**Цель работы**

Изучить сложную предметную область, связанную с вероятностно статистическим моделированием.

**Постановка задачи**

Методами структурного программирования реализовать возможность работы с тремя распределениями:

– распределением, заданным в варианте и имеющим, помимо параметра формы ν, параметры сдвига μ и масштаба λ (данное распределение будет называться основным);

– распределением в виде смеси двух основных распределений с параметрами (μ1, λ1, ν1) и (μ2, λ2, ν2) и параметром смеси p;

– эмпирическим распределением, строящимся по выборке.

Для каждого из распределений необходимо реализовать набор из следующих трех функций:

– функция для вычисления значений плотности распределения по заданному аргументу (плотность для эмпирического распределения необходимо предварительно сформировать по выборке);

– функция для вычисления математического ожидания, дисперсии, коэффициентов асимметрии и эксцесса;

– функция для моделирования случайной величины

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) |
| 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 |
| Основное | 3.1 | 3.1 | 3.3.1 |
| Смесь | 3.2 | 3.2 | 3.3.1 |
| Эмпирическое | 3.3.1 | 3.3.1, 3.3.2 | 3.3.2 |

**Алгоритм**

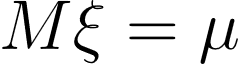
Расчет **основного** распределения:

Расчет плотности симметрического гиперболического распределения осуществляется по формуле:

 (1.1)

С учетом сдвига и масштаба:

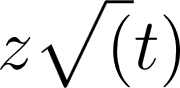
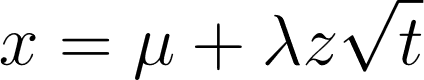
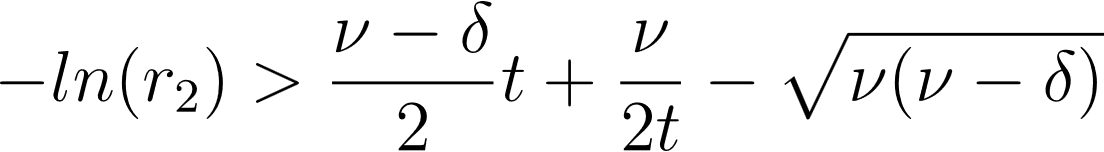
(1.2)

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)

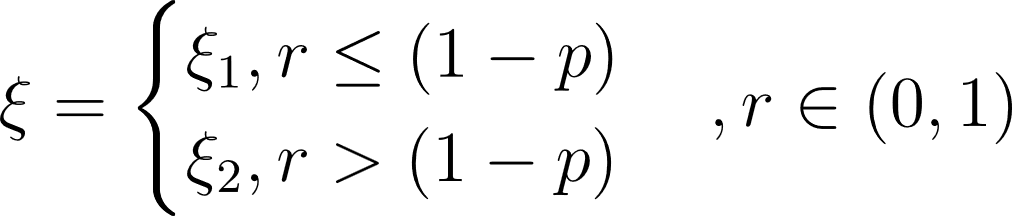
Коэффициент асимметрии:

 (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)

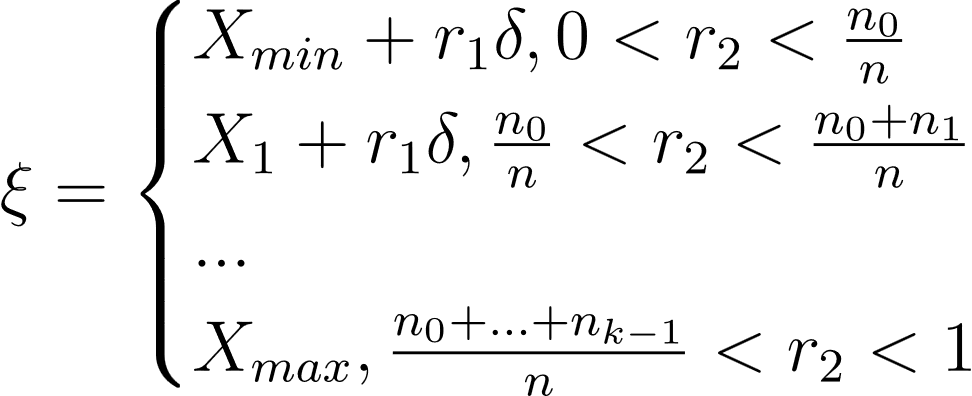
Коэффициент эксцесса:

 (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.

**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) = 0.259756, M = -0.0210726, D = 2.77066, G1 = -0.0801576, G2 = 1.67984

-----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) = 0.000782368, M = 9.99458, D = 6.10247, G1 = -0.0137251, G2 = 0.866874

-----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) = 0.197382, M = -0.00562339, D = 6.77149, G1 = -0.0337237, G2 = 3.6725

-----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) = 0.183013, M = 1.15778, D = 6.24873, G1 = 0.540441, G2 = 0.964601

-----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) = 0.279243, M = 0.00166345, D = 2.60982, G1 = 0.00160137, G2 = 1.82028

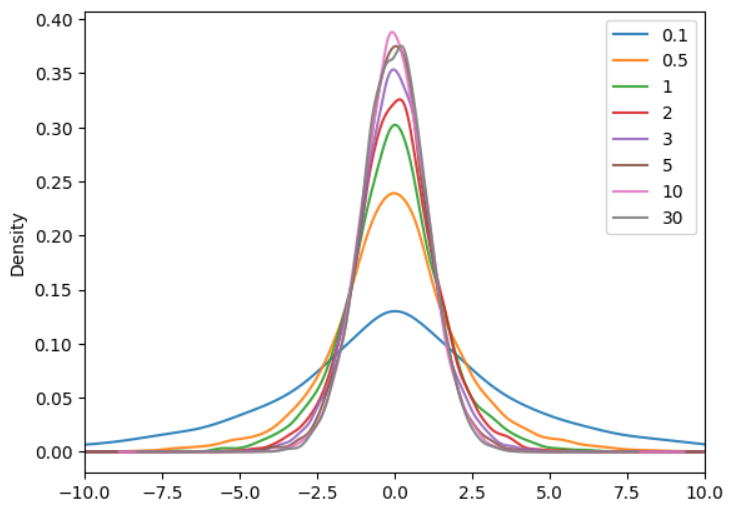
Эмпирическое распределение2:

f(0) = 0.284474, M = 0.0164176, D = 2.94009, G1 = 0.0527027, G2 = 1.24636

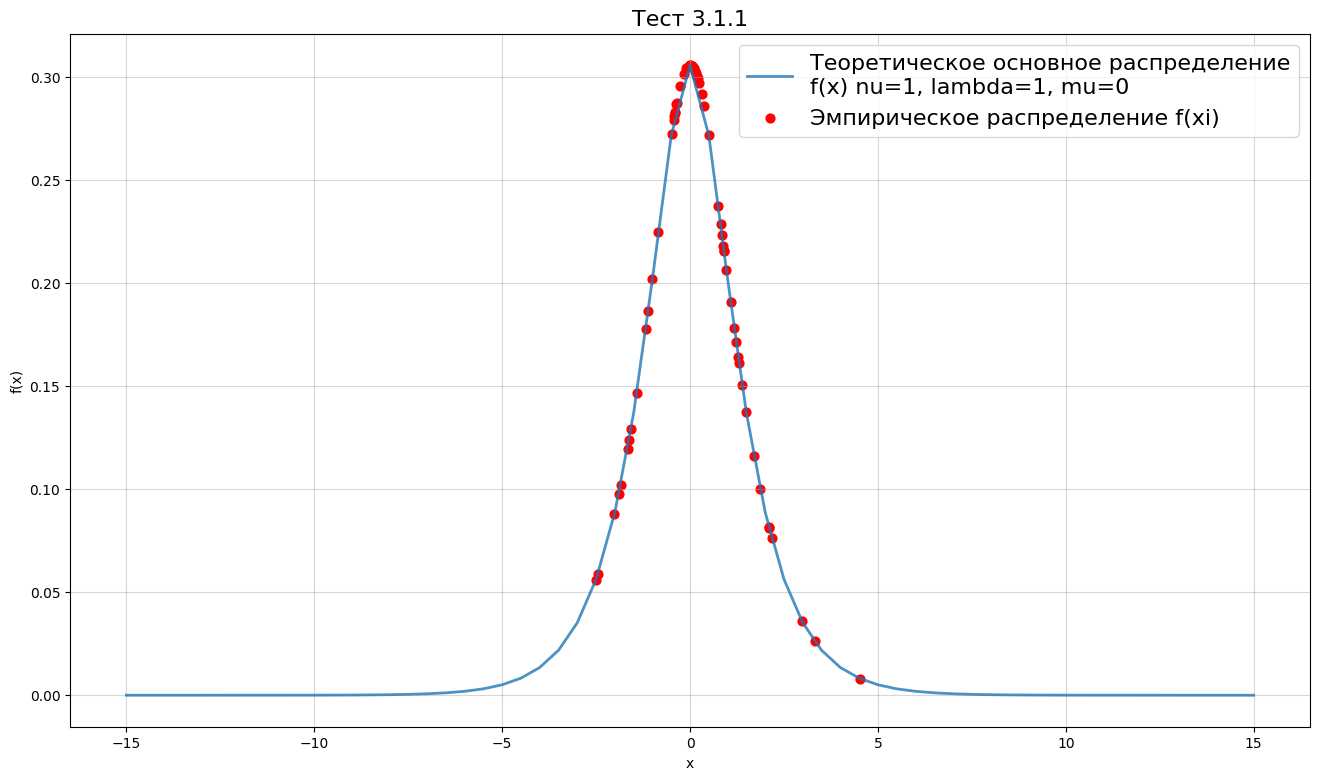
**Графики распределений**

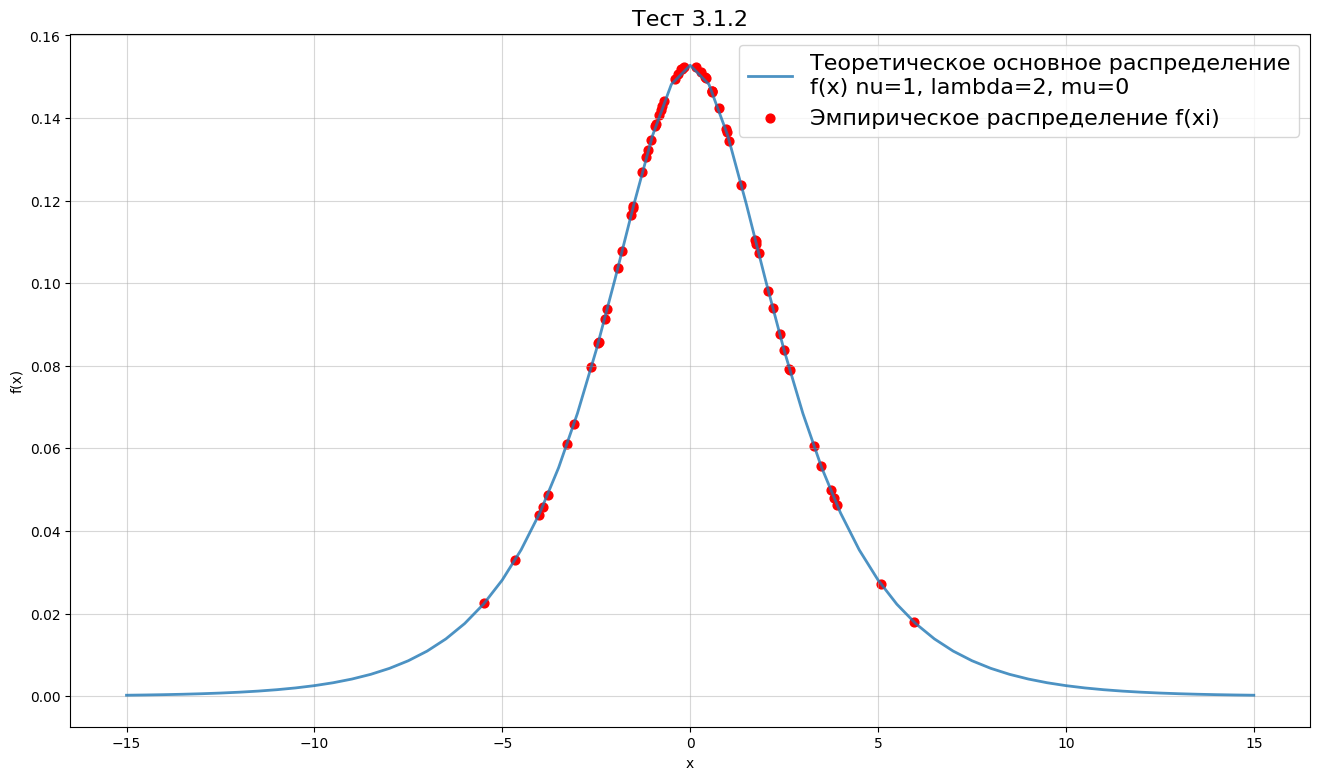
Все графики построены на выборке из 10000 элементов с помощью python.

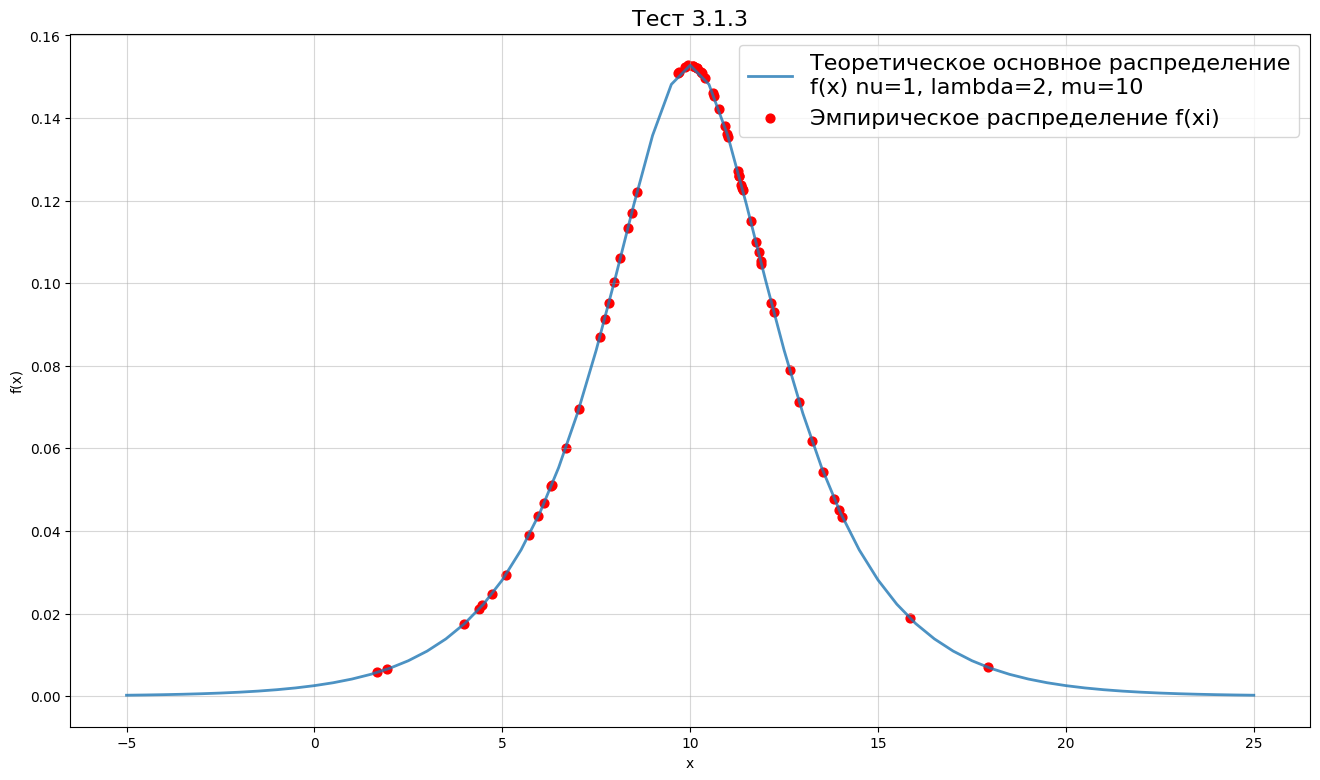
1. Основное распределение при табличных значениях ν, μ=0, λ=1

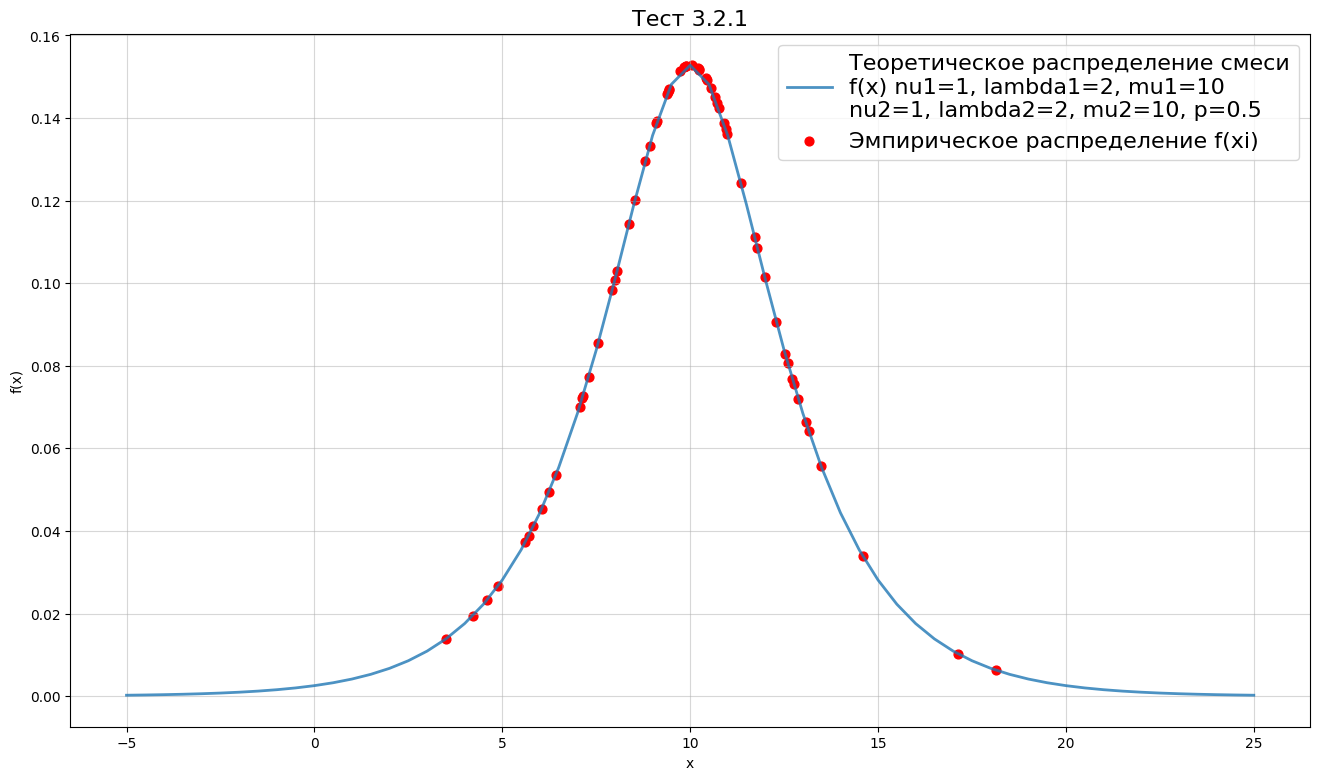


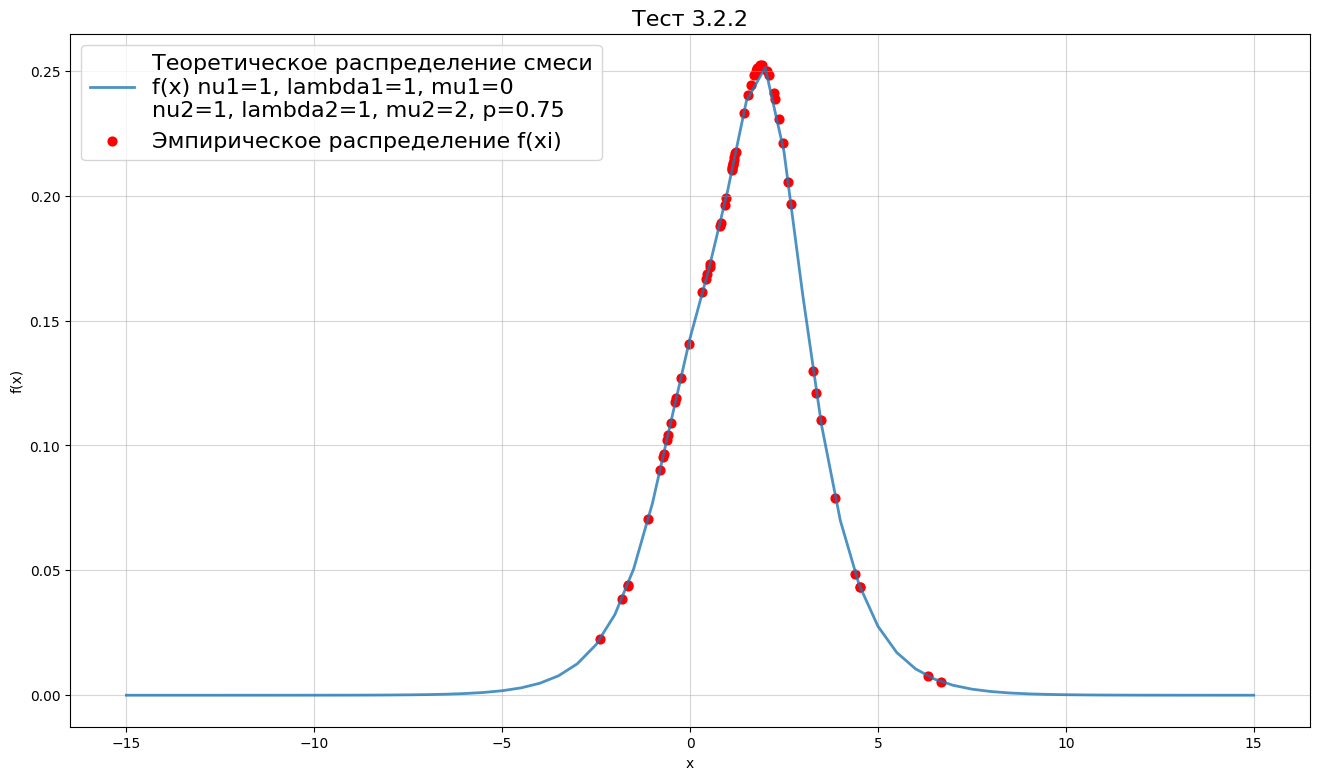
Графики тестов:

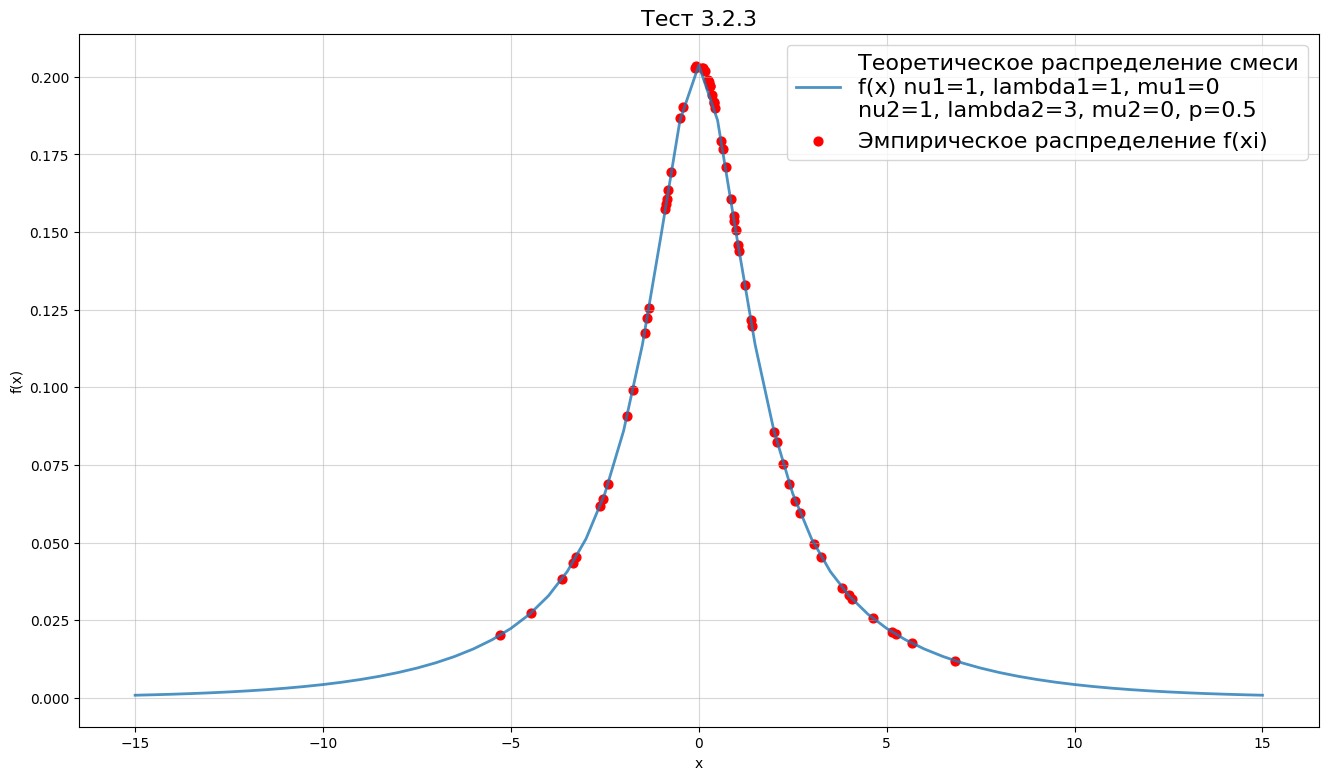
3.1.1

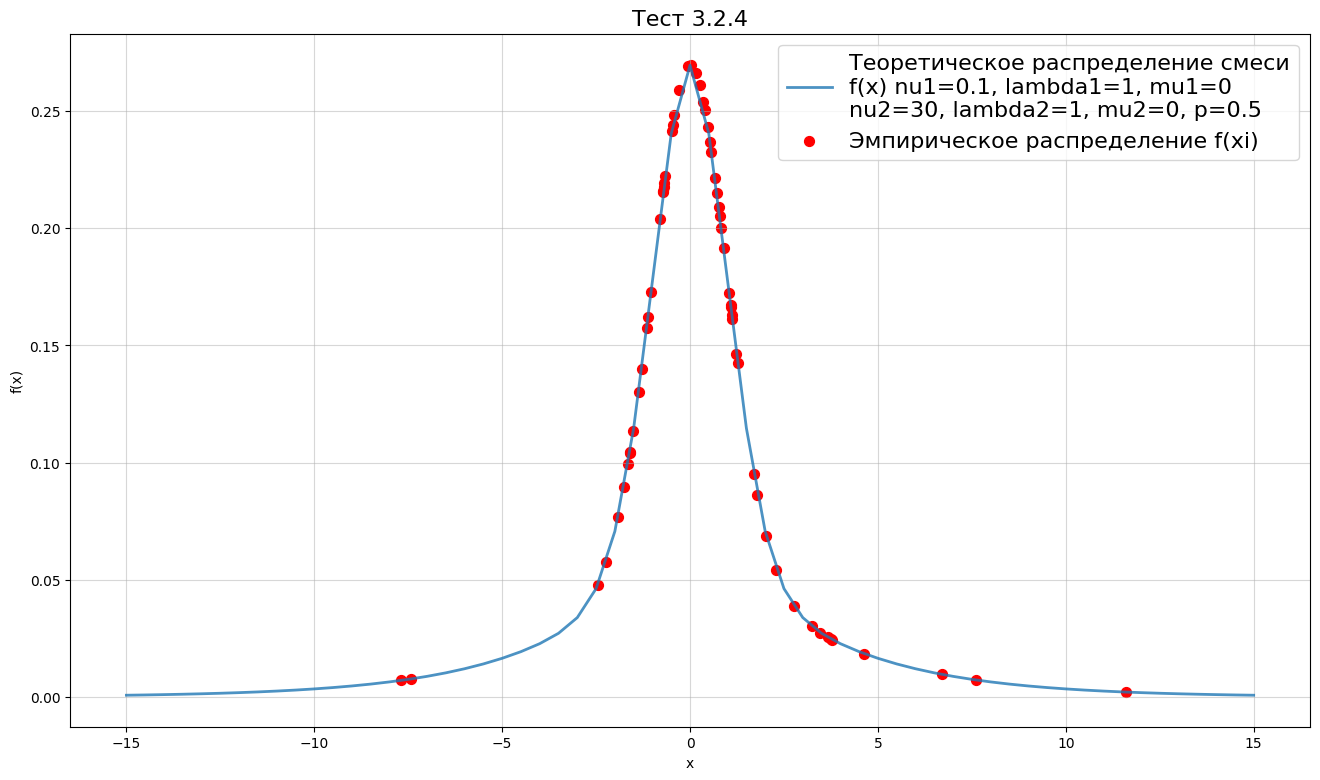
3.1.2

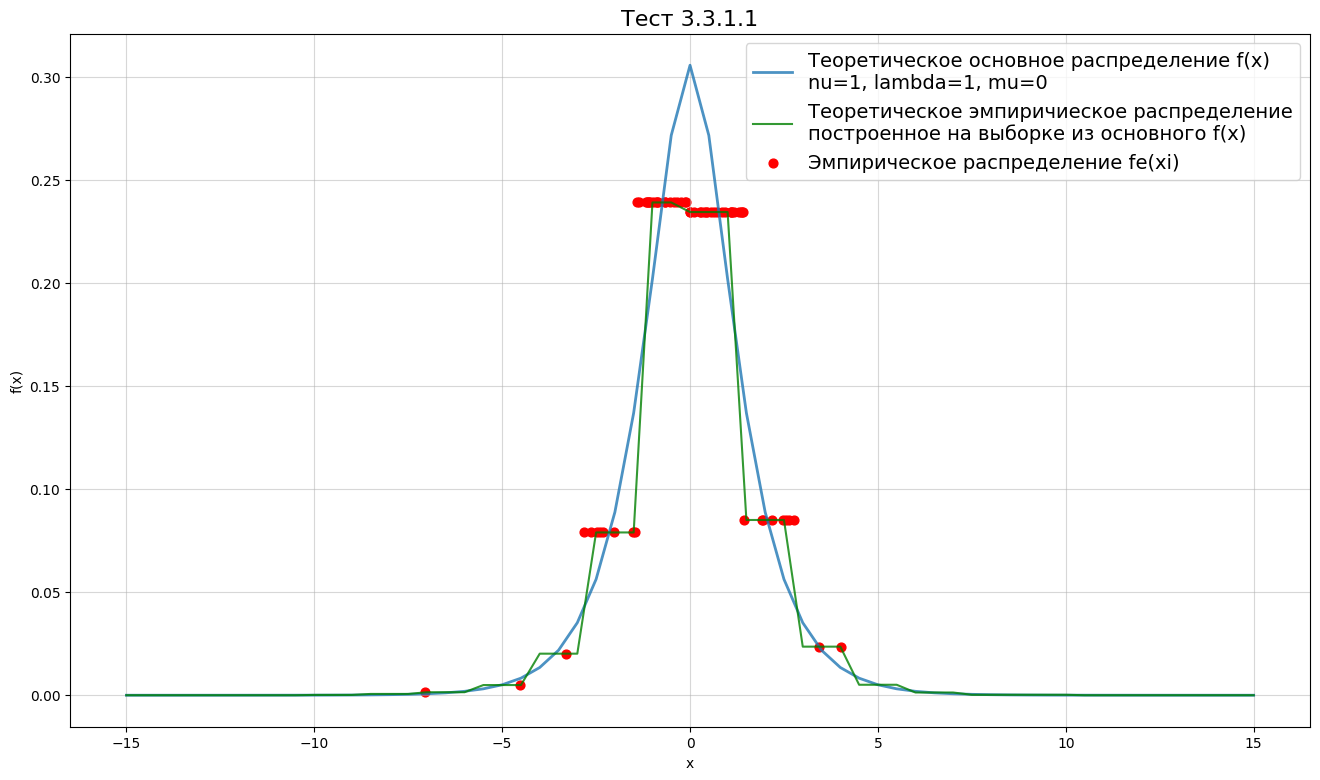
3.1.3

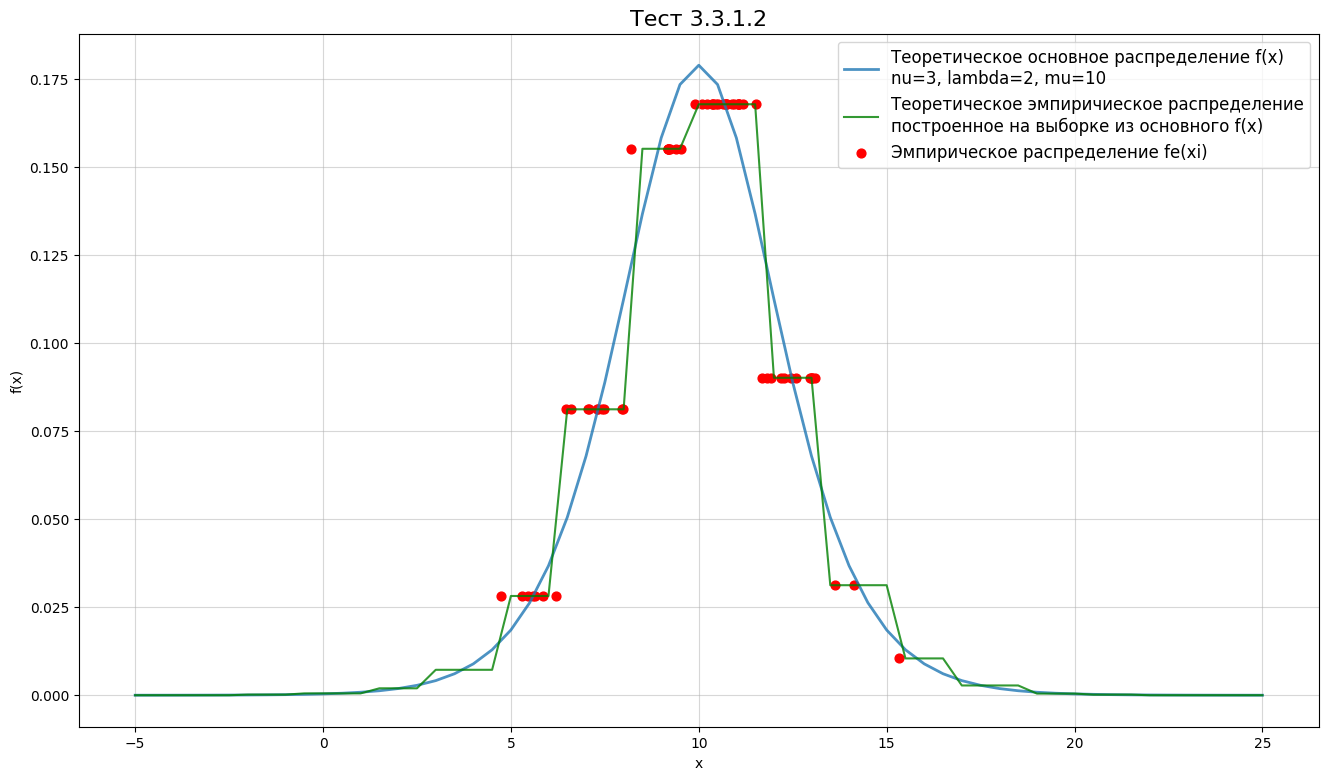
3.2.1

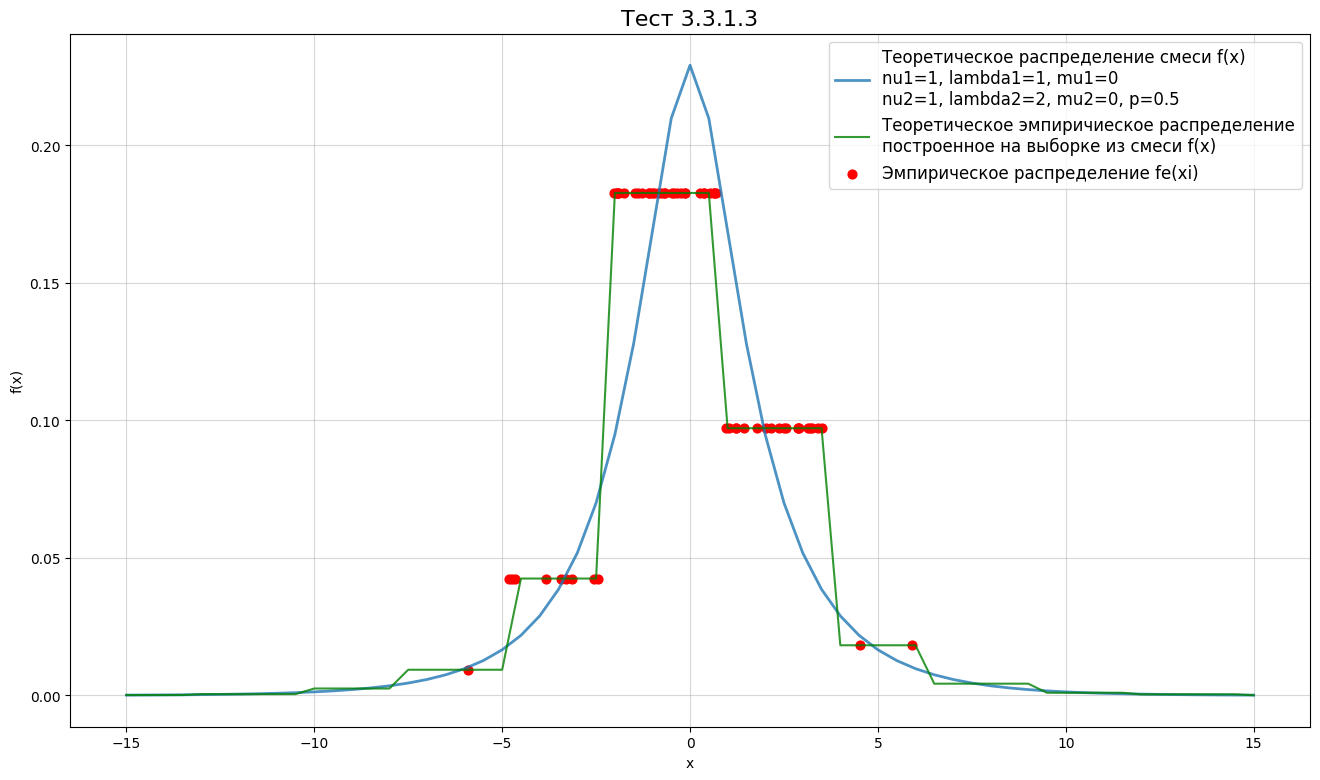
3.2.2

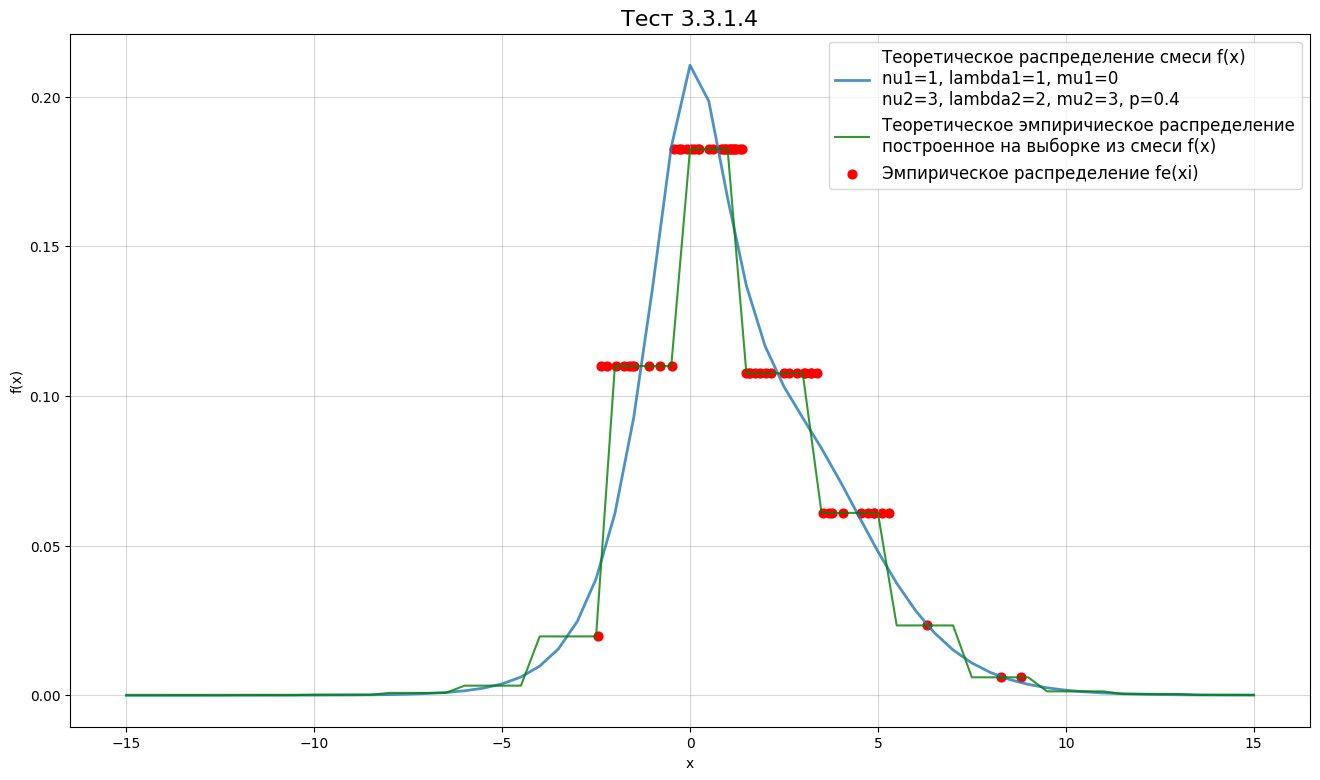
3.2.3

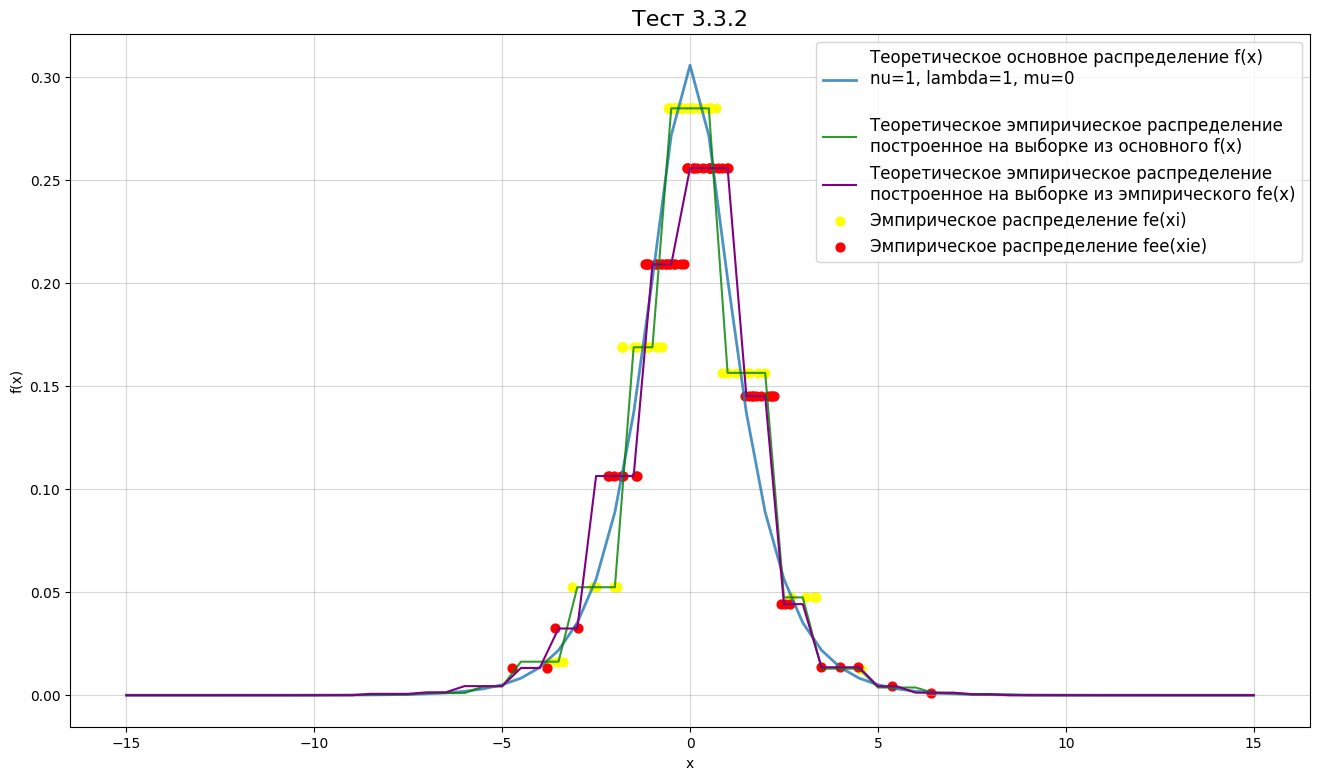
3.2.4

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 сложную предметную область, связанную с вероятностно-статистическим моделированием, и написали свою программу, реализующую статистическое моделирование с использованием структурного подхода в программировании.

**Приложение**

Код программы

**Объявление структур и функций (header.h):**

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

#include <cmath>

#include <fstream>

#include <iostream>

using ld = long double;

const ld pi = 3.14159265358979;

struct array\_t {

uint32\_t length;

ld\* data;

};

// Симметрическое гиперболическое распределение - функция плотности

struct Distribution {

ld nu, mu, lambda;

};

// инициализация

void initDistribution(Distribution& dist, ld nu, ld mu, ld lambda);

// значение функции плотности распределения от 1 аргумента

ld densityDist(const Distribution& dist, ld x);

// значение функции плотности распределения от массива аргументов

array\_t densityDistArray(const Distribution& dist, const array\_t& x);

// Математическое ожидание

ld MDist(const Distribution& dist);

// Дисперсия

ld DDist(const Distribution& dist);

// Коэффициент эксцесса

ld G2Dist(const Distribution& dist);

// Коэффициент ассиметрии

ld G1Dist();

// Моделирование случайной величины

ld XiDist(const Distribution& dist);

// Моделирование случайной величины

array\_t XiDistArray(const Distribution& dist, uint32\_t size);

// Смесь двух симметрических гиперболических распределений - функция плотности

struct Mixture {

ld p;

Distribution dist1, dist2;

};

// Инициализация смеси из двух распределений

void initMixture(Mixture& mixt, ld nu1, ld mu1, ld lambda1, ld nu2, ld mu2,

ld lambda2, ld p);

// значение функции плотности распределения от 1 аргумента

ld densityMixt(const Mixture& mixt, ld x);

// значение функции плотности распределения от массива аргументов

array\_t densityMixtArray(const Mixture& mixt, const array\_t& x);

// Математическое ожидание

ld MMixt(const Mixture& mixt);

// Дисперсия

ld DMixt(const Mixture& mixt);

// Коэффициент эксцесса

ld G2Mixt(const Mixture& mixt);

// Коэффициент ассиметрии

ld G1Mixt(const Mixture& mixt);

// Моделирование случайной величины

ld XiMixt(const Mixture& mixt);

// Моделирование случайной величины

array\_t XiMixtArray(const Mixture& mixt, uint32\_t size);

// Эмпирическое распределение - функция плотности

struct EmpiricDist {

array\_t X; // интервалы

array\_t xi; // начальная выборка

uint32\_t k; // количество интервалов

ld delta; // ширина интервала

uint32\_t\* n; // количество элементов в интервале

};

void initEmpiric(EmpiricDist& emp, const array\_t& x);

// значение функции плотности распределения от 1 аргумента

ld densityEmp(const EmpiricDist& emp, ld x);

// значение функции плотности распределения от массива аргументов

array\_t densityEmpArray(const EmpiricDist& emp, const array\_t& x);

// Математическое ожидание

ld MEmp(const EmpiricDist& emp);

// Дисперсия

ld DEmp(const EmpiricDist& emp);

// Коэффициент ассиметрии

ld G1Emp(const EmpiricDist& emp);

// Коэффициент эксцесса

ld G2Emp(const EmpiricDist& emp);

// Моделирование случайной величины

ld XiEmp(const EmpiricDist& emp);

// Моделирование случайной величины

array\_t XiEmpArray(const EmpiricDist& emp, uint32\_t size);

**Реализация функций основного распределения (dist.cpp):**

#include "header.h"

using namespace std;

namespace {

// Generate a random number in (0, 1)

ld \_randNum() {

ld r;

do {

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

} while (r == 0. || r == 1.);

return r;

}

} // namespace

void initDistribution(Distribution& dist, ld nu, ld mu, ld lambda) {

if (lambda <= 0 || nu <= 0)

throw invalid\_argument("Lambda and nu must be positive");

dist.nu = nu;

dist.mu = mu;

dist.lambda = lambda;

}

ld densityDist(const Distribution& dist, ld x) {

ld coeff =

2 \* dist.lambda \* sqrt(dist.nu) \* boost::math::cyl\_bessel\_k(1, dist.nu);

ld exponent =

exp(-dist.nu \* sqrt(1 + pow((x - dist.mu) / dist.lambda, 2) / dist.nu));

return exponent / coeff;

}

array\_t densityDistArray(const Distribution& dist, const array\_t& x) {

array\_t result;

result.length = x.length;

result.data = new ld[x.length];

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

result.data[i] = densityDist(dist, x.data[i]);

}

return result;

}

ld DDist(const Distribution& dist) {

return pow(dist.lambda, 2) \* boost::math::cyl\_bessel\_k(2, dist.nu) /

boost::math::cyl\_bessel\_k(1, dist.nu);

}

ld MDist(const Distribution& dist) { return dist.mu; }

ld G1Dist() { return 0; }

ld G2Dist(const Distribution& dist) {

return 3 \* boost::math::cyl\_bessel\_k(3, dist.nu) \*

boost::math::cyl\_bessel\_k(1, dist.nu) /

pow(boost::math::cyl\_bessel\_k(2, dist.nu), 2) -

3;

}

ld XiDist(const Distribution& dist) {

ld r1, r2, delta, t;

do {

r1 = \_randNum();

r2 = \_randNum();

delta = 2 \* (sqrt(1 + pow(dist.nu, 2)) - 1) / dist.nu;

t = -2 \* log(r1) / delta;

} while (-log(r2) <= (dist.nu - delta) \* t / 2 + dist.nu / (2 \* t) -

sqrt(dist.nu \* (dist.nu - delta)));

ld r3 = \_randNum(), r4 = \_randNum();

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

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

return z \* sqrt(t) \* dist.lambda + dist.mu;

}

array\_t XiDistArray(const Distribution& dist, uint32\_t size) {

array\_t result{size, new ld[size]};

for (uint32\_t i = 0; i < size; ++i) result.data[i] = XiDist(dist);

return result;

}

**Реализация смеси распределений (mixture.cpp):**

#include "header.h"

using namespace std;

namespace {

// Generate a random number in (0, 1)

ld \_randNum() {

ld r;

do {

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

} while (r == 0 || r == 1);

return r;

}

} // namespace

void initMixture(Mixture& mixt, ld nu1, ld mu1, ld lambda1, ld nu2, ld mu2,

ld lambda2, ld p) {

if (p < 0 || p > 1) throw invalid\_argument("p must be in [0, 1]");

initDistribution(mixt.dist1, nu1, mu1, lambda1);

initDistribution(mixt.dist2, nu2, mu2, lambda2);

mixt.p = p;

}

ld densityMixt(const Mixture& mixt, ld x) {

return (1 - mixt.p) \* densityDist(mixt.dist1, x) +

mixt.p \* densityDist(mixt.dist2, x);

}

array\_t densityMixtArray(const Mixture& mixt, const array\_t& x) {

array\_t result{x.length, new ld[x.length]};

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

result.data[i] = densityMixt(mixt, x.data[i]);

}

return result;

}

ld MMixt(const Mixture& mixt) {

return (1 - mixt.p) \* MDist(mixt.dist1) + mixt.p \* MDist(mixt.dist2);

}

ld DMixt(const Mixture& mixt) {

return (1 - mixt.p) \* (pow(MDist(mixt.dist1), 2) + DDist(mixt.dist1)) +

mixt.p \* (pow(MDist(mixt.dist2), 2) + DDist(mixt.dist2)) -

pow(MMixt(mixt), 2);

}

ld G1Mixt(const Mixture& mixt) {

ld m1 = MDist(mixt.dist1), m2 = MDist(mixt.dist2), d1 = DDist(mixt.dist1),

d2 = DDist(mixt.dist2);

ld m = MMixt(mixt);

return ((1 - mixt.p) \*

(pow(m1 - m, 3) + 3 \* (m1 - m) \* d1 + pow(d1, 1.5) \* G1Dist()) +

mixt.p \*

(pow(m2 - m, 3) + 3 \* (m2 - m) \* d2 + pow(d2, 1.5) \* G1Dist())) /

pow(DMixt(mixt), 1.5);

}

ld G2Mixt(const Mixture& mixt) {

ld m1 = MDist(mixt.dist1), m2 = MDist(mixt.dist2), d1 = DDist(mixt.dist1),

d2 = DDist(mixt.dist2);

ld m = MMixt(mixt);

return ((1 - mixt.p) \* (pow(m1 - m, 4) + 6 \* pow(m1 - m, 2) \* d1 +

4 \* (m1 - m) \* pow(d1, 1.5) \* G1Dist() +

pow(d1, 2) \* (G2Dist(mixt.dist1) + 3)) +

mixt.p \* (pow(m2 - m, 4) + 6 \* pow(m2 - m, 2) \* d2 +

4 \* (m2 - m) \* pow(d2, 1.5) \* G1Dist() +

pow(d2, 2) \* (G2Dist(mixt.dist2) + 3))) /

pow(DMixt(mixt), 2) -

3;

}

ld XiMixt(const Mixture& mixt) {

return (\_randNum() < (1 - mixt.p)) ? XiDist(mixt.dist1) : XiDist(mixt.dist2);

}

array\_t XiMixtArray(const Mixture& mixt, uint32\_t size) {

array\_t result{size, new ld[size]};

for (uint32\_t i = 0; i < size; ++i) result.data[i] = XiMixt(mixt);

return result;

}

**Реализация эмпирического распределения (empiric.cpp):**

#include "header.h"

using namespace std;

namespace {

ld \_min(const array\_t& x) {

ld m = x.data[0];

for (uint32\_t i = 1; i < x.length; ++i) {

if (x.data[i] < m) m = x.data[i];

}

return m;

}

ld \_max(const array\_t& x) {

ld m = x.data[0];

for (uint32\_t i = 1; i < x.length; ++i) {

if (x.data[i] > m) m = x.data[i];

}

return m;

}

// return random number in (0, 1)

ld \_randNum() {

ld r;

do {

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

} while (r == 0 || r == 1);

return r;

}

} // namespace

void initEmpiric(EmpiricDist& emp, const array\_t& x) {

emp.xi.length = x.length;

emp.xi.data = new ld[x.length];

for (uint32\_t i = 0; i < x.length; ++i) emp.xi.data[i] = x.data[i];

if (x.length == 0) throw invalid\_argument("Array length is 0");

emp.k = floor(log2(x.length)) + 1;

emp.n = new uint32\_t[emp.k];

for (uint32\_t i = 0; i < emp.k; ++i) emp.n[i] = 0;

emp.X.length = emp.k + 1;

emp.X.data = new ld[emp.X.length];

ld mn = \_min(x), mx = \_max(x);

emp.delta = (mx - mn) / emp.k;

/\*

std::cout << "Empiric distribution debug info:" << std::endl;

std::cout << "min = " << mn << " max = " << mx << " delta = " << emp.delta

<< " k = " << emp.k << std::endl;

\*/

emp.X.data[0] = mn;

emp.X.data[emp.X.length - 1] = mx;

for (uint32\_t i = 1; i < emp.X.length - 1; ++i)

emp.X.data[i] = emp.X.data[0] + i \* emp.delta;

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

for (uint32\_t j = 0; j < emp.k; ++j) {

if (x.data[i] >= emp.X.data[j] && x.data[i] < emp.X.data[j + 1]) {

emp.n[j] += 1;

break;

}

}

}

// Добавляем Xmax (самое правое значение)

emp.n[emp.k - 1]++;

/\*

for (uint32\_t i = 0; i < emp.k; ++i) {

std::cout << "[" << emp.X.data[i] << "; " << emp.X.data[i + 1]

<< ") : " << emp.n[i] << std::endl;

}\*/

}

ld densityEmp(const EmpiricDist& emp, ld x) {

if (x < emp.X.data[0] || x > emp.X.data[emp.X.length - 1]) return 0.0L;

for (uint32\_t i = 0; i < emp.k; ++i) {

if (x >= emp.X.data[i] && x < emp.X.data[i + 1])

return (ld)(emp.n[i]) / (emp.xi.length \* emp.delta);

}

if (x == emp.X.data[emp.X.length - 1])

return emp.n[emp.k - 1] / (emp.xi.length \* emp.delta);

return 0;

}

array\_t densityEmpArray(const EmpiricDist& emp, const array\_t& x) {

array\_t res{x.length, new ld[x.length]};

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

res.data[i] = densityEmp(emp, x.data[i]);

}

return res;

}

ld MEmp(const EmpiricDist& emp) {

ld sum = 0;

for (uint32\_t i = 0; i < emp.xi.length; ++i) sum += emp.xi.data[i];

return sum / emp.xi.length;

}

ld DEmp(const EmpiricDist& emp) {

ld m = MEmp(emp);

ld sum = 0;

for (uint32\_t i = 0; i < emp.xi.length; ++i)

sum += pow(emp.xi.data[i] - m, 2);

return sum / emp.xi.length;

}

ld G1Emp(const EmpiricDist& emp) {

ld m = MEmp(emp);

ld d = DEmp(emp);

ld sum = 0;

for (uint32\_t i = 0; i < emp.xi.length; ++i)

sum += pow(emp.xi.data[i] - m, 3);

return sum / emp.xi.length / pow(d, 1.5);

}

ld G2Emp(const EmpiricDist& emp) {

ld m = MEmp(emp);

ld d = DEmp(emp);

ld sum = 0;

for (uint32\_t i = 0; i < emp.xi.length; ++i)

sum += pow(emp.xi.data[i] - m, 4);

return sum / emp.xi.length / pow(d, 2) - 3;

}

ld XiEmp(const EmpiricDist& emp) {

ld r = \_randNum();

ld cumulative = 0;

for (uint32\_t i = 0; i < emp.k; ++i) {

cumulative += static\_cast<ld>(emp.n[i]) / emp.xi.length;

if (r < cumulative) {

return emp.X.data[i] + (emp.X.data[i + 1] - emp.X.data[i]) \* \_randNum();

}

}

return emp.X.data[emp.k - 1];

}

array\_t XiEmpArray(const EmpiricDist& emp, uint32\_t size) {

array\_t res{size, new ld[size]};

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

res.data[i] = XiEmp(emp);

}

return res;

}

**Код тестирования (main.cpp):**

#include <locale>

#include "header.h"

using namespace std;

void testDistribution\_3\_1\_1() {

ld mu = 0, lambda = 1, nu = 1;

ld x = 0;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

ld density = densityDist(dist, x);

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) = " << density << endl;

cout << "M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

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

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

auto xi = XiDist(dist);

fout << i << "," << densityDist(dist, i) << "," << xi << ","

<< densityDist(dist, xi) << endl;

}

}

void testDistribution\_3\_1\_2() {

ld mu = 0, lambda = 2, nu = 1;

ld x = 0;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

ld density = densityDist(dist, x);

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) = " << density << endl;

cout << "M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

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

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

auto xi = XiDist(dist);

fout << i << "," << densityDist(dist, i) << "," << xi << ","

<< densityDist(dist, xi) << endl;

}

}

void testDistribution\_3\_1\_3() {

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

ld x = 0;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

ld density = densityDist(dist, x);

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) = " << density << endl;

cout << "M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

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

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

auto xi = XiDist(dist);

fout << i << "," << densityDist(dist, i) << "," << xi << ","

<< densityDist(dist, xi) << endl;

}

}

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 mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

ld density = densityMixt(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) = " << density << endl;

cout << "M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << 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 = XiMixt(mixture);

fout << i << "," << densityMixt(mixture, i) << "," << xi << ","

<< densityMixt(mixture, 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 mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

ld density = densityMixt(mixture, x);

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) = " << density << endl;

cout << "M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << 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 = XiMixt(mixture);

fout << i << "," << densityMixt(mixture, i) << "," << xi << ","

<< densityMixt(mixture, 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 mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

ld density = densityMixt(mixture, x);

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) = " << density << endl;

cout << "M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << 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 = XiMixt(mixture);

fout << i << "," << densityMixt(mixture, i) << "," << xi << ","

<< densityMixt(mixture, 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 mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

ld density = densityMixt(mixture, x);

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) = " << density << endl;

cout << "M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << 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 = XiMixt(mixture);

fout << i << "," << densityMixt(mixture, i) << "," << xi << ","

<< densityMixt(mixture, xi) << endl;

}

}

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

ld mu = 0, lambda = 1, nu = 1;

ld x = 0;

uint32\_t n = 10000;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

EmpiricDist emp;

auto arr = XiDistArray(dist, n);

initEmpiric(emp, 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) = " << densityDist(dist, x);

cout << ", M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

cout << ", M = " << MEmp(emp) << ", D = " << DEmp(emp)

<< ", G1 = " << G1Emp(emp) << ", G2 = " << G2Emp(emp) << 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 = XiEmp(emp);

fout << i << "," << densityDist(dist, i) << "," << densityEmp(emp, i) << ","

<< xi << "," << densityEmp(emp, xi) << endl;

}

delete[] arr.data;

delete[] emp.X.data;

delete[] emp.n;

delete[] emp.xi.data;

}

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

ld mu = 10, lambda = 2, nu = 3;

ld x = 0;

uint32\_t n = 10000;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

EmpiricDist emp;

auto arr = XiDistArray(dist, n);

initEmpiric(emp, 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) = " << densityDist(dist, x);

cout << ", M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

cout << ", M = " << MEmp(emp) << ", D = " << DEmp(emp)

<< ", G1 = " << G1Emp(emp) << ", G2 = " << G2Emp(emp) << 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 = XiEmp(emp);

fout << i << "," << densityDist(dist, i) << "," << densityEmp(emp, i) << ","

<< xi << "," << densityEmp(emp, xi) << endl;

}

delete[] arr.data;

delete[] emp.X.data;

delete[] emp.n;

delete[] emp.xi.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;

Mixture mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

EmpiricDist emp;

auto arr = XiMixtArray(mixture, n);

initEmpiric(emp, 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) = " << densityMixt(mixture, x);

cout << ", M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << endl;

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

cout << ", M = " << MEmp(emp) << ", D = " << DEmp(emp)

<< ", G1 = " << G1Emp(emp) << ", G2 = " << G2Emp(emp) << 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 = XiEmp(emp);

fout << i << "," << densityMixt(mixture, i) << "," << densityEmp(emp, i)

<< "," << xi << "," << densityEmp(emp, xi) << endl;

}

delete[] arr.data;

delete[] emp.X.data;

delete[] emp.n;

delete[] emp.xi.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;

Mixture mixture;

initMixture(mixture, nu1, mu1, lambda1, nu2, mu2, lambda2, p);

EmpiricDist emp;

auto arr = XiMixtArray(mixture, n);

initEmpiric(emp, 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) = " << densityMixt(mixture, x);

cout << ", M = " << MMixt(mixture) << ", D = " << DMixt(mixture)

<< ", G1 = " << G1Mixt(mixture) << ", G2 = " << G2Mixt(mixture) << endl;

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

cout << ", M = " << MEmp(emp) << ", D = " << DEmp(emp)

<< ", G1 = " << G1Emp(emp) << ", G2 = " << G2Emp(emp) << 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 = XiEmp(emp);

fout << i << "," << densityMixt(mixture, i) << "," << densityEmp(emp, i)

<< "," << xi << "," << densityEmp(emp, xi) << endl;

}

delete[] arr.data;

delete[] emp.X.data;

delete[] emp.n;

delete[] emp.xi.data;

}

void testDistribution\_3\_3\_2() {

ld mu = 0, lambda = 1, nu = 1;

ld x = 0;

uint32\_t n = 10000;

Distribution dist;

initDistribution(dist, nu, mu, lambda);

EmpiricDist emp;

auto arr = XiDistArray(dist, n);

initEmpiric(emp, arr);

delete[] arr.data;

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) = " << densityDist(dist, x);

cout << ", M = " << MDist(dist) << ", D = " << DDist(dist)

<< ", G1 = " << G1Dist() << ", G2 = " << G2Dist(dist) << endl;

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

cout << ", M = " << MEmp(emp) << ", D = " << DEmp(emp)

<< ", G1 = " << G1Emp(emp) << ", G2 = " << G2Emp(emp) << endl;

EmpiricDist emp2;

auto arr2 = XiEmpArray(emp, n);

initEmpiric(emp2, arr2);

cout << "Эмпирическое распределение2:\nf(0) = " << densityEmp(emp2, x);

cout << ", M = " << MEmp(emp2) << ", D = " << DEmp(emp2)

<< ", G1 = " << G1Emp(emp2) << ", G2 = " << G2Emp(emp2) << 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 = XiEmp(emp);

auto xie = XiEmp(emp2);

fout << i << "," << densityDist(dist, i) << "," << densityEmp(emp, i) << ","

<< xi << "," << densityEmp(emp, xi) << "," << densityEmp(emp2, i)

<< "," << xie << "," << densityEmp(emp2, xie) << endl;

}

delete[] arr2.data;

delete[] emp.X.data;

delete[] emp.n;

delete[] emp.xi.data;

delete[] emp2.X.data;

delete[] emp2.n;

delete[] emp2.xi.data;

}

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) {

Distribution dist;

initDistribution(dist, nu, mu, lambda);

cout << " D = " << DDist(dist) << " G2 = " << G2Dist(dist)

<< " f(0) = " << densityDist(dist, 0) << endl;

}

}

int main() {

std::setlocale(LC\_ALL, "ru\_RU.UTF-8");

testDistribution\_3\_1\_1();

testDistribution\_3\_1\_2();

testDistribution\_3\_1\_3();

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;

}