**Цель работы**

Изучить средства работы с классами на языке С++.

**Постановка задачи**

1. Ознакомиться со следующими вопросами и понятиями: определение пользовательского типа данных с помощью конструкции class, открытая и закрытая части описания класса, атрибуты и функции класса, интерфейс, описание и реализация класса, модульность в языке С++, персистентность, генерация и обработка исключительных ситуаций.
2. Модифицировать программу, разработанную в лабораторной работе №1, так чтобы в ней был определен класс, реализующий основное распределение

Основное распределение: **симметричное гиперболическое распределение**

**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) |

*Тестирование (подпункты пункта 3 содержания работы)*

|  |  |  |  |
| --- | --- | --- | --- |
| Распределение | Основные функции | | |
| Плотность | Характеристики *M*ξ, *D*ξ, γ1, γ2 | Модели-рование |
| Основное | 3.1 | 3.1 | 3.3.1 |

**Описание разработанного класса**

Класс описывающий объект основного распределения MainDist имеет приватные (закрытые) атрибуты: m\_nu; m\_mu; m\_lambda, доступ к которым осуществляется через get/set методы, создание происходит через конструкторы: стандартный (с 3 параметрами); чтение из файла; копирования; перемещения. Конструкторы и некоторые сеттеры имеют обработку ошибок (нулевых значений nu или lambda). Методы класса предоставляют такие же функции, как и в первой лабораторной работе: плотность; мат. ожидание; дисперсия; коэффициент асимметрии; коэффициент эксцесса; моделирование случайной величины. Также для удобства вместо использования массивов был создан вспомогательный класс vector, пародирующий vector из STL.

**Алгоритм**

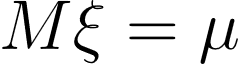
Расчет **основного** распределения:

Расчет плотности симметрического гиперболического распределения осуществляется по формуле:

 (1.1)

С учетом сдвига и масштаба:

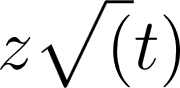
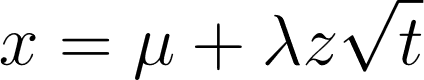
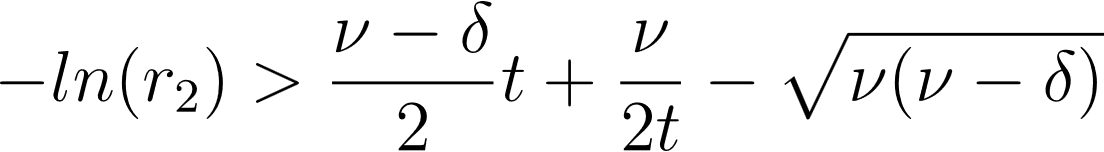
(1.2)

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)

**Тестирование**

3.1. Минимальный набор тестов для основного распределения (из предыдущей лабораторной работы):

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.

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:**

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

Тестирование значений из таблицы в варианте:

D = 20.2463 G2 = 2.93433 f(0) = 0.145189

D = 4.55808 G2 = 2.40978 f(0) = 0.258918

D = 2.69948 G2 = 1.85697 f(0) = 0.305595

D = 1.81431 G2 = 1.21843 f(0) = 0.342101

D = 1.53177 G2 = 0.889953 f(0) = 0.357908

D = 1.3126 G2 = 0.569679 f(0) = 0.372508

D = 1.15342 G2 = 0.295396 f(0) = 0.384924

D = 1.0504 G2 = 0.0998051 f(0) = 0.394066

-----Тест 3.1.1-----

Тест основного распределения:

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

Плотность вероятности f(x): s0.305595, c0.305595

M = s0, c0; D = s2.69948, c2.69948; G1 = s0, c0; G2 = s1.85697, c1.85697

-----Тест 3.1.2-----

Тест основного распределения:

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

Плотность вероятности f(x): s0.152797, c0.152797

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

-----Тест 3.1.3-----

Тест основного распределения:

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

--- 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 lambda=0 в файле: Лямбда не может быть 0

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

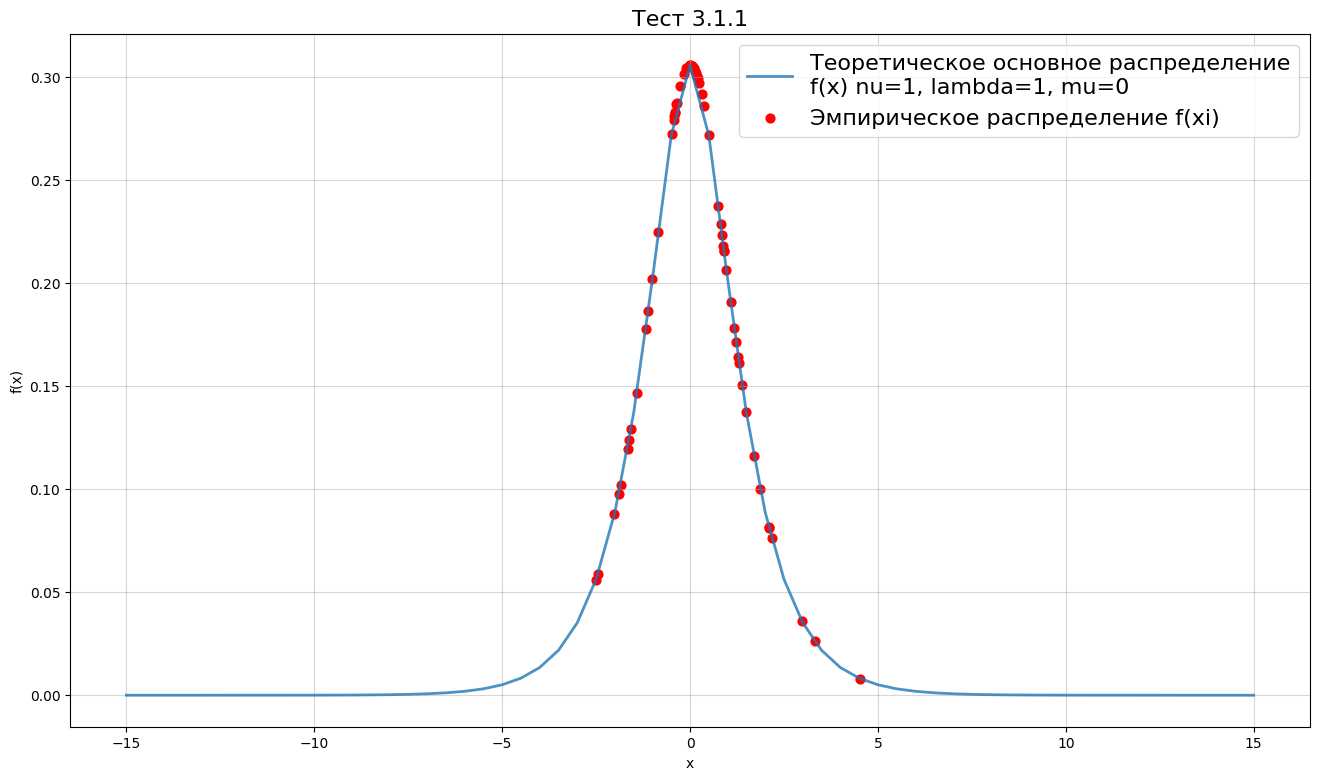
- Успех (setNu(0)): Нулевое значение nu

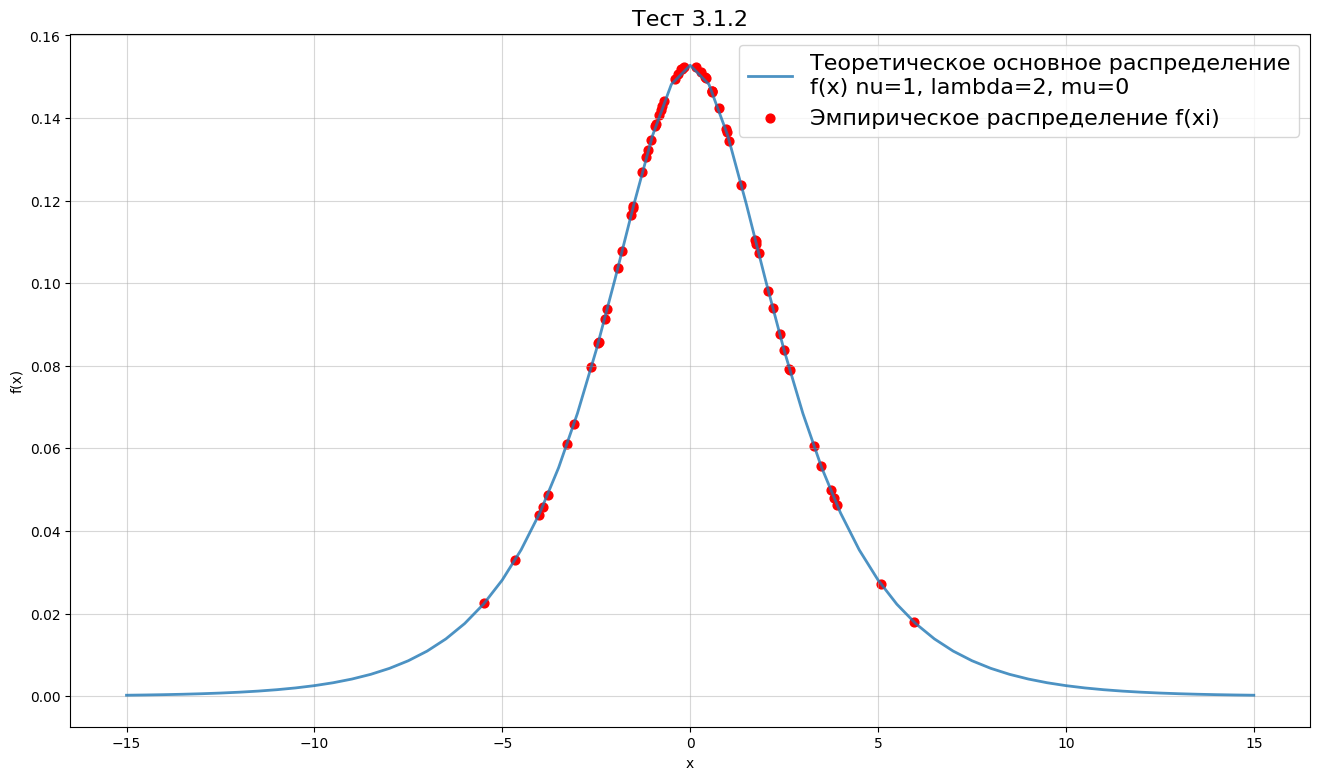
--- ПОЛНОЕ ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ---

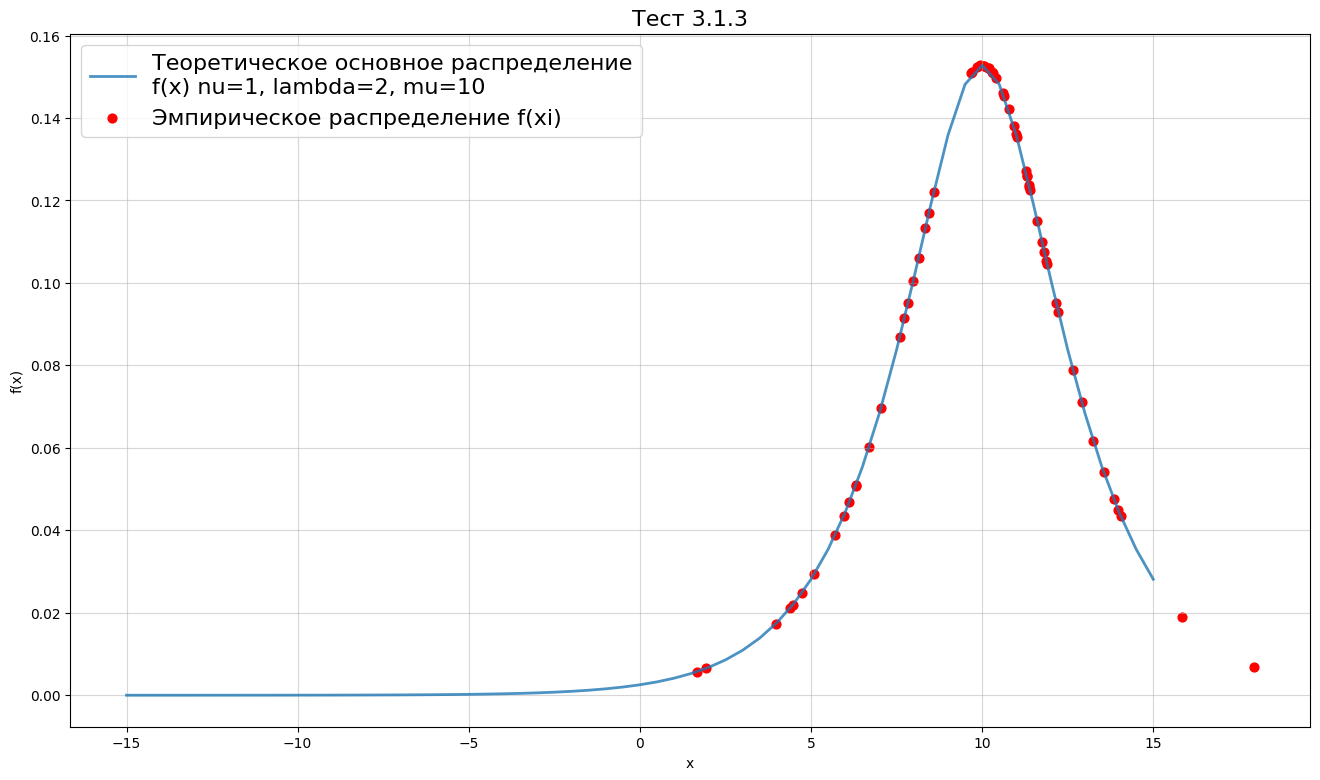
**Графики распределений**

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

Графики тестов:

3.1.1

3.1.2

3.1.3

**Вывод**

В ходе лабораторной работы мы изучили средства работы с классами на языке С++ и модифицировали программу, разработанную в лабораторной работе №1

**Приложение**

Код программы

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

#pragma once

#include <memory>

#include <stdexcept>

using ld = long double;

namespace nstu {

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

  const\_iterator cend() const noexcept {

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

  }

  ~vector() {}

};

}  // namespace nstu

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

#pragma once

#include <string>

#include "header.h"

class MainDist {

 private:

  const ld pi = 3.14159265358979;

  ld m\_nu;

  ld m\_mu;

  ld m\_lambda;

 public:

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

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

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

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

  MainDist(const MainDist& other);

  MainDist& operator=(const MainDist& other);

  MainDist(MainDist&& other) noexcept;

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

  ~MainDist() = default;

  ld getNu() const noexcept;

  ld getMu() const noexcept;

  ld getLambda() const noexcept;

  void setNu(ld nu);

  void setMu(ld mu) noexcept;

  void setLambda(ld lambda);

  ld density(ld x) const noexcept;

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

  ld M() const noexcept;

  ld D() const noexcept;

  ld G1() const noexcept;

  ld G2() const noexcept;

  ld Xi() const noexcept;

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

};

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

#include "MainDist.hpp"

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

#include <cmath>

#include <fstream>

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::invalid\_argument("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 \* 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;

}

**Код тестирования (test.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 "MainDist.hpp"

// Вспомогательная функция для печати состояния объекта MainDist

void printDistState(const MainDist& dist, const std::string& name) {

  std::cout << "   - Состояние " << name

            << " (nu, mu, lambda): " << dist.getNu() << ", " << dist.getMu()

            << ", " << dist.getLambda() << std::endl;

}

// 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) {

    // Сравниваем числа с плавающей точкой с небольшой погрешностью

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

      return false;

    }

  }

  return true;

}

void testAllMainDistMethods() {

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

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

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

  // 1. ТЕСТИРОВАНИЕ КОНСТРУКТОРОВ И ГЕТТЕРОВ

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

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

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

  // 1.1. Конструктор с параметрами

  std::cout << "1.1. Конструктор с параметрами: MainDist(5.0, 1.0, 2.0)\n";

  MainDist dist\_params(5.0, 1.0, 2.0);

  printDistState(dist\_params, "dist\_params");

  // 1.2. Конструктор из файла

  std::cout << "\n1.2. Конструктор из файла (test\_params.txt)\n";

  MainDist dist\_file\_params(5.0, 1.0, 2.0);

  dist\_file\_params.save("test\_params.txt");

  MainDist dist\_file("test\_params.txt");

  printDistState(dist\_file\_params, "dist\_file\_params");

  printDistState(dist\_file, "dist\_file");

  // 1.3. Конструктор с initializer\_list

  std::cout << "\n1.3. Конструктор с initializer\_list: {5.0, 1.0, 2.0}\n";

  MainDist dist\_list{5.0, 1.0, 2.0};

  printDistState(dist\_list, "dist\_list");

  // 1.4. Конструктор копирования

  std::cout

      << "\n1.4. Конструктор копирования: MainDist dist\_copy = dist\_params\n";

  MainDist dist\_copy = dist\_params;

  printDistState(dist\_copy, "dist\_copy");

  // Убедимся, что это глубокая копия (хотя для ld это очевидно)

  dist\_params.setNu(99.0);

  std::cout << "   Изменили nu у оригинала до 99.0\n";

  printDistState(dist\_params, "dist\_params (измененный)");

  printDistState(dist\_copy, "dist\_copy (неизмененный)");

  dist\_params.setNu(5.0);  // Вернем обратно

  // 1.5. Конструктор перемещения

  std::cout

      << "\n1.5. Конструктор перемещения: MainDist dist\_move = MainDist(...)\n";

  MainDist dist\_move = MainDist(10.0, -5.0, 1.0);

  printDistState(dist\_move, "dist\_move");

  // Состояние перемещенного объекта не определено стандартом, но в вашей

  // реализации оно обнуляется

  std::cout << "\n";

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

  // 2. ТЕСТИРОВАНИЕ ОПЕРАТОРОВ ПРИСВАИВАНИЯ

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

  std::cout << "2. Тестирование операторов присваивания\n";

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

  // 2.1. Оператор присваивания копированием

  std::cout << "2.1. Присваивание копированием\n";

  MainDist assign\_target(1, 1, 1);

  MainDist assign\_source(8, 8, 8);

  printDistState(assign\_target, "assign\_target (до)");

  printDistState(assign\_source, "assign\_source");

  assign\_target = assign\_source;

  printDistState(assign\_target, "assign\_target (после)");

  // 2.2. Оператор присваивания перемещением

  std::cout << "\n2.2. Присваивание перемещением\n";

  MainDist move\_target(1, 1, 1);

  MainDist move\_source(9, 9, 9);

  printDistState(move\_target, "move\_target (до)");

  printDistState(move\_source, "move\_source");

  move\_target = std::move(move\_source);

  printDistState(move\_target, "move\_target (после)");

  printDistState(move\_source, "move\_source (после перемещения, обнулен)");

  std::cout << "\n";

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

  // 3. ТЕСТИРОВАНИЕ СЕТТЕРОВ

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

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

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

  MainDist dist\_set(1, 1, 1);

  printDistState(dist\_set, "dist\_set (начальное)");

  dist\_set.setNu(10.0);

  dist\_set.setMu(20.0);

  dist\_set.setLambda(30.0);

  printDistState(dist\_set, "dist\_set (конечное)");

  std::cout << "\n";

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

  // 4. ТЕСТИРОВАНИЕ АНАЛИТИЧЕСКИХ МЕТОДОВ

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

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

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

  MainDist dist\_analysis(5.0, 1.0, 2.0);

  printDistState(dist\_analysis, "dist\_analysis");

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

            << std::endl;

  std::cout << "   - Дисперсия (D): " << dist\_analysis.D() << std::endl;

  std::cout << "   - Асимметрия (G1): " << dist\_analysis.G1() << std::endl;

  std::cout << "   - Эксцесс (G2): " << dist\_analysis.G2() << std::endl;

  ld x\_point = 1.5;

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

            << dist\_analysis.density(x\_point) << std::endl;

  nstu::vector x\_vec = {0.5, 1.0, 1.5, 2.0};

  nstu::vector densities = dist\_analysis.density(x\_vec);

  std::cout << "   - Плотности для вектора {0.5, 1.0, 1.5, 2.0}:\n     { ";

  for (const auto& d : densities) {

    std::cout << d << " ";

  }

  std::cout << "}\n\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;

  srand(seed);

  ld single\_xi = dist\_analysis.Xi();

  std::cout << "   - Одиночное случайное число Xi(): " << single\_xi << "\n";

  srand(seed);  // Сброс для воспроизводимости

  nstu::vector sample1 = dist\_analysis.Xi(sample\_size);

  std::cout << "   - Вектор Xi(" << sample\_size << ") [Запуск 1]: { ";

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

  std::cout << "}\n";

  srand(seed);  // Снова сброс

  nstu::vector sample2 = dist\_analysis.Xi(sample\_size);

  std::cout << "   - Вектор Xi(" << sample\_size << ") [Запуск 2]: { ";

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

  std::cout << "}\n";

  if (areVectorsEqual(sample1, sample2)) {

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

  // 6.1. lambda = 0 в конструкторе

  try {

    MainDist(1, 1, 0);

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

    std::cout << "   - Успех: перехвачено исключение для lambda=0: " << e.what()

              << std::endl;

  }

  // 6.2. Неверное число аргументов в initializer\_list

  try {

    MainDist{1.0, 2.0};

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

    std::cout

        << "   - Успех: перехвачено исключение для неверного числа аргументов: "

        << e.what() << std::endl;

  }

  // 6.3. Не удалось открыть файл

  try {

    MainDist("non\_existent\_file.txt");

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

    std::cout

        << "   - Успех: перехвачено исключение для несуществующего файла: "

        << e.what() << std::endl;

  }

  // 6.4. lambda = 0 в файле

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

  bad\_file << "5.0 1.0 0.0";

  bad\_file.close();

  try {

    MainDist("bad\_params.txt");

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

    std::cout << "   - Успех: перехвачено исключение для lambda=0 в файле: "

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

  }

  // 6.5. Установка lambda = 0 через сеттер

  MainDist dist\_exc(1, 1, 1);

  try {

    dist\_exc.setLambda(0);

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

    std::cout << "   - Успех (setLambda(0)): " << e.what() << std::endl;

  }

  // 6.6. Установка nu = 0 через сеттер

  try {

    dist\_exc.setNu(0);

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

    std::cout << "   - Успех (setNu(0)): " << e.what() << std::endl;

  }

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

}

int main() {

  try {

    testAllMainDistMethods();

  } catch (const std::exception& e) {

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

              << std::endl;

    return 1;

  }

  return 0;

}

**Код генерации тестовых данных для графиков (main.cpp):**

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

#include "MainDist.hpp"

#include "header.h"

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\_1\_1() {

  ld mu = 0, lambda = 1, nu = 1;

  ld x = 0;

  Distribution struct\_dist;

  initDistribution(struct\_dist, nu, mu, lambda);

  MainDist main\_dist(nu, mu, lambda);

  cout << "-----Тест 3.1.1-----" << endl;

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

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

       << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityDist(struct\_dist, x)

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

  cout << "M = s" << MDist(struct\_dist) << ", c" << main\_dist.M() << "; D = s"

       << DDist(struct\_dist) << ", c" << main\_dist.D() << "; G1 = s" << G1Dist()

       << ", c" << main\_dist.G1() << "; G2 = s" << G2Dist(struct\_dist) << ", c"

       << main\_dist.G2() << endl;

  auto fout = ofstream("test311.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

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

    ld xi = main\_dist.Xi();

    ld density = main\_dist.density(i);

    ld density\_xi = main\_dist.density(xi);

    fout << i << "," << density << "," << xi << "," << density\_xi << endl;

  }

}

void testDistribution\_3\_1\_2() {

  ld mu = 0, lambda = 2, nu = 1;

  ld x = 0;

  Distribution struct\_dist;

  initDistribution(struct\_dist, nu, mu, lambda);

  MainDist main\_dist(nu, mu, lambda);

  cout << "\n-----Тест 3.1.2-----" << endl;

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

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

       << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityDist(struct\_dist, x)

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

  cout << "M = s" << MDist(struct\_dist) << ", c" << main\_dist.M() << "; D = s"

       << DDist(struct\_dist) << ", c" << main\_dist.D() << "; G1 = s" << G1Dist()

       << ", c" << main\_dist.G1() << "; G2 = s" << G2Dist(struct\_dist) << ", c"

       << main\_dist.G2() << endl;

  auto fout = ofstream("test312.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

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

    ld xi = main\_dist.Xi();

    ld density = main\_dist.density(i);

    ld density\_xi = main\_dist.density(xi);

    fout << i << "," << density << "," << xi << "," << density\_xi << endl;

  }

}

void testDistribution\_3\_1\_3() {

  ld mu = 10, lambda = 2, nu = 1;

  ld x = 0;

  Distribution struct\_dist;

  initDistribution(struct\_dist, nu, mu, lambda);

  MainDist main\_dist(nu, mu, lambda);

  cout << "\n-----Тест 3.1.3-----" << endl;

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

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

       << ", x = " << x << endl;

  cout << "Плотность вероятности f(x): s" << densityDist(struct\_dist, x)

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

  cout << "M = s" << MDist(struct\_dist) << ", c" << main\_dist.M() << "; D = s"

       << DDist(struct\_dist) << ", c" << main\_dist.D() << "; G1 = s" << G1Dist()

       << ", c" << main\_dist.G1() << "; G2 = s" << G2Dist(struct\_dist) << ", c"

       << main\_dist.G2() << endl;

  auto fout = ofstream("test313.csv");

  fout << "x,f(x),xi,f(xi)" << endl;

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

    ld xi = main\_dist.Xi();

    ld density = main\_dist.density(i);

    ld density\_xi = main\_dist.density(xi);

    fout << i << "," << density << "," << xi << "," << density\_xi << endl;

  }

}

void testVariantTable() {

  ld mu = 0, lambda = 1;

  ld nuarr[] = {0.1, 0.5, 1, 2, 3, 5, 10, 30};

  cout << "--------------------------------------------" << endl;

  cout << "Тестирование значений из таблицы в варианте:" << endl;

  for (ld nu : nuarr) {

    MainDist struct\_dist(nu, mu, lambda);

    cout << " D = " << struct\_dist.D() << " G2 = " << struct\_dist.G2()

         << " f(0) = " << struct\_dist.density(0) << endl;

  }

}

int main() {

  testVariantTable();

  testDistribution\_3\_1\_1();

  testDistribution\_3\_1\_2();

  testDistribution\_3\_1\_3();

  return 0;

}