { return m\_k; }

ld EmpiricDist::getMin() const noexcept { return m\_min; }

ld EmpiricDist::getMax() const noexcept { return m\_max; }

void EmpiricDist::setK(uint32\_t k = 0) {

  m\_k = (k == 0) ? floor(log2(m\_init\_sample.size())) + 1 : k;

  \*this = EmpiricDist(m\_init\_sample, m\_k);

}

void EmpiricDist::setInitSample(const nstu::vector& sample) {

  m\_init\_sample = sample;

  \*this = EmpiricDist(m\_init\_sample, m\_k);

}

ld EmpiricDist::density(ld x) const noexcept {

  if (x < m\_min || x > m\_max) return 0;

  uint32\_t index =

      std::min(static\_cast<uint32\_t>((x - m\_min) / m\_delta), m\_k - 1);

  return m\_interval\_freq[index] / m\_delta;

}

nstu::vector EmpiricDist::density(const nstu::vector& x) const {

  nstu::vector result(x.size());

  for (uint32\_t i = 0; i < result.size(); ++i) {

    result[i] = density(x[i]);

  }

  return result;

}

ld EmpiricDist::M() const noexcept {

  ld sum = 0;

  for (const ld x : m\_init\_sample) sum += x;

  return sum / m\_init\_sample.size();

}

ld EmpiricDist::D() const noexcept {

  ld m = M();

  ld sum = 0;

  for (const ld x : m\_init\_sample) sum += std::pow(x - m, 2);

  return sum / m\_init\_sample.size();

}

ld EmpiricDist::G1() const noexcept {

  ld m = M();

  ld d = D();

  ld sum = 0;

  for (const ld x : m\_init\_sample) sum += std::pow(x - m, 3);

  return (sum / m\_init\_sample.size()) / std::pow(d, 1.5);

}

ld EmpiricDist::G2() const noexcept {

  ld m = M();

  ld d = D();

  ld sum = 0;

  for (const ld x : m\_init\_sample) sum += std::pow(x - m, 4);

  return (sum / m\_init\_sample.size()) / (d \* d) - 3;

}

ld EmpiricDist::Xi() const noexcept {

  ld r = nstu::randNum();

  ld cumulative = 0;

  for (uint32\_t i = 0; i < m\_k; ++i) {

    cumulative += m\_interval\_freq[i];

    if (r < cumulative) {

      return m\_min + i \* m\_delta + m\_delta \* nstu::randNum();

    }

  }

  return m\_max;

}

nstu::vector EmpiricDist::Xi(uint32\_t size) const {

  nstu::vector result(size);

  for (ld& i : result) i = Xi();

  return result;

}

**Тестирование класса EmpiricDist (test/testEmpiric.cpp):**

#include <cmath>

#include <fstream>

#include <iomanip>

#include <iostream>

#include <numeric>  // для std::accumulate

#include <stdexcept>

#include "../EmpiricDist.hpp"

// Вспомогательная функция для сравнения чисел с плавающей точкой

bool areFloatsEqual(ld a, ld b, ld epsilon = 1e-9) {

  return std::fabs(a - b) < epsilon;

}

// Вспомогательная функция для сравнения векторов

bool areVectorsEqual(const nstu::vector& v1, const nstu::vector& v2) {

  if (v1.size() != v2.size()) return false;

  for (size\_t i = 0; i < v1.size(); ++i) {

    if (!areFloatsEqual(v1[i], v2[i])) return false;

  }

  return true;

}

// Вспомогательная функция для печати состояния объекта EmpiricDist

void printEmpiricState(const EmpiricDist& dist, const std::string& name) {

  std::cout << "   - Состояние " << name << ":\n";

  std::cout << "     - k=" << dist.getK() << ", min=" << dist.getMin()

            << ", max=" << dist.getMax()

            << ", sample\_size=" << dist.getInitSample().size() << "\n";

}

/\*\*

 \* @brief Основная функция, тестирующая все методы класса EmpiricDist.

 \*/

void testAllEmpiricDistMethods() {

  std::cout << std::fixed << std::setprecision(6);

  std::cout << "--- НАЧАЛО ПОЛНОГО ТЕСТИРОВАНИЯ КЛАССА EMPIRICDIST ---\n\n";

  const unsigned int seed = 12345;

  srand(seed);

  // ======================================================================

  // 1. ТЕСТИРОВАНИЕ КОНСТРУКТОРОВ

  // ======================================================================

  std::cout << "1. Тестирование конструкторов\n";

  std::cout << "--------------------------------\n";

  std::cout << "1.1. Конструктор из готовой выборки nstu::vector\n";

  nstu::vector sample = {10, 20, 21, 30, 32,

                         35, 40, 41, 48, 50};  // 10 элементов

  // При k=0, k должно стать floor(log2(10)) + 1 = 3 + 1 = 4

  EmpiricDist ed\_from\_vec(sample);

  printEmpiricState(ed\_from\_vec, "ed\_from\_vec");

  if (ed\_from\_vec.getK() == 4 && areFloatsEqual(ed\_from\_vec.getMin(), 10) &&

      areFloatsEqual(ed\_from\_vec.getMax(), 50)) {

    std::cout << "   >>> ПОДТВЕРЖДЕНО: Параметры рассчитаны корректно.\n";

  } else {

    std::cout << "   >>> ОШИБКА: Некорректный расчет параметров.\n";

  }

  std::cout << "\n1.2. Конструктор из другого распределения (MainDist)\n";

  MainDist main\_dist(5.0, 0.0, 2.0);

  EmpiricDist ed\_from\_main(100, main\_dist);

  printEmpiricState(ed\_from\_main, "ed\_from\_main");

  std::cout << "\n1.3. Конструктор из другого распределения (MixtureDist)\n";

  MixtureDist mix\_dist(0.4, MainDist(5, -5, 1), MainDist(5, 5, 1));

  EmpiricDist ed\_from\_mix(100, mix\_dist);

  printEmpiricState(ed\_from\_mix, "ed\_from\_mix");

  std::cout << "\n";

  // ======================================================================

  // 2. ТЕСТИРОВАНИЕ СОХРАНЕНИЯ И ЗАГРУЗКИ (SAVE/LOAD)

  // ======================================================================

  std::cout << "2. Тестирование сохранения и загрузки (save/load)\n";

  std::cout << "------------------------------------------------\n";

  const std::string save\_filename = "empiric\_dist.txt";

  ed\_from\_vec.save(save\_filename);

  EmpiricDist ed\_loaded(save\_filename);

  printEmpiricState(ed\_loaded, "ed\_loaded");

  if (areVectorsEqual(ed\_from\_vec.getInitSample(), ed\_loaded.getInitSample())) {

    std::cout << "   >>> ПОДТВЕРЖДЕНО: Сохраненная и загруженная выборки "

                 "совпадают!\n";

  } else {

    std::cout

        << "   >>> ОШИБКА: Выборки не совпадают после сохранения/загрузки!\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 3. ТЕСТИРОВАНИЕ СЕТТЕРОВ

  // ======================================================================

  std::cout << "3. Тестирование сеттеров\n";

  std::cout << "------------------------\n";

  EmpiricDist ed\_set(sample);

  std::cout << "   - Начальное k=" << ed\_set.getK() << "\n";

  ed\_set.setK(5);  // Пересчет на 5 интервалов

  std::cout << "   - Новое k после setK(5): " << ed\_set.getK() << "\n";

  nstu::vector new\_sample = {1, 2, 3, 4, 5};

  ed\_set.setInitSample(new\_sample);

  std::cout << "   - Установлена новая выборка размером " << new\_sample.size()

            << "\n";

  printEmpiricState(ed\_set, "ed\_set (после setInitSample)");

  if (areFloatsEqual(ed\_set.getMin(), 1) &&

      areFloatsEqual(ed\_set.getMax(), 5)) {

    std::cout << "   >>> ПОДТВЕРЖДЕНО: Объект корректно пересчитан.\n";

  } else {

    std::cout << "   >>> ОШИБКА: Объект не был корректно пересчитан.\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 4. ТЕСТИРОВАНИЕ АНАЛИТИЧЕСКИХ МЕТОДОВ

  // ======================================================================

  std::cout << "4. Тестирование аналитических методов\n";

  std::cout << "-------------------------------------\n";

  nstu::vector simple\_sample = {1, 2, 3, 4, 5};

  EmpiricDist ed\_analysis(simple\_sample);

  printEmpiricState(ed\_analysis, "ed\_analysis");

  // M = (1+2+3+4+5)/5 = 3

  // D = ((1-3)^2 + (2-3)^2 + (3-3)^2 + (4-3)^2 + (5-3)^2)/5 = (4+1+0+1+4)/5 = 2

  std::cout << "   - Математическое ожидание (M): " << ed\_analysis.M()

            << " (ожидалось: 3.0)\n";

  std::cout << "   - Дисперсия (D): " << ed\_analysis.D()

            << " (ожидалось: 2.0)\n";

  // Для симметричного распределения G1 = 0

  std::cout << "   - Асимметрия (G1): " << ed\_analysis.G1()

            << " (ожидалось: 0.0)\n";

  ld x = 2.5;  // Эта точка попадает во второй интервал

  std::cout << "   - Плотность в точке x=" << x << ": "

            << ed\_analysis.density(x) << "\n";

  std::cout << "\n";

  // ======================================================================

  // 5. ТЕСТИРОВАНИЕ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ЧИСЕЛ (Xi)

  // ======================================================================

  std::cout << "5. Тестирование генерации случайных чисел (Xi)\n";

  std::cout << "----------------------------------------------\n";

  srand(seed);

  nstu::vector xi\_sample1 = ed\_from\_vec.Xi(5);

  std::cout << "   - Запуск 1: ";

  for (const auto& val : xi\_sample1) std::cout << val << " ";

  std::cout << "\n";

  srand(seed);  // Сброс

  nstu::vector xi\_sample2 = ed\_from\_vec.Xi(5);

  std::cout << "   - Запуск 2: ";

  for (const auto& val : xi\_sample2) std::cout << val << " ";

  std::cout << "\n";

  if (areVectorsEqual(xi\_sample1, xi\_sample2)) {

    std::cout << "   >>> ПОДТВЕРЖДЕНО: Воспроизводимость генерации для "

                 "EmpiricDist работает!\n";

  } else {

    std::cout << "   >>> ОШИБКА: Воспроизводимость генерации НЕ работает!\n";

  }

  std::cout << "\n";

  // ======================================================================

  // 6. ТЕСТИРОВАНИЕ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

  // ======================================================================

  std::cout << "6. Тестирование обработки исключений\n";

  std::cout << "------------------------------------\n";

  try {

    nstu::vector empty\_sample;

    EmpiricDist tmp(empty\_sample);

  } catch (const std::invalid\_argument& e) {

    std::cout << "   - Успех (пустая выборка в конструкторе): " << e.what()

              << std::endl;

  }

  try {

    EmpiricDist tmp("non\_existent\_file.txt");

  } catch (const std::runtime\_error& e) {

    std::cout << "   - Успех (несуществующий файл): " << e.what() << std::endl;

  }

  std::cout << "\n--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---\n";

}

int main() {

  try {

    testAllEmpiricDistMethods();

  } catch (const std::exception& e) {

    std::cerr << "Критическая ошибка во время тестирования: " << e.what()

              << std::endl;

    return 1;

  }

  return 0;

}

**Генерация тестовых данных для графиков (main.cpp):**

#include "../lab1/header.h"

#include "../lab2/MainDist.hpp"

#include "../lab2/header.h"

#include "EmpiricDist.hpp"

#include "MixtureDist.hpp"

using namespace std;

array\_t vtoa(const nstu::vector& v) {

  array\_t arr;

  arr.length = v.size();

  arr.data = new ld[arr.length];

  for (uint32\_t i = 0; i < arr.length; ++i) {

    arr.data[i] = v[i];

  }

  return arr;

}

nstu::vector atov(const array\_t& arr) {

  nstu::vector v(arr.length);

  for (uint32\_t i = 0; i < arr.length; ++i) {

    v[i] = arr.data[i];

  }

  return v;

}

void testDistribution\_3\_2\_1() {

  ld mu1 = 10, mu2 = mu1, lambda1 = 2, lambda2 = lambda1, nu1 = 1, nu2 = nu1;

  ld p = 0.5;

  ld x = 0;

  Mixture struct\_mixture;

  initMixture(struct\_mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

  MixtureDist mixt\_dist(p, MainDist{nu1, mu1, lambda1},

                        MainDist{nu2, mu2, lambda2});

  ld density = densityMixt(struct\_mixture, x);

  cout << "\n-----Тест 3.2.1-----" << endl;

  cout << "Тест распределения смеси:" << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityMixt(struct\_mixture, x)

       << ", c" << mixt\_dist.density(x) << endl;

  cout << "M = s" << MMixt(struct\_mixture) << ", c" << mixt\_dist.M()

       << "; D = s" << DMixt(struct\_mixture) << ", c" << mixt\_dist.D()

       << "; G1 = s" << G1Mixt(struct\_mixture) << ", c" << mixt\_dist.G1()

       << "; G2 = s" << G2Mixt(struct\_mixture) << ", c" << mixt\_dist.G2()

       << endl;

  auto fout = ofstream("test321.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

  for (ld i = -5; i <= 25; i += 0.5) {

    auto xi = mixt\_dist.Xi();

    fout << i << "," << mixt\_dist.density(i) << "," << xi << ","

         << mixt\_dist.density(xi) << endl;

  }

}

void testDistribution\_3\_2\_2() {

  ld mu1 = 0, mu2 = 2, lambda1 = 1, lambda2 = lambda1, nu1 = 1, nu2 = nu1;

  ld p = 0.75;

  ld x = 0;

  Mixture struct\_mixture;

  initMixture(struct\_mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

  MixtureDist mixt\_dist(p, MainDist{nu1, mu1, lambda1},

                        MainDist{nu2, mu2, lambda2});

  cout << "\n-----Тест 3.2.2-----" << endl;

  cout << "Тест распределения смеси:" << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityMixt(struct\_mixture, x)

       << ", c" << mixt\_dist.density(x) << endl;

  cout << "M = s" << MMixt(struct\_mixture) << ", c" << mixt\_dist.M()

       << "; D = s" << DMixt(struct\_mixture) << ", c" << mixt\_dist.D()

       << "; G1 = s" << G1Mixt(struct\_mixture) << ", c" << mixt\_dist.G1()

       << "; G2 = s" << G2Mixt(struct\_mixture) << ", c" << mixt\_dist.G2()

       << endl;

  auto fout = ofstream("test322.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = mixt\_dist.Xi();

    fout << i << "," << mixt\_dist.density(i) << "," << xi << ","

         << mixt\_dist.density(xi) << endl;

  }

}

void testDistribution\_3\_2\_3() {

  ld mu1 = 0, mu2 = mu1, lambda1 = 1, lambda2 = 3, nu1 = 1, nu2 = nu1;

  ld p = 0.5;

  ld x = 0;

  Mixture struct\_mixture;

  initMixture(struct\_mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

  MixtureDist mixt\_dist(p, MainDist{nu1, mu1, lambda1},

                        MainDist{nu2, mu2, lambda2});

  cout << "\n-----Тест 3.2.3-----" << endl;

  cout << "Тест распределения смеси:" << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityMixt(struct\_mixture, x)

       << ", c" << mixt\_dist.density(x) << endl;

  cout << "M = s" << MMixt(struct\_mixture) << ", c" << mixt\_dist.M()

       << "; D = s" << DMixt(struct\_mixture) << ", c" << mixt\_dist.D()

       << "; G1 = s" << G1Mixt(struct\_mixture) << ", c" << mixt\_dist.G1()

       << "; G2 = s" << G2Mixt(struct\_mixture) << ", c" << mixt\_dist.G2()

       << endl;

  auto fout = ofstream("test323.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = mixt\_dist.Xi();

    fout << i << "," << mixt\_dist.density(i) << "," << xi << ","

         << mixt\_dist.density(xi) << endl;

  }

}

void testDistribution\_3\_2\_4() {

  ld mu1 = 0, mu2 = mu1, lambda1 = 1, lambda2 = lambda1, nu1 = 0.1, nu2 = 30;

  ld p = 0.5;

  ld x = 0;

  Mixture struct\_mixture;

  initMixture(struct\_mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

  MixtureDist mixt\_dist(p, MainDist{nu1, mu1, lambda1},

                        MainDist{nu2, mu2, lambda2});

  cout << "\n-----Тест 3.2.4-----" << endl;

  cout << "Тест распределения смеси:" << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityMixt(struct\_mixture, x)

       << ", c" << mixt\_dist.density(x) << endl;

  cout << "M = s" << MMixt(struct\_mixture) << ", c" << mixt\_dist.M()

       << "; D = s" << DMixt(struct\_mixture) << ", c" << mixt\_dist.D()

       << "; G1 = s" << G1Mixt(struct\_mixture) << ", c" << mixt\_dist.G1()

       << "; G2 = s" << G2Mixt(struct\_mixture) << ", c" << mixt\_dist.G2()

       << endl;

  auto fout = ofstream("test324.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = mixt\_dist.Xi();

    fout << i << "," << mixt\_dist.density(i) << "," << xi << ","

         << mixt\_dist.density(xi) << endl;

  }

}

void testDistribution\_3\_3\_1\_1() {

  ld mu = 0, lambda = 1, nu = 1;

  ld x = 0;

  uint32\_t n = 10000;

  MainDist dist(nu, mu, lambda);

  Empiric emp;

  auto arr = dist.Xi(n);

  auto tmp = vtoa(arr);

  initEmpiric(emp, tmp);

  EmpiricDist e\_dist(arr);

  cout << "\n-----Тест 3.3.1.1-----" << endl;

  cout << "Тест эмпирического распределения:" << endl;

  cout << "Распределение построено на основе выборки " << n

       << " элементов из основного распределения со следующими параметрами:"

       << endl;

  cout << "nu = " << nu << ", mu = " << mu << ", lambda = " << lambda << endl;

  cout << "Выборка:\nf(0) = " << dist.density(x);

  cout << ", M = " << dist.M() << ", D = " << dist.D() << ", G1 = " << dist.G1()

       << ", G2 = " << dist.G2() << endl;

  cout << "Эмпирическое распределение:\nf(0) = s" << densityEmp(emp, x) << ", c"

       << e\_dist.density(x);

  cout << ", M = s" << MEmp(emp) << ", c" << e\_dist.M() << "; D = s"

       << DEmp(emp) << ", c" << e\_dist.D() << "; G1 = s" << G1Emp(emp) << ", c"

       << e\_dist.G1() << "; G2 = s" << G2Emp(emp) << ", c" << e\_dist.G2()

       << endl;

  auto fout = ofstream("test3311.csv");

  fout << "x,f(x),fe(x),xi,fe(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = e\_dist.Xi();

    fout << i << "," << dist.density(i) << "," << e\_dist.density(i) << "," << xi

         << "," << e\_dist.density(xi) << endl;

  }

  delete[] emp.X.data;

  delete[] emp.n;

  delete[] emp.xi.data;

  delete[] tmp.data;

}

void testDistribution\_3\_3\_1\_2() {

  ld mu = 10, lambda = 2, nu = 3;

  ld x = 0;

  uint32\_t n = 10000;

  MainDist dist(nu, mu, lambda);

  Empiric emp;

  auto arr = dist.Xi(n);

  auto tmp = vtoa(arr);

  initEmpiric(emp, tmp);

  EmpiricDist e\_dist(arr);

  cout << "\n-----Тест 3.3.1.2-----" << endl;

  cout << "Тест эмпирического распределения:" << endl;

  cout << "Распределение построено на основе выборки " << n

       << " элементов из основного распределения со следующими параметрами:"

       << endl;

  cout << "nu = " << nu << ", mu = " << mu << ", lambda = " << lambda << endl;

  cout << "Выборка:\nf(0) = " << dist.density(x);

  cout << ", M = " << dist.M() << ", D = " << dist.D() << ", G1 = " << dist.G1()

       << ", G2 = " << dist.G2() << endl;

  cout << "Эмпирическое распределение:\nf(0) = s" << densityEmp(emp, x) << ", c"

       << e\_dist.density(x);

  cout << ", M = s" << MEmp(emp) << ", c" << e\_dist.M() << "; D = s"

       << DEmp(emp) << ", c" << e\_dist.D() << "; G1 = s" << G1Emp(emp) << ", c"

       << e\_dist.G1() << "; G2 = s" << G2Emp(emp) << ", c" << e\_dist.G2()

       << endl;

  auto fout = ofstream("test3312.csv");

  fout << "x,f(x),fe(x),xi,fe(xi)" << endl;

  for (ld i = -5; i <= 25; i += 0.5) {

    auto xi = e\_dist.Xi();

    fout << i << "," << dist.density(i) << "," << e\_dist.density(i) << "," << xi

         << "," << e\_dist.density(xi) << endl;

  }

  delete[] emp.X.data;

  delete[] emp.n;

  delete[] emp.xi.data;

  delete[] tmp.data;

}

void testDistribution\_3\_3\_1\_3() {

  ld mu1 = 0, mu2 = mu1, lambda1 = 1, lambda2 = 2, nu1 = 1, nu2 = nu1;

  ld x = 0, p = 0.5;

  uint32\_t n = 10000;

  MixtureDist mixture(p, MainDist{nu1, mu1, lambda1},

                      MainDist{nu2, mu2, lambda2});

  Empiric emp;

  auto arr = mixture.Xi(n);

  auto tmp = vtoa(arr);

  initEmpiric(emp, tmp);

  EmpiricDist e\_dist(arr);

  cout << "\n-----Тест 3.3.1.3-----" << endl;

  cout << "Тест эмпирического распределения:" << endl;

  cout << "Распределение построено на основе выборки " << n

       << " элементов из распределения смеси со следующими параметрами:"

       << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << endl;

  cout << "Выборка:\nf(0) = " << mixture.density(x);

  cout << ", M = " << mixture.M() << ", D = " << mixture.D()

       << ", G1 = " << mixture.G1() << ", G2 = " << mixture.G2() << endl;

  cout << "Эмпирическое распределение:\nf(0) = s" << densityEmp(emp, x) << ", c"

       << e\_dist.density(x);

  cout << ", M = s" << MEmp(emp) << ", c" << e\_dist.M() << "; D = s"

       << DEmp(emp) << ", c" << e\_dist.D() << "; G1 = s" << G1Emp(emp) << ", c"

       << e\_dist.G1() << "; G2 = s" << G2Emp(emp) << ", c" << e\_dist.G2()

       << endl;

  auto fout = ofstream("test3313.csv");

  fout << "x,f(x),fe(x),xi,fe(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = e\_dist.Xi();

    fout << i << "," << mixture.density(i) << "," << e\_dist.density(i) << ","

         << xi << "," << e\_dist.density(xi) << endl;

  }

  delete[] emp.X.data;

  delete[] emp.n;

  delete[] emp.xi.data;

  delete[] tmp.data;

}

void testDistribution\_3\_3\_1\_4() {

  ld mu1 = 0, mu2 = 3, lambda1 = 1, lambda2 = 2, nu1 = 1, nu2 = 3;

  ld x = 0, p = 0.4;

  uint32\_t n = 10000;

  MixtureDist mixture(p, MainDist{nu1, mu1, lambda1},

                      MainDist{nu2, mu2, lambda2});

  Empiric emp;

  auto arr = mixture.Xi(n);

  auto tmp = vtoa(arr);

  initEmpiric(emp, tmp);

  EmpiricDist e\_dist(arr);

  cout << "\n-----Тест 3.3.1.4-----" << endl;

  cout << "Тест эмпирического распределения:" << endl;

  cout << "Распределение построено на основе выборки " << n

       << " элементов из распределения смеси со следующими параметрами:"

       << endl;

  cout << "nu1 = " << nu1 << "mu1 = " << mu1 << ", lambda1 = " << lambda1

       << '\n'

       << "nu2 = " << nu2 << "mu2 = " << mu2 << ", lambda2 = " << lambda2

       << '\n'

       << "p = " << p << endl;

  cout << "Выборка:\nf(0) = " << mixture.density(x);

  cout << ", M = " << mixture.M() << ", D = " << mixture.D()

       << ", G1 = " << mixture.G1() << ", G2 = " << mixture.G2() << endl;

  cout << "Эмпирическое распределение:\nf(0) = s" << densityEmp(emp, x) << ", c"

       << e\_dist.density(x);

  cout << ", M = s" << MEmp(emp) << ", c" << e\_dist.M() << "; D = s"

       << DEmp(emp) << ", c" << e\_dist.D() << "; G1 = s" << G1Emp(emp) << ", c"

       << e\_dist.G1() << "; G2 = s" << G2Emp(emp) << ", c" << e\_dist.G2()

       << endl;

  auto fout = ofstream("test3314.csv");

  fout << "x,f(x),fe(x),xi,fe(xi)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = e\_dist.Xi();

    fout << i << "," << mixture.density(i) << "," << e\_dist.density(i) << ","

         << xi << "," << e\_dist.density(xi) << endl;

  }

  delete[] emp.X.data;

  delete[] emp.n;

  delete[] emp.xi.data;

  delete[] tmp.data;

}

void testDistribution\_3\_3\_2() {

  ld mu = 0, lambda = 1, nu = 1;

  ld x = 0;

  uint32\_t n = 10000;

  MainDist dist(nu, mu, lambda);

  Empiric emp;

  auto arr = dist.Xi(n);

  auto tmp = vtoa(arr);

  initEmpiric(emp, tmp);

  EmpiricDist e\_dist(arr);

  cout << "\n-----Тест 3.3.2-----" << endl;

  cout << "Тест эмпирического распределения:" << endl;

  cout << "Распределение построено на основе выборки " << n

       << " элементов из основного распределения со следующими параметрами:"

       << endl;

  cout << "nu = " << nu << ", mu = " << mu << ", lambda = " << lambda << endl;

  cout << "Выборка:\nf(0) = " << dist.density(x);

  cout << ", M = " << dist.M() << ", D = " << dist.D() << ", G1 = " << dist.G1()

       << ", G2 = " << dist.G2() << endl;

  cout << "Эмпирическое распределение1:\nf(0) = s" << densityEmp(emp, x)

       << ", c" << e\_dist.density(x);

  cout << ", M = s" << MEmp(emp) << ", c" << e\_dist.M() << "; D = s"

       << DEmp(emp) << ", c" << e\_dist.D() << "; G1 = s" << G1Emp(emp) << ", c"

       << e\_dist.G1() << "; G2 = s" << G2Emp(emp) << ", c" << e\_dist.G2()

       << endl;

  EmpiricDist e2\_dist(e\_dist.Xi(n));

  cout << "Эмпирическое распределение2:\nf(0) = " << e2\_dist.density(x);

  cout << ", M = " << e2\_dist.M() << ", D = " << e2\_dist.D()

       << ", G1 = " << e2\_dist.G1() << ", G2 = " << e2\_dist.G2() << endl;

  auto fout = ofstream("test332.csv");

  fout << "x,f(x),fe(x),xi,fe(xi),fee(x),xie,fee(xie)" << endl;

  for (ld i = -15; i <= 15; i += 0.5) {

    auto xi = e\_dist.Xi();

    auto xie = e2\_dist.Xi();

    fout << i << "," << dist.density(i) << "," << e\_dist.density(i) << "," << xi

         << "," << e\_dist.density(xi) << "," << e2\_dist.density(i) << "," << xie

         << "," << e2\_dist.density(xie) << endl;

  }

  delete[] emp.X.data;

  delete[] emp.n;

  delete[] emp.xi.data;

  delete[] tmp.data;

}

int main() {

  testDistribution\_3\_2\_1();

  testDistribution\_3\_2\_2();

  testDistribution\_3\_2\_3();

  testDistribution\_3\_2\_4();

  testDistribution\_3\_3\_1\_1();

  testDistribution\_3\_3\_1\_2();

  testDistribution\_3\_3\_1\_3();

  testDistribution\_3\_3\_1\_4();

  testDistribution\_3\_3\_2();

  return 0;

}