**Цель работы**

Изучить реализацию на языке С++ отношений между классами и понятие интерфейса.

**Постановка задачи**

1. Ознакомиться с понятиями, связанными с отношениями между классами (в том числе с механизмом позднего связывания), и понятием интерфейса.
2. 2. Модифицировать программу, разработанную в лабораторных работах №2, 3, так чтобы в ней были определены интерфейс распределения и интерфейс персистентного объекта, а класс смеси был шаблоном.

2.1. Определить в интерфейсе распределения следующие чисто виртуальные функции: – функция для вычисления значений плотности распределения по заданному аргументу; – функция для вычисления математического ожидания, дисперсии, коэффициентов асимметрии и эксцесса; – функция для моделирования случайной величины.

2.2. Определить в интерфейсе персистентного объекта чисто виртуальные функции, обеспечивающие персистентность.

2.3. Классы, разработанные в работах № 2, 3, должны быть реализациями интерфейсов, кроме того класс смеси необходимо сделать шаблоном, параметризованным типами каждого из агрегированных объектовраспределений (отношение инстанцирования).

2.4. В классе эмпирического распределения заменить все конструкторы, получающие ссылки на объекты разных классов с целью инициализации массива данных, единым конструктором, получающим ссылку на интерфейс распределения (отношение зависимости класса от интерфейса).

2.5. Объекты всегда должны находиться в корректном состоянии. При возникновении ошибок необходимо генерировать исключительные ситуации, а их обработку осуществлять вне функций-членов классов. Ввод-вывод следует осуществлять вне функций-членов классов.

**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 интерфейса**

**Интерфейс распределения – IDist**:

* Чисто виртуальные публичные методы:
  + 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

**Интерфейс персистентности – IPersistent**:

* Чисто виртуальные публичные методы:
  + Save(file\_name) – сохраняет состояние объекта в файл с именем file\_name.
  + Save(file) – сохраняет состояние объекта в файловом потоке file (удобно, в случае вызова метода дочернего объекта, к примеру смесь распределений также вызывает метод save из двух своих компонент, и не открывает/закрывает при этом файл 3 раза.
  + Load(file\_name) – читает состояние объекта из файла и изменяет состояние объекта на то, что в файле file\_name.
  + Load(file) – загружает состояние объекта из файлового потока file.

**Алгоритм**

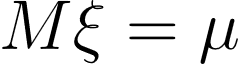
Расчет **основного** распределения:

Расчет плотности симметрического гиперболического распределения осуществляется по формуле:

 (1.1)

С учетом сдвига и масштаба:

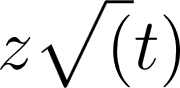
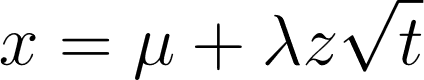
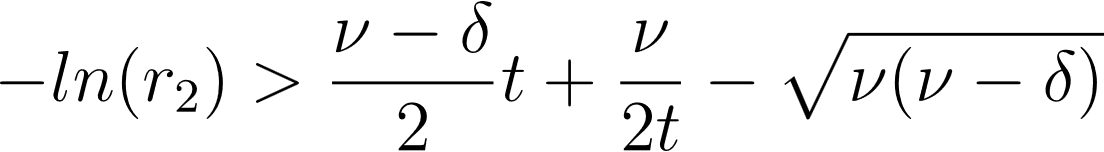
(1.2)

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)

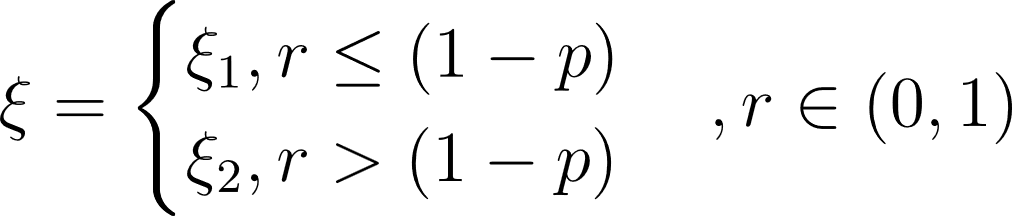
Коэффициент асимметрии:

 (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)

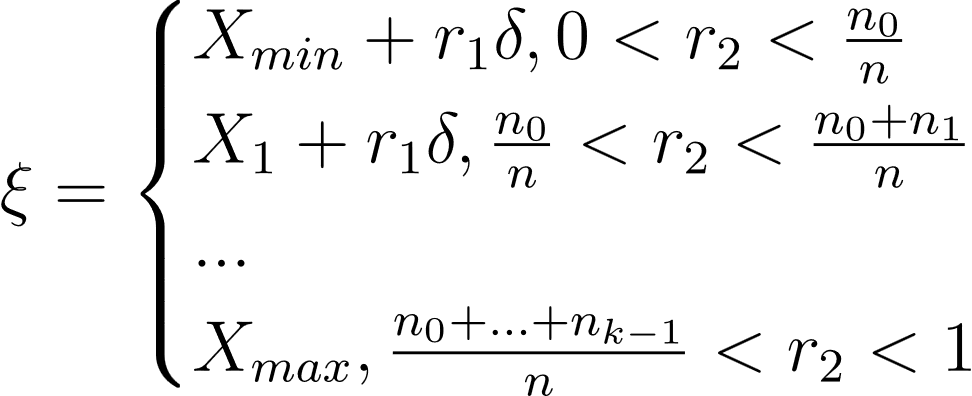
Коэффициент эксцесса:

 (3.5)

Коэффициент эксцесса:

 (3.6)

Случайная величина:

 (3.7)

**Тестирование**

3.1. Минимальный набор тестов для основного распределения:

3.1.1) тест для стандартного распределения: μ=0, λ=1, ν = 1;

3.1.2) тест для масштабных преобразований: μ=0, λ=2, ν = 1.

3.1.3) тест для сдвиг-масштабных преобразований: μ=10, λ=2, ν = 1.

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 3.1.1-----

Тест основного распределения:

nu = 1, mu = 0, lambda = 1, x = 0

Плотность вероятности f(x): 0.305595

M = 0; D = 2.69948; G1 = 0; G2 = 1.85697

-----Тест 3.1.2-----

Тест основного распределения:

nu = 1, mu = 0, lambda = 2, x = 0

Плотность вероятности f(x): 0.152797

M = 0; D = 10.7979; G1 = 0; G2 = 1.85697

-----Тест 3.1.3-----

Тест основного распределения:

nu = 1, mu = 10, lambda = 2, x = 0

Плотность вероятности f(x): 0.00253475

M = 10; D = 10.7979; G1 = 0; G2 = 1.85697

-----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): 0.00253475

M = 10; D = 10.7979; G1 = 0; G2 = 1.85697

-----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): 0.142986

M = 1.5; D = 3.44948; G1 = -0.117066; G2 = 1.10574

-----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): 0.20373

M = 0; D = 13.4974; G1 = 0; G2 = 4.96544

-----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): 0.269628

M = 0; D = 10.6484; G1 = 0; G2 = 7.74183

-----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) = 0.23437, M = -0.00178498; D = 2.74961; G1 = -0.059497; G2 = 2.05714

-----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) = 0.000519893, M = 9.9819; D = 5.94034; G1 = 0.00605621; G2 = 0.854818

-----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) = 0.182739, M = 0.00828148; D = 6.75462; G1 = 0.100481; G2 = 4.39083

-----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) = 0.182477, M = 1.15306; D = 6.13926; G1 = 0.528948; G2 = 0.888189

-----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 распределение:

f(0) = 0.284684, M = 0.0108839; D = 2.68854; G1 = -0.0241399; G2 = 1.7531

Эмпирическое распределение2:

f(0) = 0.255681, M = 0.0171469, D = 2.97876, G1 = -0.142753, G2 = 1.49249

--- НАЧАЛО ПОЛНОГО ТЕСТИРОВАНИЯ КЛАССА MAINDIST ---

1. Тестирование конструкторов и геттеров

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

1.1. Конструктор с параметрами: MainDist(5.0, 1.0, 2.0)

- Состояние dist\_params (nu, mu, lambda): 5.000000, 1.000000, 2.000000

1.2. Конструктор из файла (test\_params.txt)

- Состояние dist\_file\_params (nu, mu, lambda): 5.000000, 1.000000, 2.000000

- Состояние dist\_file (nu, mu, lambda): 5.000000, 1.000000, 2.000000

1.3. Конструктор с initializer\_list: {5.0, 1.0, 2.0}

- Состояние dist\_list (nu, mu, lambda): 5.000000, 1.000000, 2.000000

1.4. Конструктор копирования: MainDist dist\_copy = dist\_params

- Состояние dist\_copy (nu, mu, lambda): 5.000000, 1.000000, 2.000000

Изменили nu у оригинала до 99.0

- Состояние dist\_params (измененный) (nu, mu, lambda): 99.000000, 1.000000, 2.000000

- Состояние dist\_copy (неизмененный) (nu, mu, lambda): 5.000000, 1.000000, 2.000000

1.5. Конструктор перемещения: MainDist dist\_move = MainDist(...)

- Состояние dist\_move (nu, mu, lambda): 10.000000, -5.000000, 1.000000

2. Тестирование операторов присваивания

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

2.1. Присваивание копированием

- Состояние assign\_target (до) (nu, mu, lambda): 1.000000, 1.000000, 1.000000

- Состояние assign\_source (nu, mu, lambda): 8.000000, 8.000000, 8.000000

- Состояние assign\_target (после) (nu, mu, lambda): 8.000000, 8.000000, 8.000000

2.2. Присваивание перемещением

- Состояние move\_target (до) (nu, mu, lambda): 1.000000, 1.000000, 1.000000

- Состояние move\_source (nu, mu, lambda): 9.000000, 9.000000, 9.000000

- Состояние move\_target (после) (nu, mu, lambda): 9.000000, 9.000000, 9.000000

- Состояние move\_source (после перемещения, обнулен) (nu, mu, lambda): 0.000000, 0.000000, 0.000000

3. Тестирование сеттеров

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

- Состояние dist\_set (начальное) (nu, mu, lambda): 1.000000, 1.000000, 1.000000

- Состояние dist\_set (конечное) (nu, mu, lambda): 10.000000, 20.000000, 30.000000

4. Тестирование аналитических методов

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

- Состояние dist\_analysis (nu, mu, lambda): 5.000000, 1.000000, 2.000000

- Математическое ожидание (M): 1.000000

- Дисперсия (D): 5.250384

- Асимметрия (G1): 0.000000

- Эксцесс (G2): 0.569679

- Плотность в точке x = 1.500000: 0.180541

- Плотности для вектора {0.5, 1.0, 1.5, 2.0}:

{ 0.180541 0.186254 0.180541 0.164619 }

5. Тестирование генерации случайных чисел (Xi)

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

- Одиночное случайное число Xi(): 2.294427

- Вектор Xi(5) [Запуск 1]: { 2.294427 6.649925 1.575349 -2.780684 3.789816 }

- Вектор Xi(5) [Запуск 2]: { 2.294427 6.649925 1.575349 -2.780684 3.789816 }

>>> ПОДТВЕРЖДЕНО: Воспроизводимость генерации работает!

6. Тестирование обработки исключений

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

- Успех: перехвачено исключение для lambda=0: Лямбда не может быть 0

- Успех: перехвачено исключение для неверного числа аргументов: Неверное количество аргументов

- Успех: перехвачено исключение для несуществующего файла: Не удалось открыть файл

- Успех: перехвачено исключение для неверного типа в файле: Неверный тип распределения

- Успех: перехвачено исключение для lambda=0 в файле: Лямбда не может быть 0

- Успех (setLambda(0)): Лямбда не может быть 0

- Успех (setNu(0)): Нулевое значение nu

--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---

--- 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): Неверный тип распределения

- 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

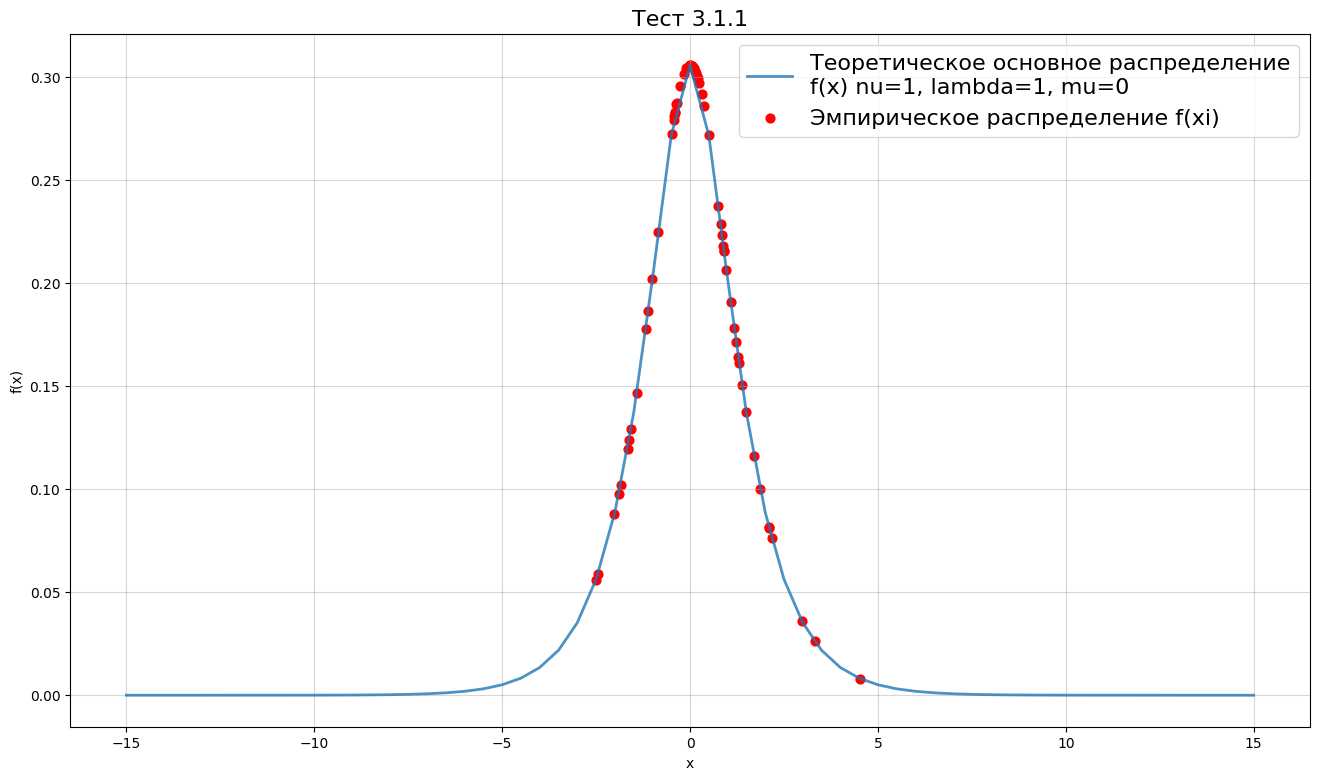
- Успех (несуществующий файл): Не удалось открыть файл non\_existent\_file.txt

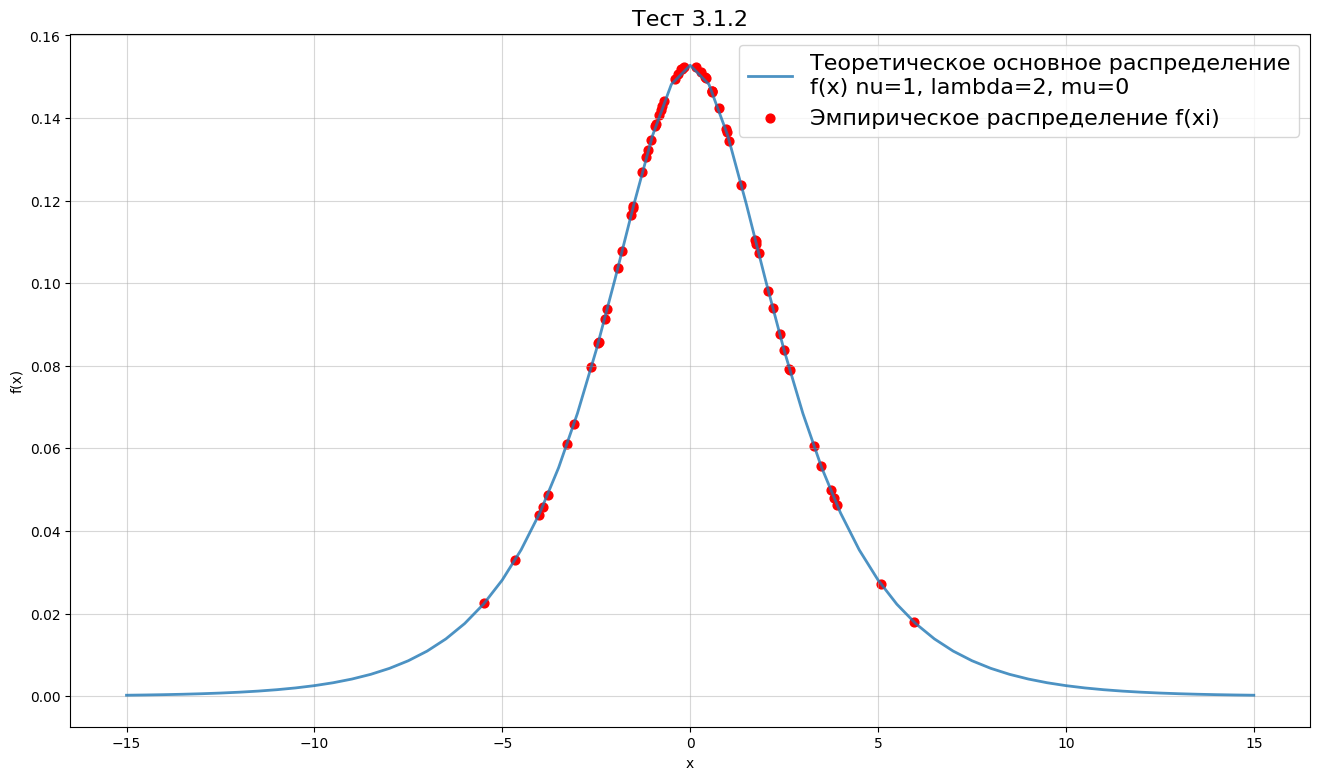
--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---

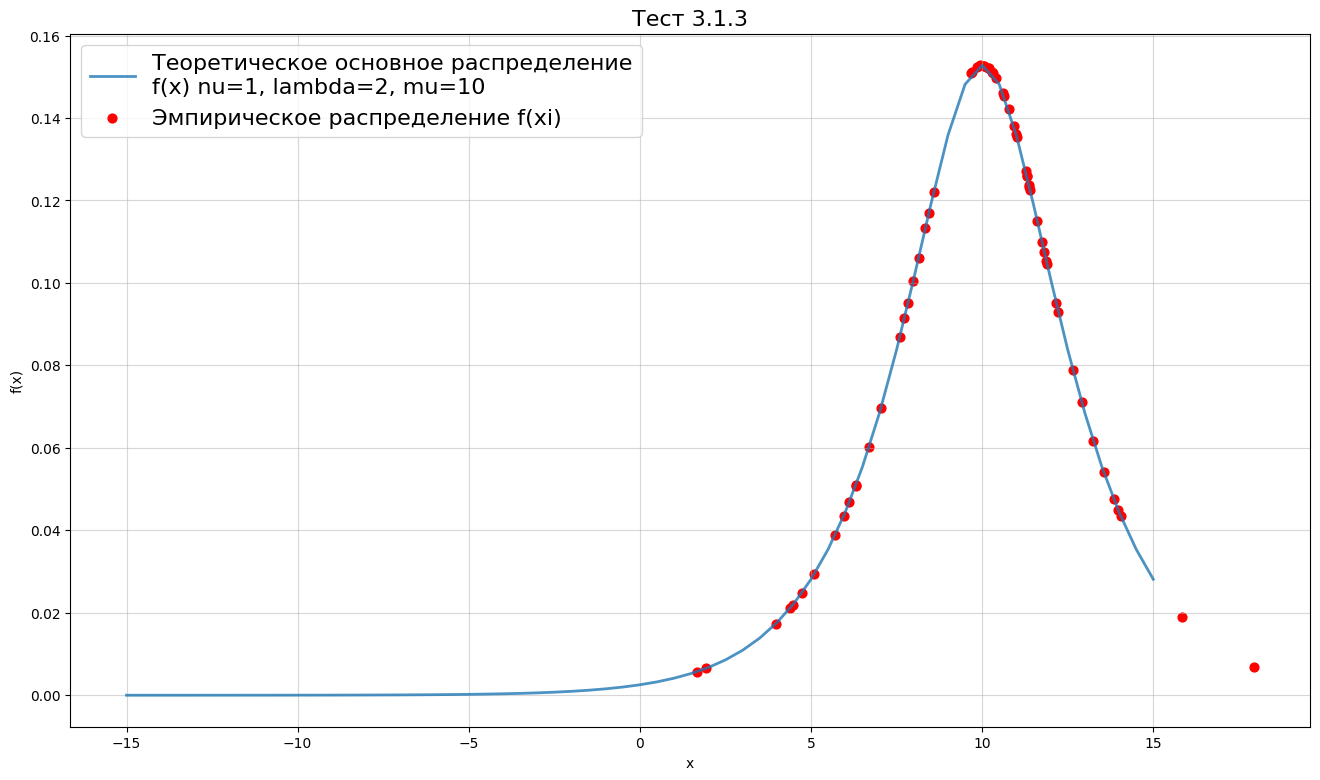
**Графики распределений**

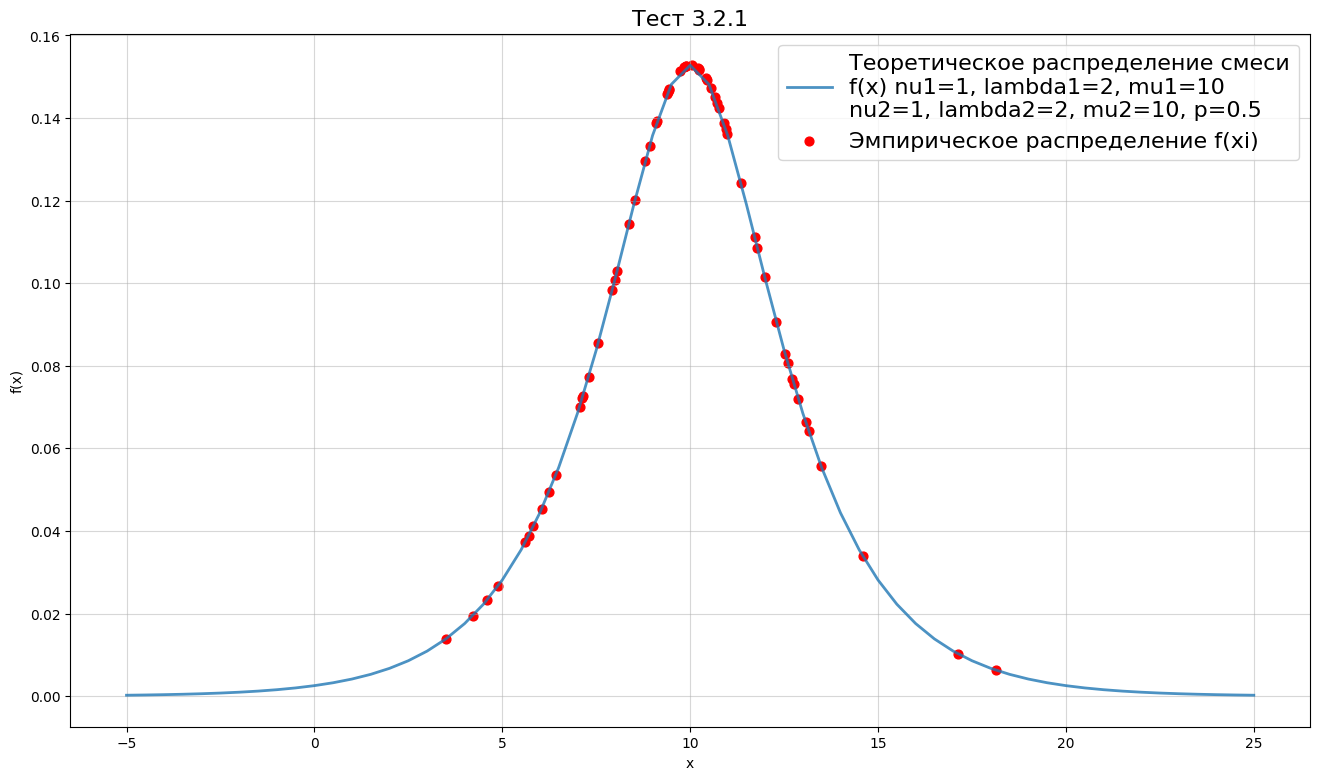
Все графики построены на выборке из 10000 элементов с помощью python.

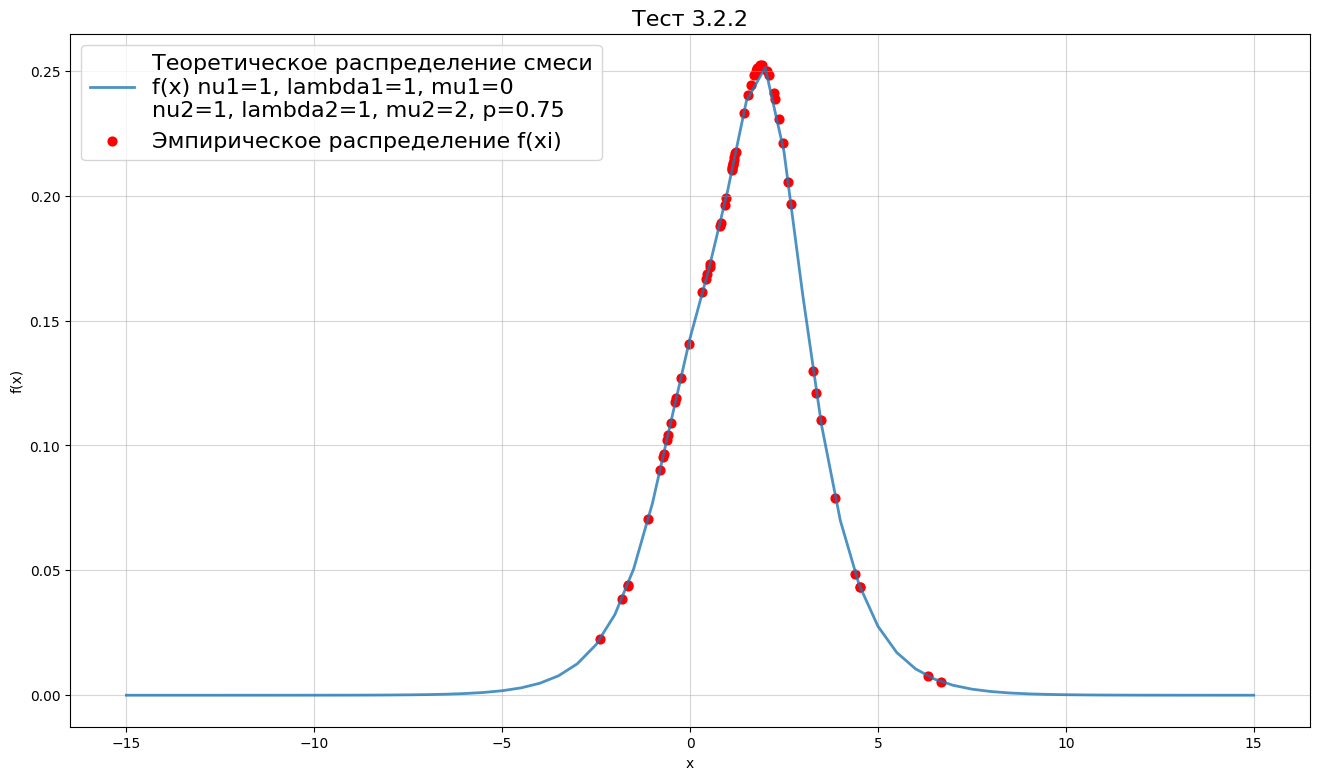
Графики тестов:

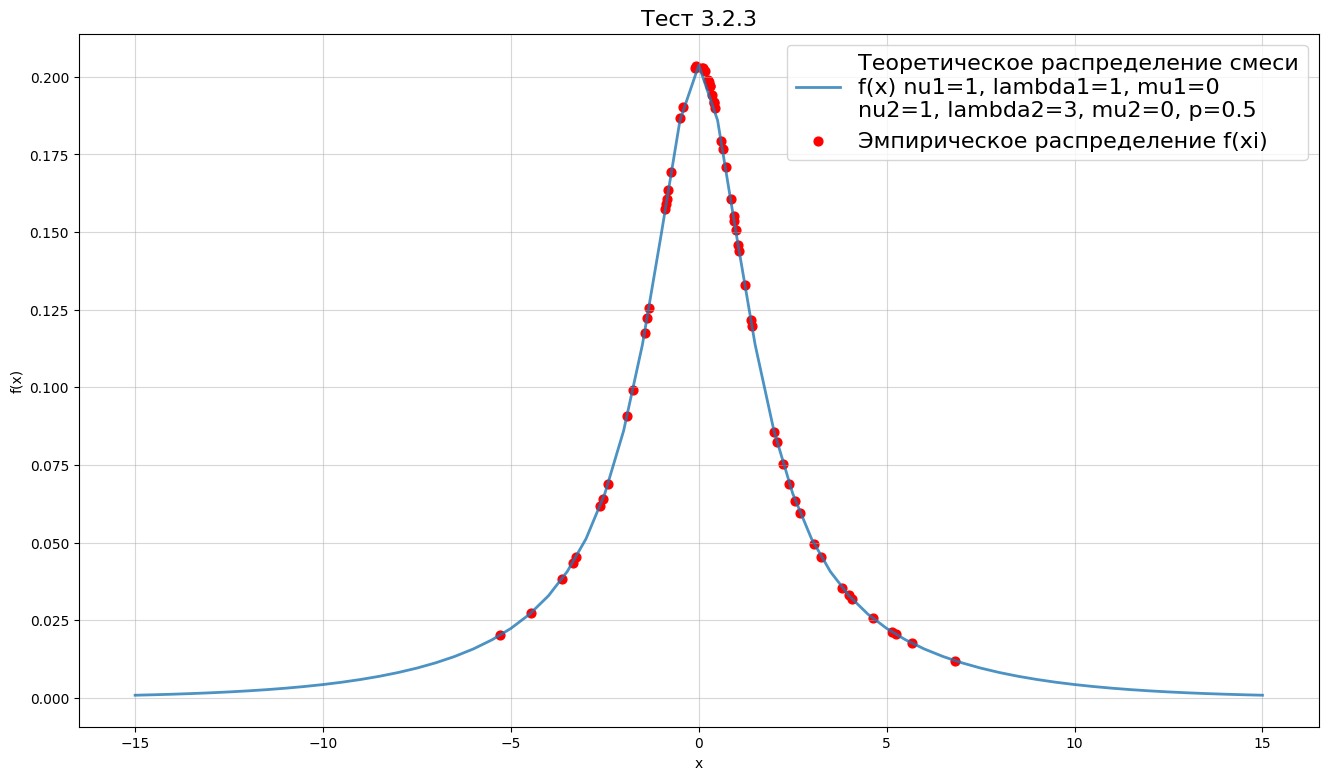
3.1.1

3.1.2

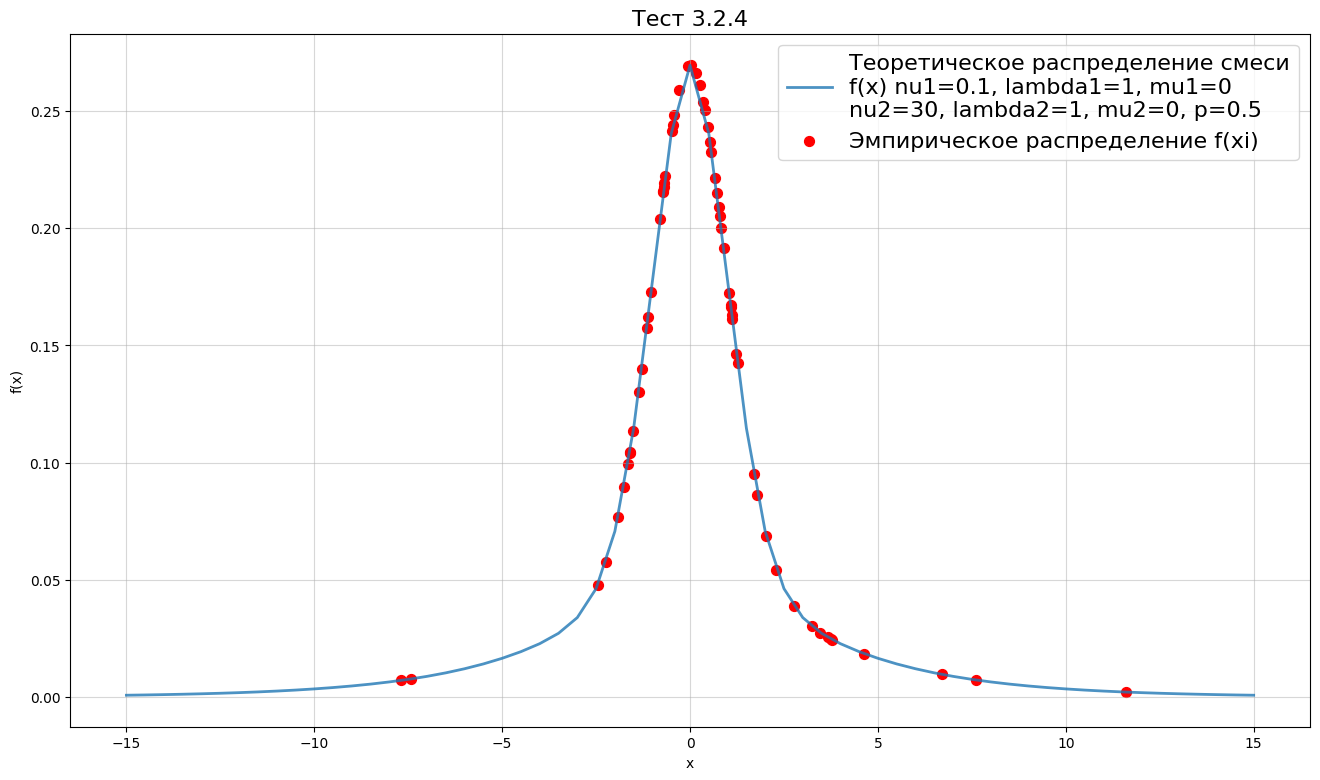
3.1.3

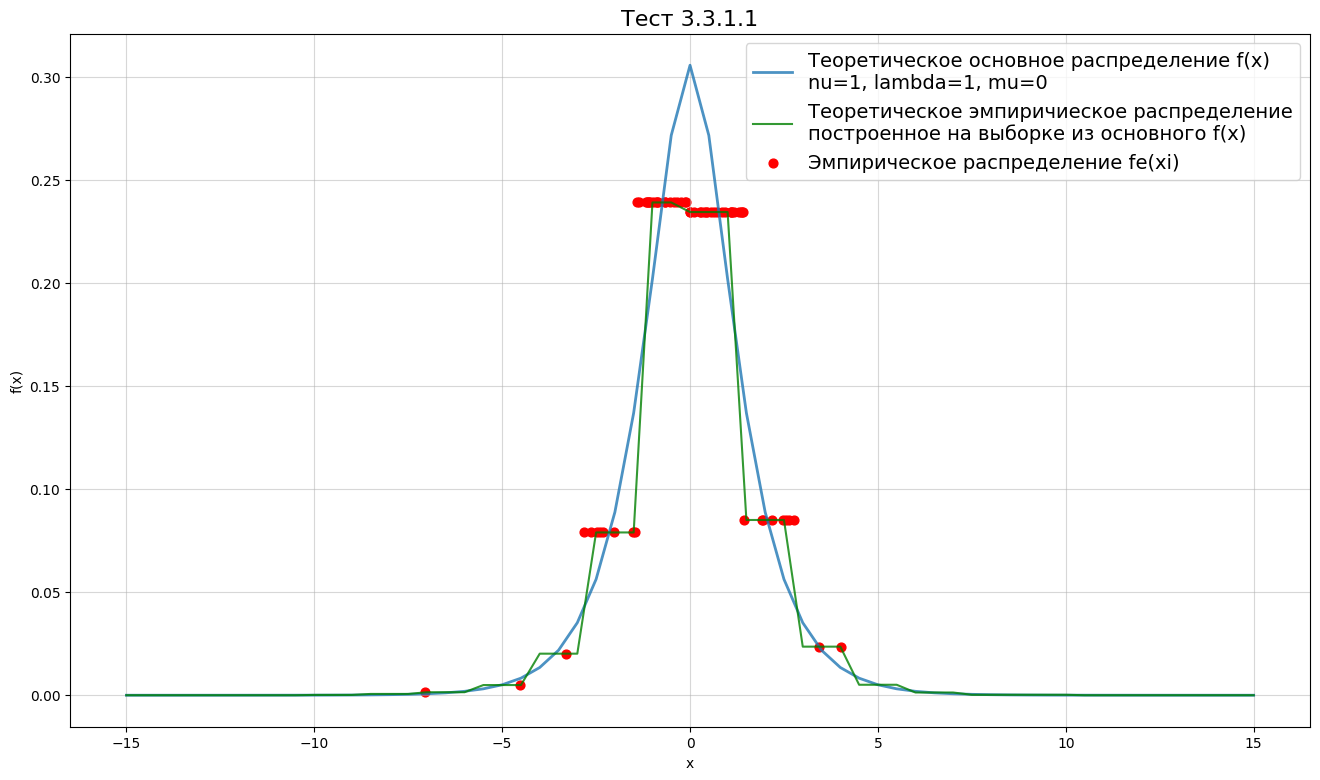
3.2.1

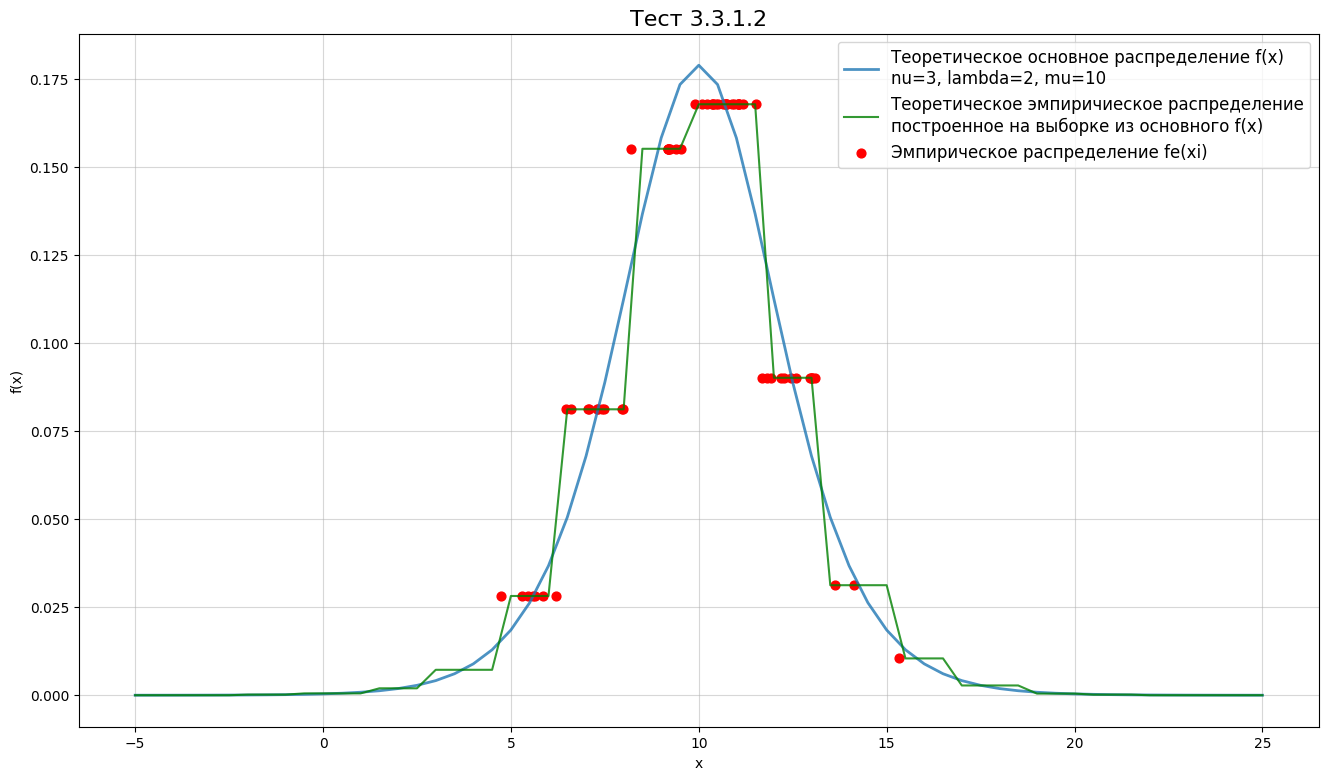
3.2.2

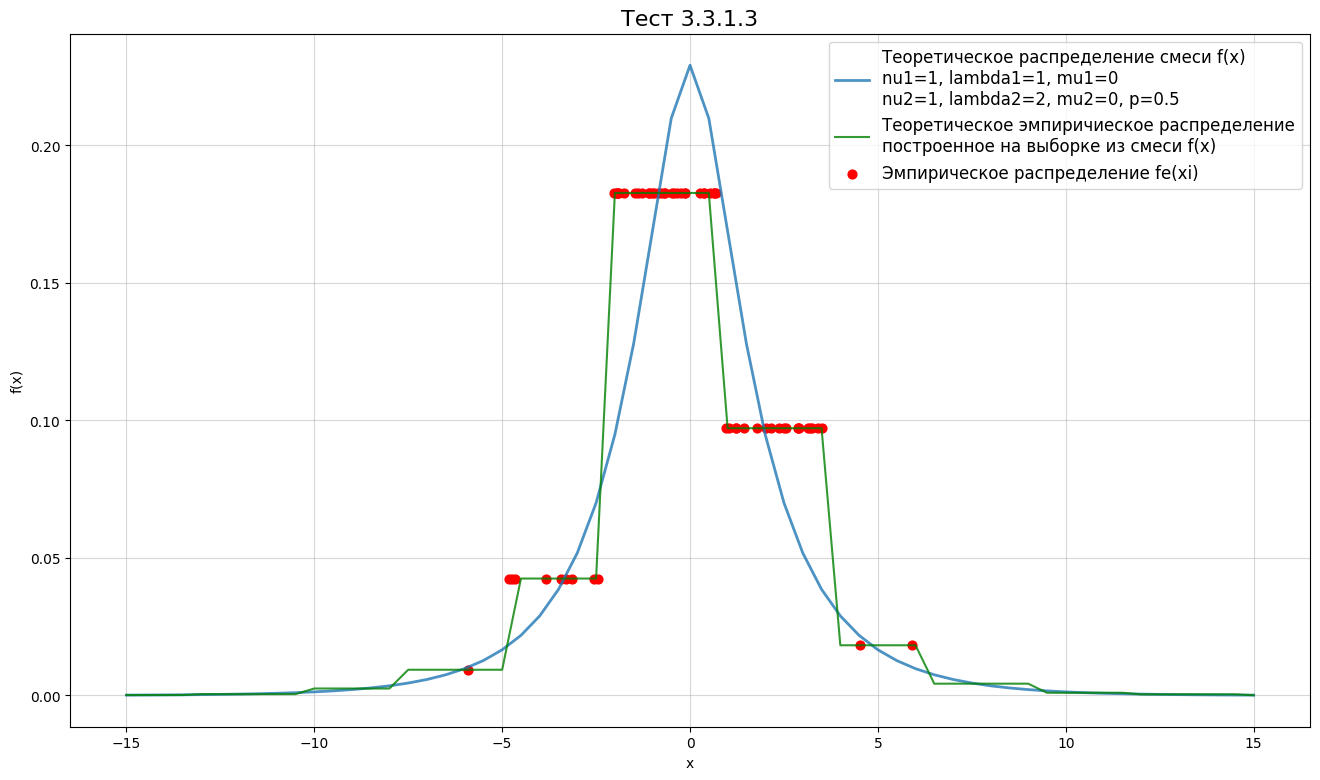
3.2.3

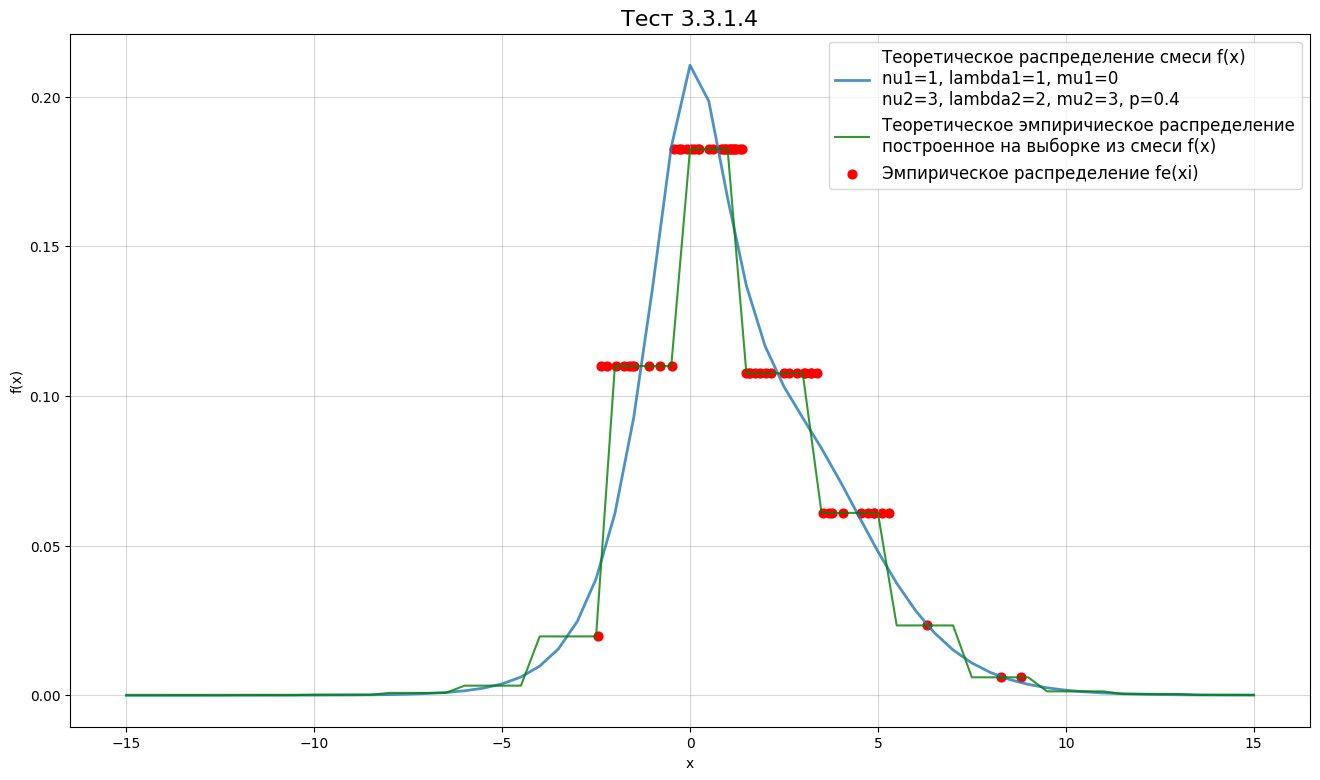
3.2.4

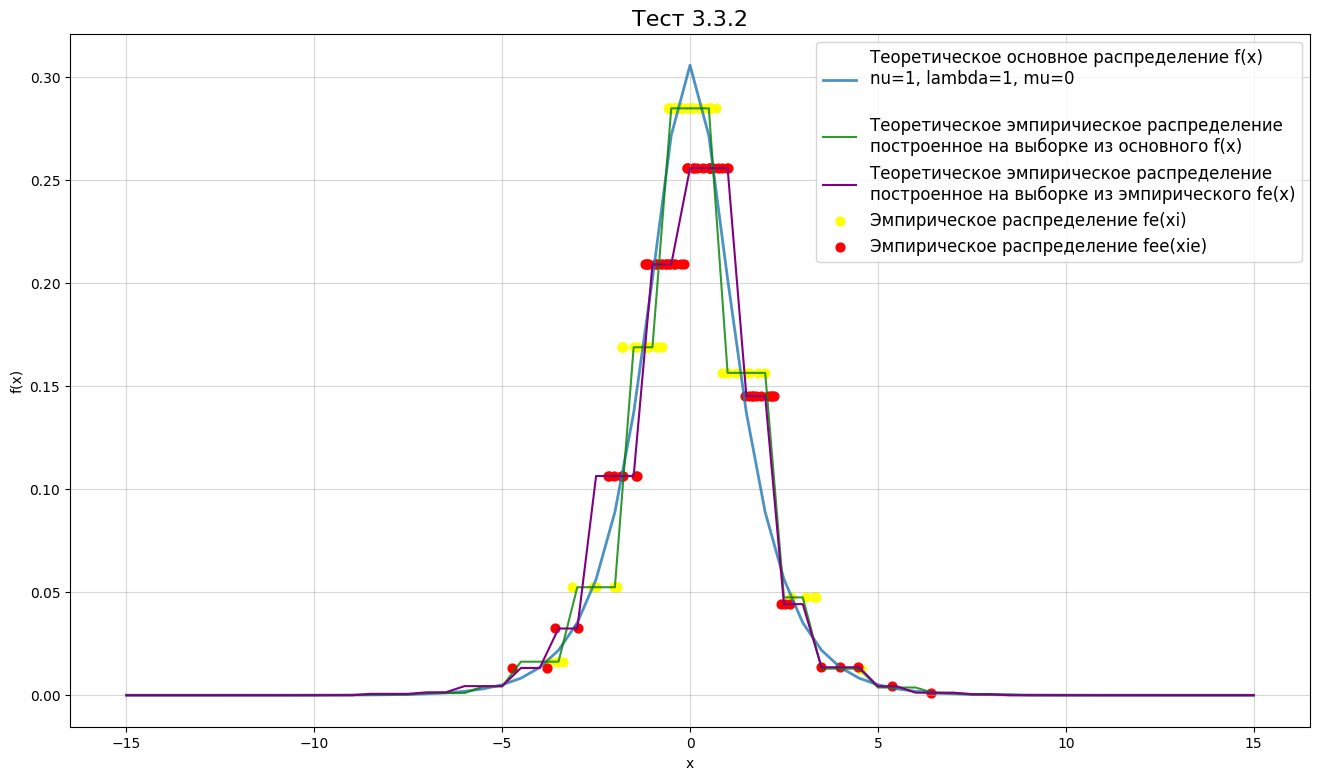


3.3.1.1

3.3.1.2

3.3.1.3

3.3.1.4

3.3.2

**Вывод**

В ходе лабораторной работы мы изучили реализацию отношений между классами и понятие интерфейса на языке С++.

**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

**Интерфейс распределений (IDist.hpp):**

#pragma once

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

class IDist {

 public:

  /\*\*

   \* Создание копии объекта и возвращение указателя на него

   \*/

  // virtual std::unique\_ptr<IDist> copy() const noexcept = 0;

  /\*\*

   \* Вычисление плотности вероятности в точке x

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

   \* @return Значение плотности вероятности в точке x

   \*/

  virtual ld density(ld x) const noexcept = 0;

  /\*\*

   \* 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

   \*/

  virtual nstu::vector density(const nstu::vector& x) const = 0;

  /\*\*

   \* Вычисление математического ожидания

   \* @return Значение математического ожидания

   \*/

  virtual ld M() const noexcept = 0;

  /\*\*

   \* Вычисление дисперсии

   \* @return Значение дисперсии

   \*/

  virtual ld D() const noexcept = 0;

  /\*\*

   \* Вычисление коэффициента асимметрии

   \* @return Значение коэффициента асимметрии

   \*/

  virtual ld G1() const noexcept = 0;

  /\*\*

   \* Вычисление коэффициента эксцесса

   \* @return Значение коэффициента эксцесса

   \*/

  virtual ld G2() const noexcept = 0;

  /\*\*

   \* Моделирование случайной величины

   \* @return Случайная величина

   \*/

  virtual ld Xi() const noexcept = 0;

  /\*\*

   \* Моделирование выборки случайных величин

   \* @param size Размер выборки случайных величин

   \* @return Выборка случайных величин

   \*/

  virtual nstu::vector Xi(uint32\_t size) const = 0;

};

**Интерфейс персистентности (IPersistent.hpp):**

#pragma once

#include <fstream>

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

class IPersistent {

 public:

  /\*\*

   \* Сохранение в файл

   \* @param file\_name Имя файла

   \*/

  virtual void save(const std::string& file\_name) const = 0;

  /\*\*

   \* Чтение из файла

   \* @param file\_name Имя файла

   \*/

  virtual void load(const std::string& file\_name) = 0;

  /\*\*

   \* Сохранение в файл

   \* @param file Поток открытого файла

   \*/

  virtual void save(std::ofstream& file) const noexcept = 0;

  /\*\*

   \* Чтение из файла

   \* @param file Поток открытого файла

   \*/

  virtual void load(std::ifstream& file) = 0;

};

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

#pragma once

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

#include "IDist.hpp"

#include "IPersistent.hpp"

class MainDist : public IDist, private IPersistent {

 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 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;

  // Персистентность

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

  void save(std::ofstream& file) const noexcept override;

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

  void load(std::ifstream& file) override;

  // 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

  ld density(ld x) const noexcept override;

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

  ld M() const noexcept override;

  ld D() const noexcept override;

  ld G1() const noexcept override;

  ld G2() const noexcept override;

  ld Xi() const noexcept override;

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

};

**Реализация класса MainDist (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");

}

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

  std::ifstream file(file\_name);

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

  load(file);

  file.close();

}

void MainDist::load(std::ifstream& file) {

  std::string d\_name;

  file >> d\_name;

  if (d\_name != "MainDist")

    throw std::runtime\_error("Неверный тип распределения");

  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");

}

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");

  save(file);

  file.close();

}

void MainDist::save(std::ofstream& file) const noexcept {

  file << "MainDist ";

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

}

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"

#include "IDist.hpp"

#include "IPersistent.hpp"

template <typename dist1\_t, typename dist2\_t>

class MixtureDist : public IDist, public IPersistent {

  static\_assert(std::is\_base\_of<IDist, dist1\_t>::value,

                "dist1\_t должен наследоваться от IDist");

  static\_assert(std::is\_base\_of<IDist, dist2\_t>::value,

                "dist2\_t должен наследоваться от IDist");

 private:

  ld m\_p;           // Параметр смеси

  dist1\_t m\_dist1;  // Первый компонент

  dist2\_t 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 dist1\_t& dist1, const dist2\_t& dist2);

  // 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;

  // Персистентность

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

  void save(std::ofstream& file) const noexcept override;

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

  void load(std::ifstream& file) override;

  // Геттеры

  // Получить первый компонент

  dist1\_t& component1() noexcept;

  const dist1\_t& component1() const noexcept;

  // Получить второй компонент

  dist2\_t& component2() noexcept;

  const dist2\_t& 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

  ld density(ld x) const noexcept override;

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

  ld M() const noexcept override;

  ld D() const noexcept override;

  ld G1() const noexcept override;

  ld G2() const noexcept override;

  ld Xi() const noexcept override;

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

};

#include "MixtureDist.tpp"

**Реализация шаблонного класса MixtureDist (MixtureDist.tpp):**

#include <cmath>

#include <fstream>

#include <memory>

#include "MixtureDist.hpp"

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>::MixtureDist() : m\_p(0.5), m\_dist1(), m\_dist2() {}

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>::MixtureDist(ld p, const dist1\_t& dist1,

                                           const dist2\_t& 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]");

}

template <typename dist1\_t, typename dist2\_t>

void MixtureDist<dist1\_t, dist2\_t>::save(const std::string& file\_name) const {

  std::ofstream file(file\_name);

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

  save(file);

  file.close();

}

template <typename dist1\_t, typename dist2\_t>

void MixtureDist<dist1\_t, dist2\_t>::save(std::ofstream& file) const noexcept {

  file << "MixtureDist ";

  file << m\_p << " ";

  m\_dist1.save(file);

  m\_dist2.save(file);

}

template <typename dist1\_t, typename dist2\_t>

void MixtureDist<dist1\_t, dist2\_t>::load(const std::string& file\_name) {

  std::ifstream file(file\_name);

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

  load(file);

  file.close();

}

template <typename dist1\_t, typename dist2\_t>

void MixtureDist<dist1\_t, dist2\_t>::load(std::ifstream& file) {

  std::string d\_name;

  file >> d\_name;

  if (d\_name == "MixtureDist") {

    file >> m\_p;

    if (m\_p < 0 || m\_p > 1)

      throw std::invalid\_argument(

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

    m\_dist1.load(file);

    m\_dist2.load(file);

  } else

    throw std::runtime\_error("Неверный тип распределения");

}

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>::MixtureDist(const MixtureDist& other)

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

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>& MixtureDist<dist1\_t, dist2\_t>::operator=(

    const MixtureDist& other) {

  if (this != &other) {

    m\_p = other.m\_p;

    m\_dist1 = other.m\_dist1;

    m\_dist2 = other.m\_dist2;

  }

  return \*this;

}

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>::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;

}

template <typename dist1\_t, typename dist2\_t>

MixtureDist<dist1\_t, dist2\_t>& MixtureDist<dist1\_t, dist2\_t>::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;

}

template <typename dist1\_t, typename dist2\_t>

dist1\_t& MixtureDist<dist1\_t, dist2\_t>::component1() noexcept {

  return m\_dist1;

}

template <typename dist1\_t, typename dist2\_t>

const dist1\_t& MixtureDist<dist1\_t, dist2\_t>::component1() const noexcept {

  return m\_dist1;

}

template <typename dist1\_t, typename dist2\_t>

dist2\_t& MixtureDist<dist1\_t, dist2\_t>::component2() noexcept {

  return m\_dist2;

}

template <typename dist1\_t, typename dist2\_t>

const dist2\_t& MixtureDist<dist1\_t, dist2\_t>::component2() const noexcept {

  return m\_dist2;

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::getP() const noexcept {

  return m\_p;

}

template <typename dist1\_t, typename dist2\_t>

void MixtureDist<dist1\_t, dist2\_t>::setP(ld p) {

  if (p < 0 || p > 1)

    throw std::invalid\_argument(

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

  else

    m\_p = p;

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::density(ld x) const noexcept {

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

}

template <typename dist1\_t, typename dist2\_t>

nstu::vector MixtureDist<dist1\_t, dist2\_t>::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;

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::M() const noexcept {

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

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::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);

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::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);

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::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;

}

template <typename dist1\_t, typename dist2\_t>

ld MixtureDist<dist1\_t, dist2\_t>::Xi() const noexcept {

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

}

template <typename dist1\_t, typename dist2\_t>

nstu::vector MixtureDist<dist1\_t, dist2\_t>::Xi(uint32\_t size) const {

  nstu::vector result(size);

  for (ld& i : result) i = Xi();

  return result;

}

**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 "IDist.hpp"

#include "IPersistent.hpp"

class EmpiricDist : public IDist, public IPersistent {

 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 распределения.

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

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

   \* Стёрджеса.

   \*/

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

  // Персистентность

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

  void save(std::ofstream& file) const noexcept override;

  void load(std::ifstream& file) override;

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

  // 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

  ld density(ld x) const noexcept override;

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

  ld M() const noexcept override;

  ld D() const noexcept override;

  ld G1() const noexcept override;

  ld G2() const noexcept override;

  ld Xi() const noexcept override;

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

};

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

#include "EmpiricDist.hpp"

#include <cmath>

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();

  static int n = 0;

  if (n == 0) {

    std::ofstream file("logini.txt");

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

    file << m\_k << "\n";

    file << m\_min << "\n";

    file << m\_max << "\n";

    file << m\_delta << "\n";

    for (const ld i : m\_interval\_freq) file << i << " ";

    n++;

  }

  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, IDist& dist, uint32\_t k)

    : EmpiricDist(dist.Xi(n), k) {}

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

  std::ifstream file(file\_name);

  if (!file.is\_open())

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

  load(file);

  file.close();

}

void EmpiricDist::load(std::ifstream& file) {

  std::string name;

  file >> name;

  if (name != "EmpiricDist")

    throw std::runtime\_error("Неверный тип распределения");

  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;

  }

  \*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);

  save(file);

  file.close();

}

void EmpiricDist::save(std::ofstream& file) const noexcept {

  file << "EmpiricDist ";

  file << m\_k << " ";

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

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

    file << i << " ";

  }

}

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);

  static int n = 0;

  if (n == 0) {

    std::ofstream file("log" + std::to\_string(n++) + ".txt");

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

      file << m\_interval\_freq[i] << "\n";

    }

  }

  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;

}