Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ПМ

УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Практическое задание № 1

# Решение эллиптических краевых задач методом конечных разностей

Факультет: ПМИ Преподаватели:

Задорожный А. Г.,

Патрушев И.И.

Группа: ПМ-81

Студенты: Ефремов А.А.,

Ртищева К. С.

Бригада: 1

Вариант: 5

Новосибирск

2021

1. **Цель работы**

Разработать программу решения эллиптической краевой задачи методом конечных разностей. Протестировать программу и численно оценить порядок аппроксимации.

1. **Задание**

Область может иметь любую форму. Предусмотреть учет первых и вторых краевых условий.

1. **Анализ задачи**

Эллиптическая краевая задача для функции определяется дифференциальным уравнением

Заданным в двумерной области Ω с границей , и краевым условием

Для двумерного оператора Лапласа дискретный аналог на неравномерной прямоугольной сетке может быть представлен пятиточечным разностным выражением

Подставим данное разностное выражение в дифференциальное уравнение и получим:

**Учет первых краевых условий:**

в матрице СЛАУ в строке на место диагонального элемента ставится единица, все остальные элементы этой строки матрицы обнуляются, а компоненте вектора правой части присваивается значение .

1. **Структура входных данных**

Первое число n в файле со входными данными - количество прямоугольных подобластей (регионов). Далее идет n наборов строк, описывающих каждый регион. В первой строке набора задаются 4 числа - координаты левой, правой, нижней и верхней границы региона. Во второй строке идут две пары чисел q и n - коэффициент разрядки и число шагов для дробления сетки по оси x и y. В третьей строке идут 4 числа m для описания условий для левой, правой, нижней и верхней границ расчетной области. Число m может быть 1 - тогда на границе будет 1 краевое условие, или номером региона, с которым граничит данное ребро, взятым с обратным знаком.

1. **Текст программы**

***Файл “SLAE.h”***

#pragma once

#include <vector>

using namespace std;

class SLAE

{

public:

vector<vector<double>> matrix; // Матрица системы

vector<vector<int>> index; // Индексы столбцов

vector<double> f; // Вектор правой части

const int D = 5; // Количество диагоналей матрицы

int N = 0; // Размерность матрицы

vector<double> xk, xk1; // Вспомогательные векторы

SLAE(const int& t\_n)

{

N = t\_n;

index.resize(D);

for(int d = 0; d < D; d++)

index[d].resize(N);

matrix.resize(D);

for(int d = 0; d < D; d++)

matrix[d].resize(N);

xk.resize(N);

xk1.resize(N);

f.resize(N);

}

// Умножение матрицы системы на вектор vec,

// результат в res

void Multiplication(vector<double>& vec, vector<double>& res)

{

int n = vec.size(), k = 0;

for(int j = 0; j < n; j++)

{

for(int d = 0; d < D; d++)

{

k = index[d][j];

if(k < 0 && j + k > 0 ||

k > 0 && j + k < n)

res[j] += matrix[d][j] \* vec[k + j];

}

}

}

// Норма вектора

double Norm(const vector<double>& vec)

{

double res = 0;

for(int i = 0; i < N; i++)

res += vec[i] \* vec[i];

return sqrt(res);

}

// Получение относительной невязки системы

double RelativeResidual(vector<double>& vec)

{

vector<double> mult(N);

Multiplication(vec, mult);

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

mult[i] = f[i] - mult[i];

return Norm(mult) / Norm(f);

}

// Итерационный процесс метода Гаусса-Зейделя

void IterativeProcess(const int& j, double& sum)

{

int k = 0, n = xk.size();

for(int i = 0; i < D; i++)

{

k = index[i][j];

if(k + j >= 0 && k + j < n)

{

if(i < 3) // нижний треугольник

sum += matrix[i][j] \* xk1[k + j];

else // верхний треугольник

sum += matrix[i][j] \* xk[k + j];

}

}

}

// Решение системы методом Гаусса-Зейделя

void GaussSeidel(const int& max\_iter, const double& eps,

const double& relax)

{

double residual = 0.0, sum = 0.0;

residual = RelativeResidual(xk);

for(int k = 0; k < max\_iter && residual > eps; k++)

{

for(int j = 0; j < N; j++)

{

IterativeProcess(j, sum);

xk1[j] = xk[j] + (relax / matrix[2][j]) \* (f[j] - sum);

sum = 0.;

}

xk.swap(xk1);

residual = RelativeResidual(xk);

}

}

};

***Файл “Region.h”***

#pragma once

#include <vector>

using namespace std;

struct Region

{

double left, right, top, bot; // Границы областей

vector<double> x\_node; // Координаты узлов по X

vector<double> y\_node; // Координаты узлов по Y

int n\_nodes; // Количество узлов

int n\_x; // Количество узлов по X

int n\_y; // Количество узлов по Y

int first, last; // Индексы первого и последнего

// узлов в глобальной нумерации

// Массив c информацией о краевых условиях региона

// 0 - нижнее

// 1 - правое

// 2 - верхнее

// 3 - левое

vector<int> borders;

};

***Файл “EllipticalProblem.h”***

#pragma once

#include <vector>

#include <fstream>

#include <string>

#include <iomanip>

#include "SLAE.h"

#include "Test.h"

#include "Region.h"

using namespace std;

class EllipticalProblem

{

public:

vector<Region> regions; // Регионы расчетной области

int n\_regions = 0; // Количество регионов

int n\_nodes = 0; // Общее количество узлов

vector<vector<int>> borders; // Информация о граничных условиях

SLAE\* slae; // Система

Test test; // Тестовая информация

EllipticalProblem()

{

}

~EllipticalProblem()

{

delete slae;

}

// Функция считывания областей из файла FILE\_NAME

// и формирования сетки

void ReadFormGrid(const string& FILE\_NAME)

{

ifstream fin(FILE\_NAME);

fin >> n\_regions;

string s;

regions.resize(n\_regions);

for(int reg\_i = 0; reg\_i < n\_regions; reg\_i++)

{

fin >> s;

Region\* r = &regions[reg\_i];

// Считывание границы области

fin >> r->left;

fin >> r->right;

fin >> r->bot;

fin >> r->top;

// Генерация координат узлов по X

int n;

double h, q;

fin >> q >> n;

r->n\_x = n + 1;

r->x\_node.resize(r->n\_x);

h = r->right - r->left;

if(q != 1)

h \*= (1 - q) / (1 - pow(q, n));

else

h /= n;

r->x\_node[0] = r->left;

for(int i = 0; i < n; i++)

r->x\_node[i + 1] = r->x\_node[i] + h \* pow(q, i);

// Генерация координат узлов по Y

fin >> q >> n;

r->n\_y = n + 1;

r->y\_node.resize(r->n\_y);

h = r->top - r->bot;

if(q != 1)

h \*= (1 - q) / (1 - pow(q, n));

else

h /= n;

r->y\_node[0] = r->bot;

for(int i = 0; i < n; i++)

r->y\_node[i + 1] = r->y\_node[i] + h \* pow(q, i);

if(reg\_i != 0)

r->first = regions[reg\_i - 1].last + 1;

else

r->first = 0;

r->n\_nodes = r->n\_x \* r->n\_y;

r->last = r->first + r->n\_nodes - 1;

n\_nodes += r->n\_nodes;

r->borders.resize(4);

// Считывание информации о краевых условиях

for(int bord\_i = 0; bord\_i < 4; bord\_i++)

fin >> r->borders[bord\_i];

}

fin.close();

}

// Формирование матрицы системы

void FormMatrix()

{

// Проход по всем регионам

for(int reg\_i = 0; reg\_i < n\_regions; reg\_i++)

{

Region\* r = &regions[reg\_i];

// Проход по всем узлам региона

for(int node\_i = 0; node\_i < r->n\_nodes; node\_i++)

{

// Индекс узла в глобальной нумерации

int global\_i = node\_i + r->first;

// Индексы центрального узла

int x\_cent = node\_i % r->n\_x;

int y\_cent = floor(node\_i / r->n\_x);

// Обработка некраевых узлов

if(0 < x\_cent && x\_cent < r->n\_x - 1 &&

0 < y\_cent && y\_cent < r->n\_y - 1)

{

// Приросты по X

double hi = r->x\_node[x\_cent + 1] - r->x\_node[x\_cent + 0];

double hi1 = r->x\_node[x\_cent - 0] - r->x\_node[x\_cent - 1];

// Приросты по Y

double hj = r->y\_node[y\_cent + 1] - r->y\_node[y\_cent + 0];

double hj1 = r->y\_node[y\_cent - 0] - r->y\_node[y\_cent - 1];

// Нижний узел

slae->matrix[0][global\_i] = -test.lambda() \*

(2.0 / (hj1 \* (hj + hj1)));

// Левый узел

slae->matrix[1][global\_i] = -test.lambda() \*

(2.0 / (hi1 \* (hi + hi1)));

// Центральный узел

slae->matrix[2][global\_i] = +test.lambda() \*

(2.0 / (hi1 \* hi) + 2.0 / (hj1 \* hj)) + test.gamma();

// Правый узел

slae->matrix[3][global\_i] = -test.lambda() \*

(2.0 / (hi \* (hi + hi1)));

// Верхний узел

slae->matrix[4][global\_i] = -test.lambda() \*

(2.0 / (hj \* (hj + hj1)));

// Вектор правой части

slae->f[global\_i] = test.f(r->x\_node[x\_cent], r->y\_node[y\_cent]);

}

// Обработка краевых узлов

else

{

int border\_x = 0, border\_y = 0;

if(x\_cent == 0)

border\_x = r->borders[0];

else if(x\_cent == r->n\_x - 1)

border\_x = r->borders[1];

if(y\_cent == 0)

border\_y = r->borders[2];

else if(y\_cent == r->n\_y - 1)

border\_y = r->borders[3];

// Если узел на границе между соседями

if(border\_x != 1 && border\_y != 1 ||

border\_x != 1 && border\_y == 0 ||

border\_x == 0 && border\_y != 1)

{

double hi = 0, hi1 = 0, hj = 0, hj1 = 0;

int neib\_x = 0;

int neib\_y = 0;

int neib\_left, neib\_right, neib\_bot, neib\_top;

// Если есть сосед по X

if(border\_x != 0)

{

neib\_x = -border\_x - 1;

// Сосед слева

if(x\_cent == 0)

{

neib\_left = regions[neib\_x].n\_x \* (y\_cent + 1) - 2;

slae->index[1][global\_i] = -abs(global\_i -

(regions[neib\_x].first + neib\_left));

hi = r->x\_node[x\_cent + 1] - r->x\_node[x\_cent + 0];

hi1 = r->x\_node[x\_cent - 0] –

regions[neib\_x].x\_node[regions[neib\_x].n\_x - 2];

}

// Сосед справа

if(x\_cent == r->n\_x - 1)

{

neib\_right = regions[neib\_x].n\_x \* y\_cent + 1;

slae->index[3][global\_i] = abs(global\_i -

(regions[neib\_x].first + neib\_right));

hi = regions[neib\_x].x\_node[1] - r->x\_node[x\_cent + 0];

hi1 = r->x\_node[x\_cent - 0] - r->x\_node[x\_cent - 1];

}

if(border\_y == 0)

{

hj = r->y\_node[y\_cent + 1] - r->y\_node[y\_cent];

hj1 = r->y\_node[y\_cent] - r->y\_node[y\_cent - 1];

}

}

// Если есть сосед по Y

if(border\_y != 0)

{

neib\_y = -border\_y - 1;

// Сосед снизу

if(y\_cent == 0)

{

neib\_bot = regions[neib\_y].n\_x \* (regions[neib\_y].n\_y - 2) +

x\_cent;

slae->index[0][global\_i] = -abs(global\_i -

(regions[neib\_y].first + neib\_bot));

hj = r->y\_node[y\_cent + 1] - r->y\_node[y\_cent + 0];

hj1 = r->y\_node[y\_cent - 0] –

regions[neib\_y].y\_node[regions[neib\_y].n\_y - 2];

}

// Сосед сверху

if(y\_cent == r->n\_y - 1)

{

neib\_top = regions[neib\_y].n\_x + x\_cent;

slae->index[4][global\_i] = abs(global\_i -

(regions[neib\_y].first + neib\_top));

hj = regions[neib\_y].y\_node[1] - r->y\_node[y\_cent + 0];

hj1 = r->y\_node[y\_cent - 0] - r->y\_node[y\_cent - 1];

}

if(border\_x == 0)

{

hi = r->x\_node[x\_cent + 1] - r->x\_node[x\_cent + 0];

hi1 = r->x\_node[x\_cent - 0] - r->x\_node[x\_cent - 1];

}

}

// Нижний узел

slae->matrix[0][global\_i] = -test.lambda() \*

(2.0 / (hj1 \* (hj + hj1)));

// Левый узел

slae->matrix[1][global\_i] = -test.lambda() \*

(2.0 / (hi1 \* (hi + hi1)));

// Центральный узел

slae->matrix[2][global\_i] = +test.lambda() \*

(2.0 / (hi1 \* hi) + 2.0 / (hj1 \* hj)) + test.gamma();

// Правый узел

slae->matrix[3][global\_i] = -test.lambda() \*

(2.0 / (hi \* (hi + hi1)));

// Верхний узел

slae->matrix[4][global\_i] = -test.lambda() \*

(2.0 / (hj \* (hj + hj1)));

// Вектор правой части

slae->f[global\_i] = test.f(r->x\_node[x\_cent], r->y\_node[y\_cent]);

}

}

}

}

// Обработка первого кревого условия

// Проход по всем регионам

for(int reg\_i = 0; reg\_i < n\_regions; reg\_i++)

{

Region\* r = &regions[reg\_i];

// Проход по всем узлам региона

for(int node\_i = 0; node\_i < r->n\_nodes; node\_i++)

{

// Индекс узла в глобальной нумерации

int global\_i = node\_i + r->first;

// Индексы центрального узла

int x\_cent = node\_i % r->n\_x;

int y\_cent = floor(node\_i / r->n\_x);

// Обработка некраевых узлов

if(x\_cent == 0 || x\_cent == r->n\_x - 1 ||

0 == y\_cent || y\_cent == r->n\_y - 1)

{

int border\_x = 0, border\_y = 0;

if(x\_cent == 0)

border\_x = r->borders[0];

else if(x\_cent == r->n\_x - 1)

border\_x = r->borders[1];

if(y\_cent == 0)

border\_y = r->borders[2];

else if(y\_cent == r->n\_y - 1)

border\_y = r->borders[3];

// Первое краевое

if(border\_x == 1 || border\_y == 1)

{

slae->matrix[0][global\_i] = 0;

slae->matrix[1][global\_i] = 0;

slae->matrix[2][global\_i] = 1.0;

slae->matrix[3][global\_i] = 0;

slae->matrix[4][global\_i] = 0;

slae->f[global\_i] = test.u(r->x\_node[x\_cent], r->y\_node[y\_cent]);

slae->index[0][global\_i] = -r->n\_x;

slae->index[1][global\_i] = -1;

slae->index[2][global\_i] = 0;

slae->index[3][global\_i] = 1;

slae->index[4][global\_i] = r->n\_x;

}

}

}

}

}

void PrintSolution(const string& file\_name)

{

ofstream fout(file\_name);

double norm = 0.0, norm\_u = 0.0;

fout << " y x calc prec";

fout << " dif N reg location" << endl << fixed;

// Проход по всем регионам

for(int reg\_i = 0; reg\_i < n\_regions; reg\_i++)

{

Region\* r = &regions[reg\_i];

// Проход по всем узлам региона

for(int node\_i = 0; node\_i < r->n\_nodes; node\_i++)

{

// Индекс узла в глобальной нумерации

int global\_i = node\_i + r->first;

// Индексы центрального узла

int x\_cent = node\_i % r->n\_x;

int y\_cent = floor(node\_i / r->n\_x);

fout << setw(9) << r->y\_node[y\_cent];

fout << setw(11) << r->x\_node[x\_cent];

double calc = slae->xk[global\_i];

fout << setw(15) << calc;

double prec = test.u(r->x\_node[x\_cent], r->y\_node[y\_cent]);

fout << setw(15) << prec;

fout << setw(14) << scientific << abs(prec - calc);

fout << fixed << setw(5) << global\_i << setw(4) << reg\_i + 1;

// Обработка некраевых узлов

if(0 < x\_cent && x\_cent < r->n\_x - 1 &&

0 < y\_cent && y\_cent < r->n\_y - 1)

fout << " inner";

else

{

int border\_x = 0, border\_y = 0;

if(x\_cent == 0)

border\_x = r->borders[0];

else if(x\_cent == r->n\_x - 1)

border\_x = r->borders[1];

if(y\_cent == 0)

border\_y = r->borders[2];

else if(y\_cent == r->n\_y - 1)

border\_y = r->borders[3];

// Первое краевое

if(border\_x == 1 || border\_y == 1)

fout << " border";

else

if(border\_x != 1 && border\_y != 1 ||

border\_x != 1 && border\_y == 0 ||

border\_x == 0 && border\_y != 1)

fout << " inner border";

}

fout << endl;

norm\_u += prec \* prec;

norm += abs(calc - prec) \* abs(calc - prec);

}

}

fout << "||u-u\*||/||u\*|| = " << scientific << sqrt(norm) / sqrt(norm\_u) << endl;

fout << "||u-u\*|| = " << scientific << sqrt(norm);

fout.close();

}

};

***Файл “Test.h”***

#pragma once

using namespace std;

class Test

{

public:

int N;

Test(const int& t\_N) : N(t\_N) {};

Test() : N(0) {};

double f(const double& x, const double& y)

{

switch(N)

{

case(0): return (0)\* lambda() + u(x, y) \* gamma();

case(1): return (0)\* lambda() + u(x, y) \* gamma();

case(2): return (-4)\* lambda() + u(x, y) \* gamma();

case(3): return (-6 \* x - 6 \* y) \* lambda() + u(x, y) \* gamma();

case(4): return (-12 \* x \* x - 12 \* y \* y) \* lambda() + u(x, y) \* gamma();

};

}

double lambda()

{

return 1;

}

double gamma()

{

return 1;

}

double u(const double& x, const double& y)

{

switch(N)

{

case(0): return 2.0;

case(1): return x + y;

case(2): return x \* x + y \* y;

case(3): return x \* x \* x + y \* y \* y;

case(4): return x \* x \* x \* x + y \* y \* y \* y;

};

}

};

***Файл “Main.cpp”***

#include <iostream>

#include "EllipticalProblem.h"

using namespace std;

void main()

{

EllipticalProblem ep = EllipticalProblem();

ep.ReadFormGrid("regions.txt");

// Инициализация СЛАУ

ep.slae = new SLAE(ep.n\_nodes);

for(int reg\_i = 0; reg\_i < ep.n\_regions; reg\_i++)

{

Region\* r = &ep.regions[reg\_i];

for(int node\_i = 0; node\_i < r->n\_nodes; node\_i++)

{

int global\_i = node\_i + r->first;

ep.slae->index[0][global\_i] = -r->n\_x;

ep.slae->index[1][global\_i] = -1;

ep.slae->index[2][global\_i] = 0;

ep.slae->index[3][global\_i] = 1;

ep.slae->index[4][global\_i] = r->n\_x;

}

}

// Инициализация тестовых данных

ep.test = Test(3);

// Формирование матрицы системы

ep.FormMatrix();

ep.slae->GaussSeidel(10000, 1e-14, 0.65);

ep.PrintSolution("res.txt");

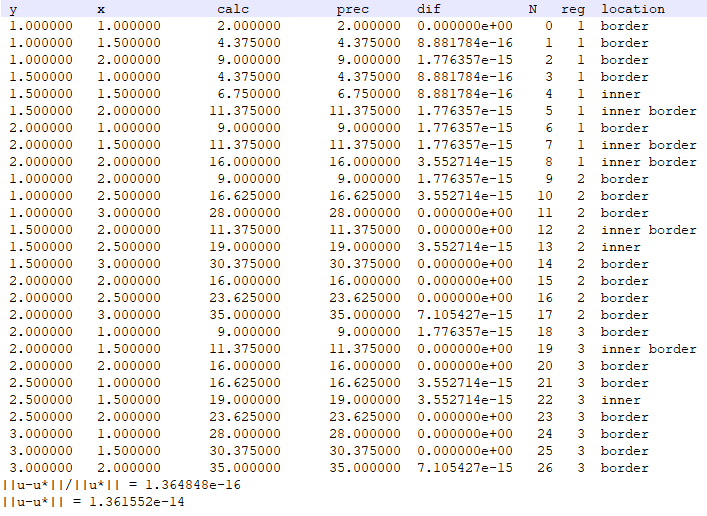
}

1. **Тестирование на равномерной сетке**

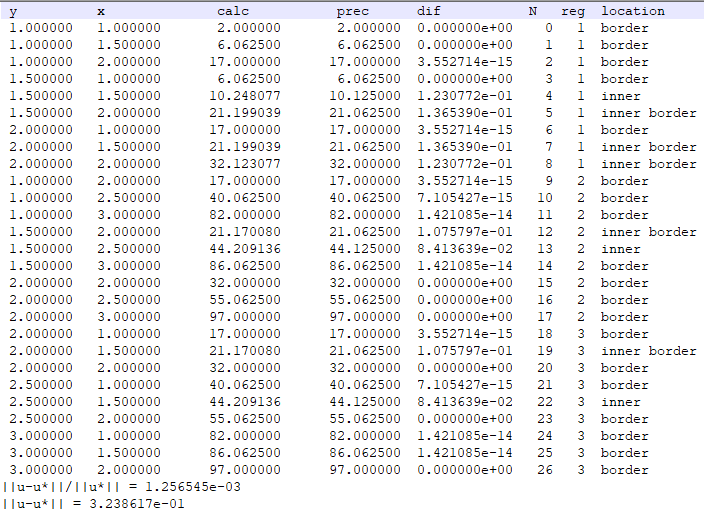
**Область имеет L-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3 | ***Файл regions.txt***  3  -1-  1 2 1 2  1 2 1 2  1 -2 1 -3  -2-  2 3 1 2  1 2 1 2  -1 1 1 1  -3-  1 2 2 3  1 2 1 2  1 1 -1 1 |

* λ = 1, γ = 1,



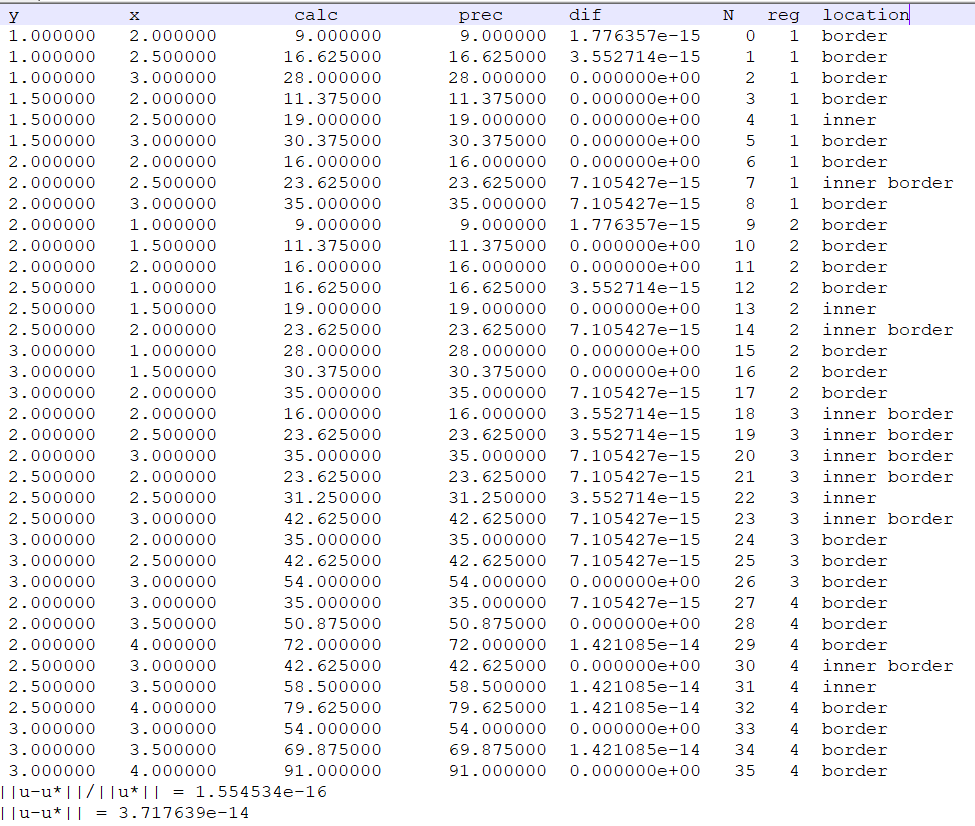
* λ = 1, γ = 1,



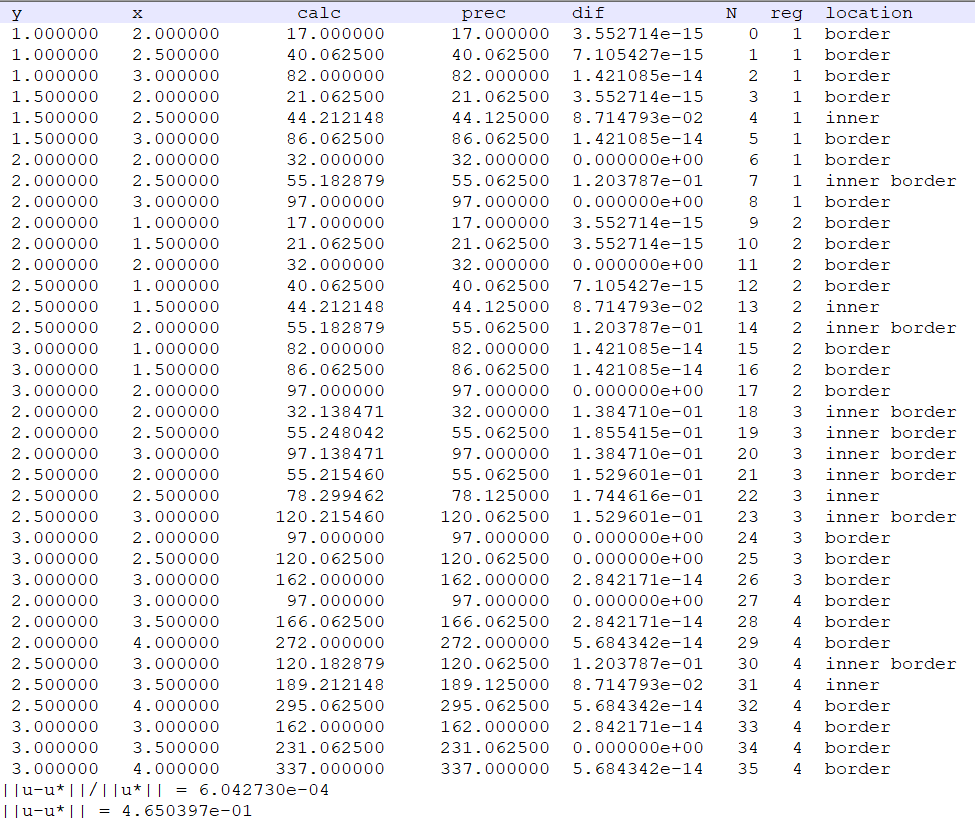
**Область имеет Т-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  4 | ***Файл regions.txt***  4  -1-  2 3 1 2  1 2 1 2  1 1 1 -3  -2-  1 2 2 3  1 2 1 2  1 -3 1 1  -3-  2 3 2 3  1 2 1 2  -2 -4 -1 1  -4-  3 4 2 3  1 2 1 2  -3 1 1 1 |

* λ = 1, γ = 1,



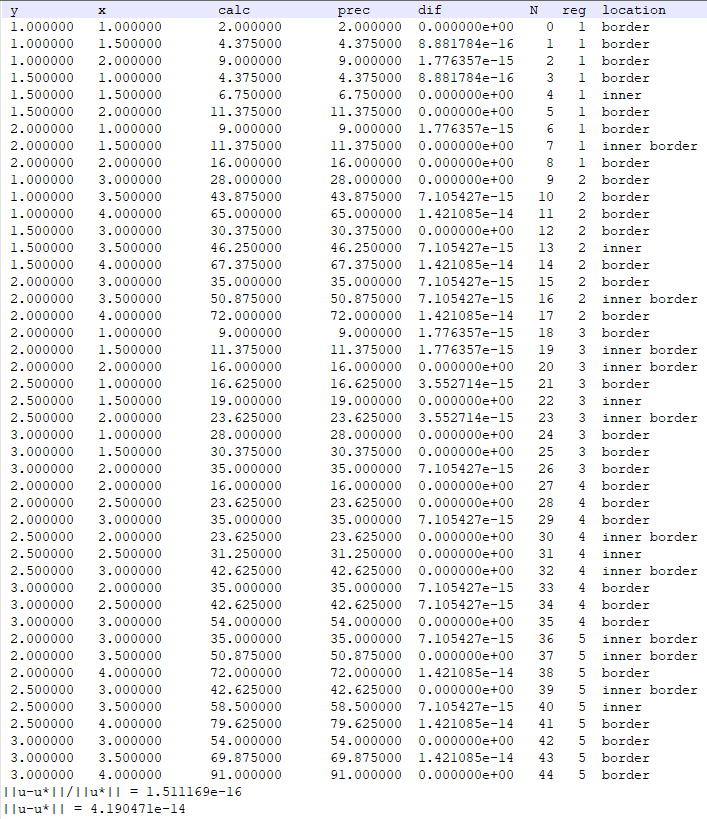
* λ = 1, γ = 1,



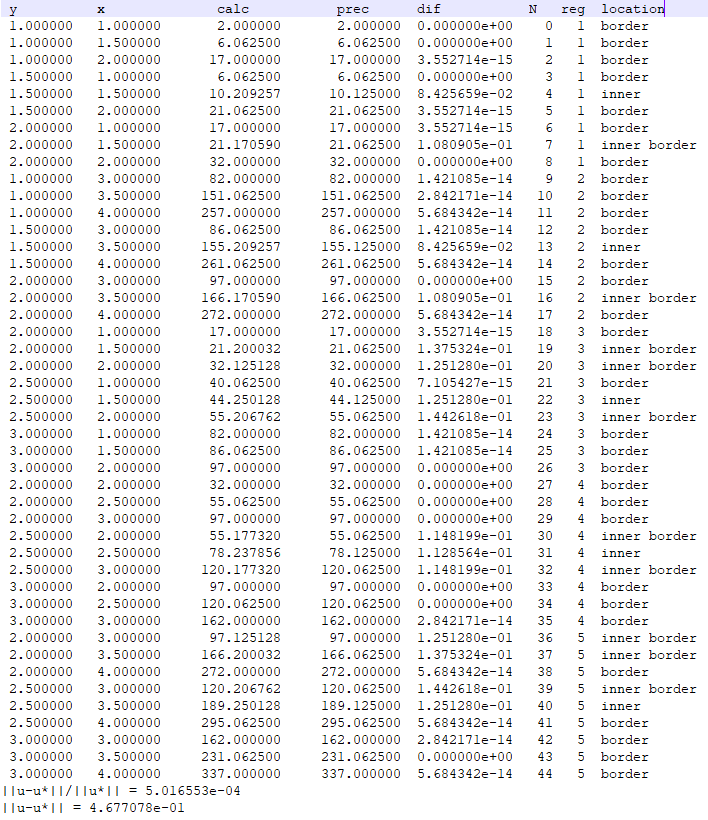
**Область имеет П-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  4  5 | ***Файл regions.txt***  5  -1-  1 2 1 2  1 2 1 2  1 1 1 -3  -2-  3 4 1 2  1 2 1 2  1 1 1 -5  -3-  1 2 2 3  1 2 1 2  1 -4 -1 1  -4-  2 3 2 3  1 2 1 2  -3 -5 1 1  -5-  3 4 2 3  1 2 1 2  -4 1 -2 1 |

* λ = 1, γ = 1,



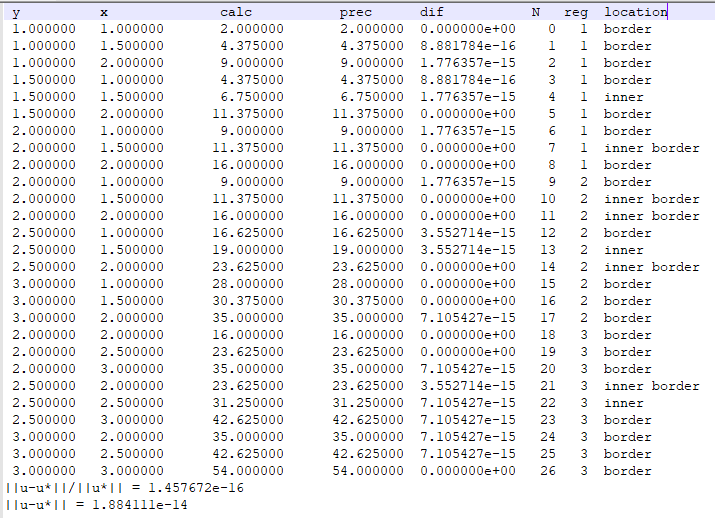
* λ = 1, γ = 1,



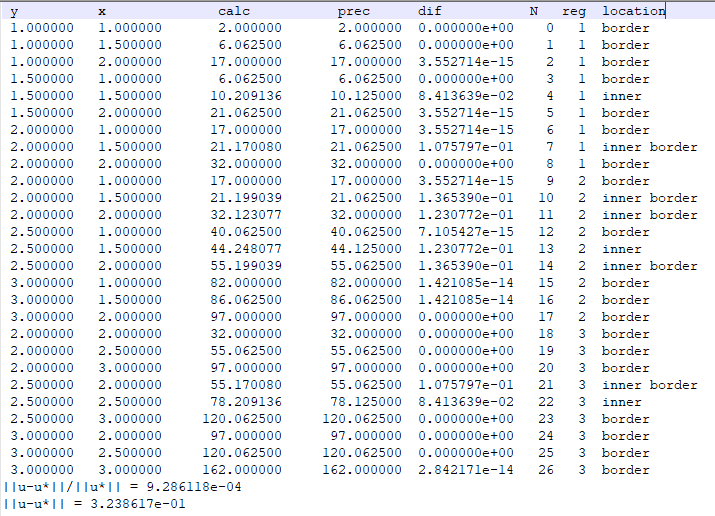
**Область имеет Г-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3 | ***Файл regions.txt***  3  -1-  1 2 1 2  1 2 1 2  1 1 1 -2  -2-  1 2 2 3  1 2 1 2  1 -3 -1 1  -3-  2 3 2 3  1 2 1 2  -2 1 1 1 |

* λ = 1, γ = 1,



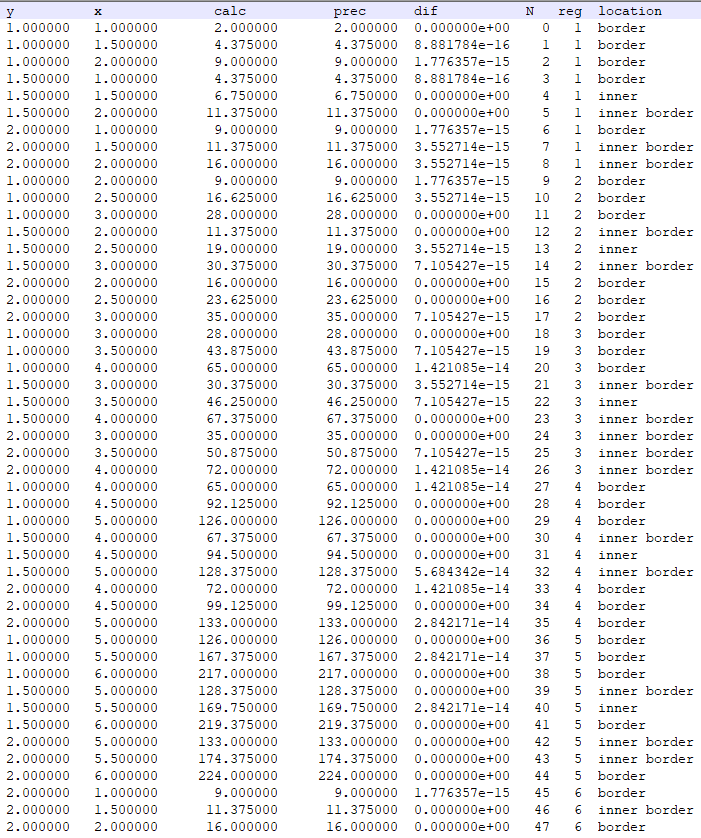
* λ = 1, γ = 1,

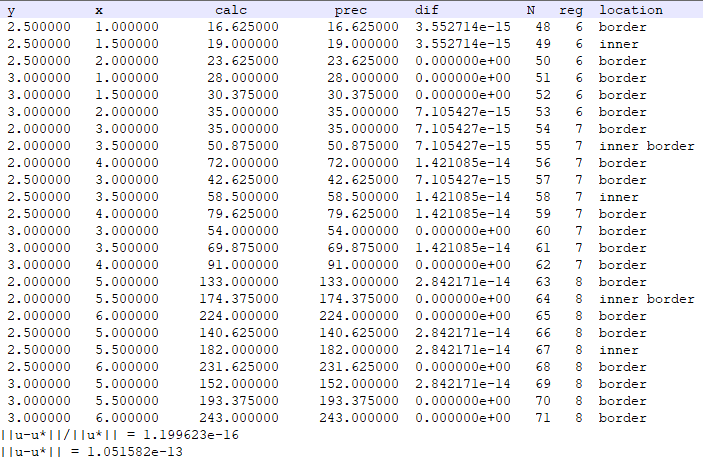


**Область имеет Ш-образную форму**

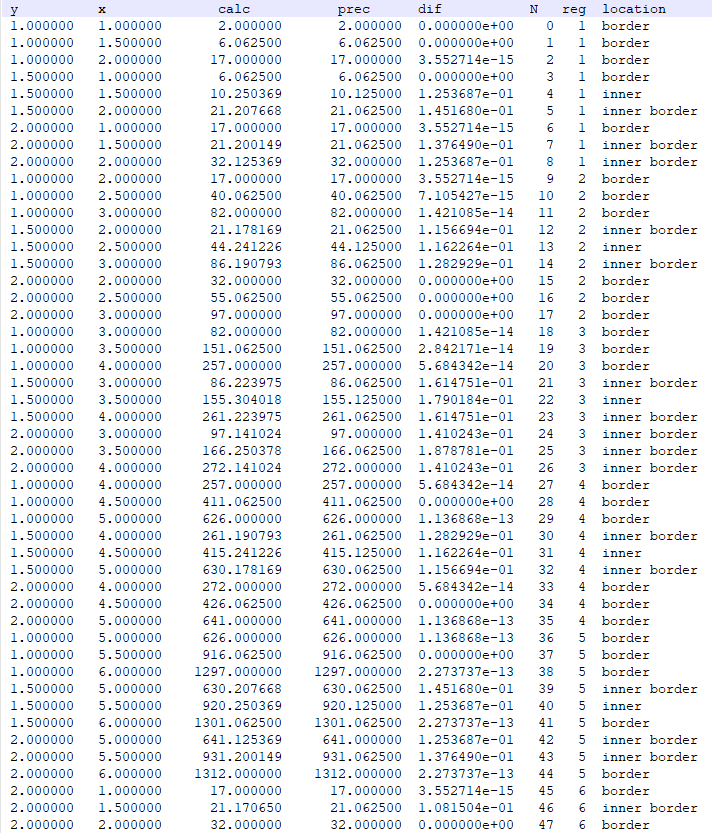
|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  5  6  7  8  4  5  6 | ***Файл regions.txt***  8  -1-  1 2 1 2  1 2 1 2  1 -2 1 -6  -2-  2 3 1 2  1 2 1 2  -1 -3 1 1  -3-  3 4 1 2  1 2 1 2  -2 -4 1 -7  -4-  4 5 1 2  1 2 1 2  -3 -5 1 1  -5-  5 6 1 2  1 2 1 2  -4 1 1 -8  -6-  1 2 2 3  1 2 1 2  1 1 -1 1  -7-  3 4 2 3  1 2 1 2  1 1 -3 1  -8-  5 6 2 3  1 2 1 2  1 1 -5 1 |

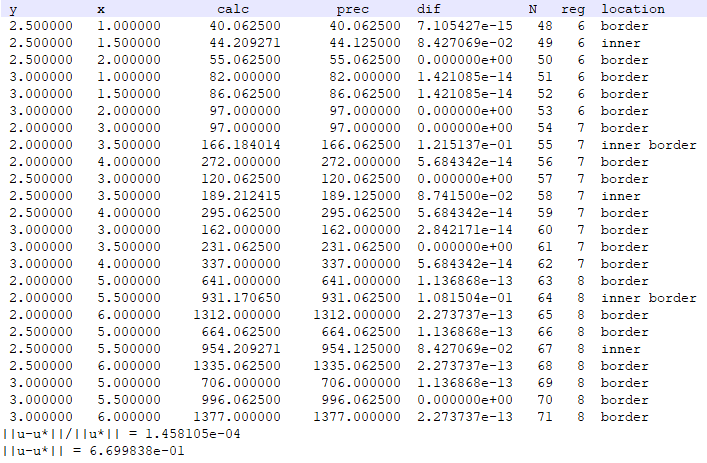
* λ = 1, γ = 1,





* λ = 1, γ = 1,

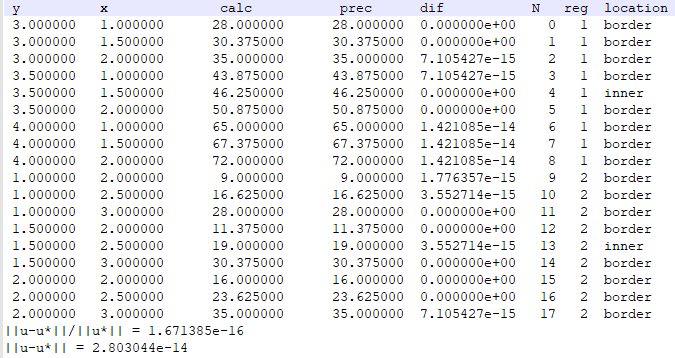




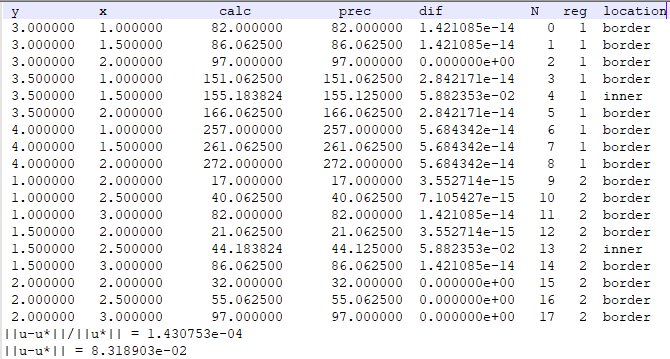
**Область имеет разрывную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  4 | ***Файл regions.txt***  2  -1-  1 2 3 4  1 2 1 2  1 1 1 1  -2-  2 3 1 2  1 2 1 2  1 1 1 1 |

* λ = 1, γ = 1,



* λ = 1, γ = 1,

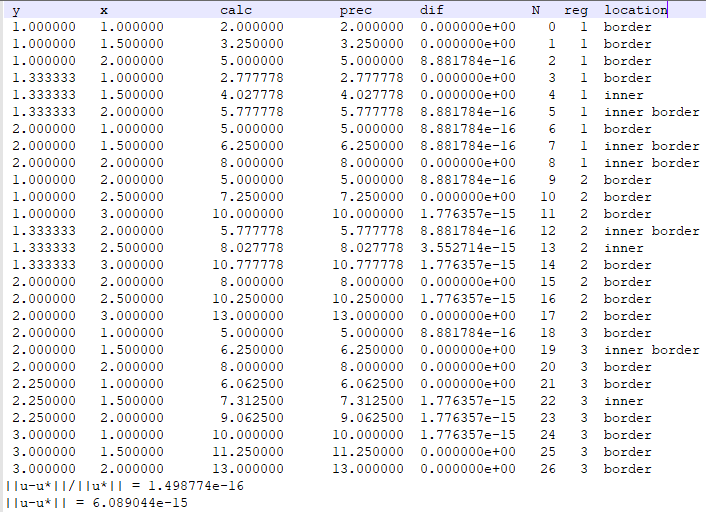


1. **Тестирование на неравномерной сетке**

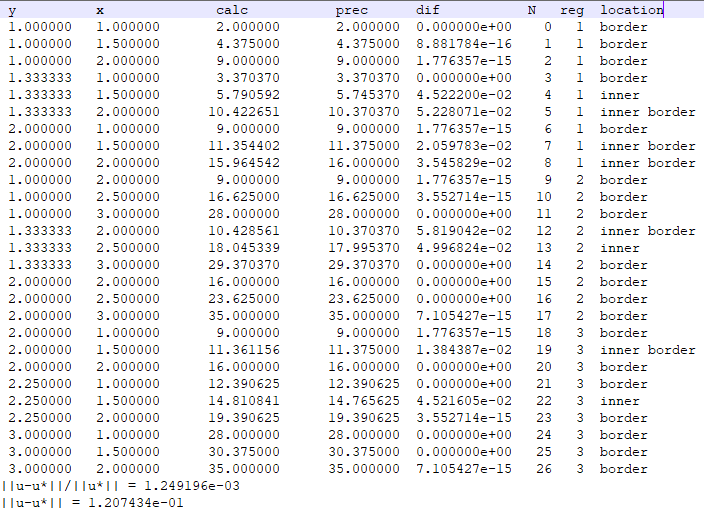
**Область имеет L-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3 | ***Файл regions.txt***  3  -1-  1 2 1 2  1 2 2 2  1 -2 1 -3  -2-  2 3 1 2  1 2 2 2  -1 1 1 1  -3-  1 2 2 3  1 2 3 2  1 1 -1 1 |

* λ = 1, γ = 1,



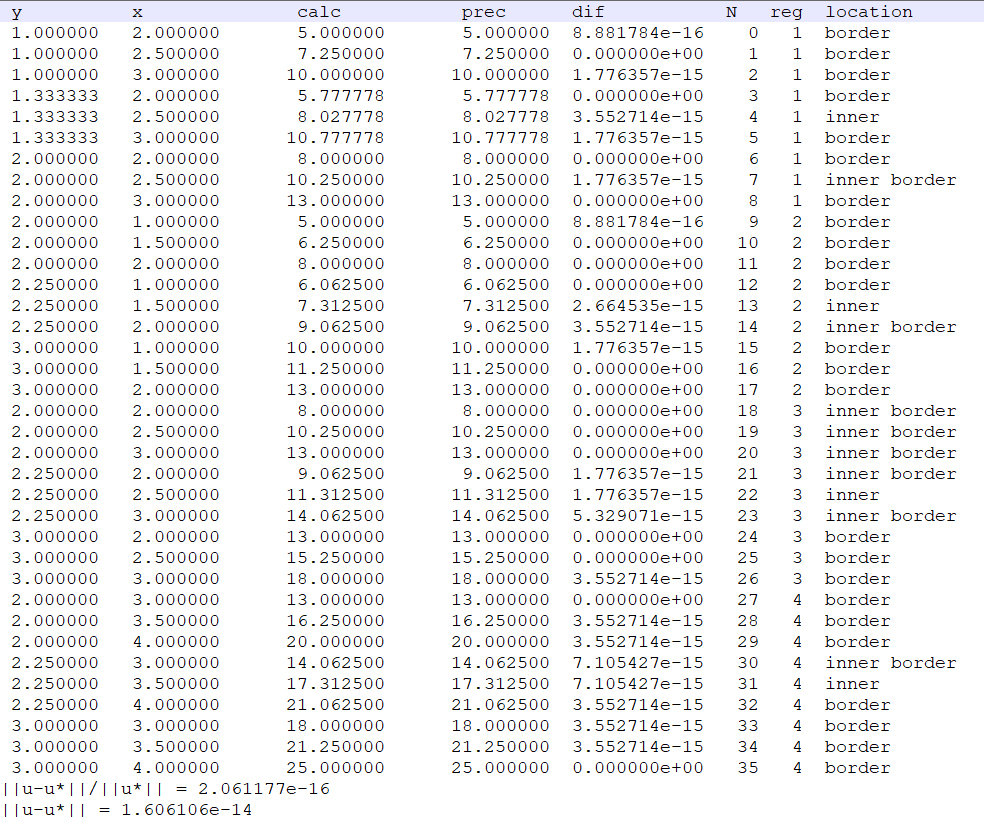
* λ = 1, γ = 1,



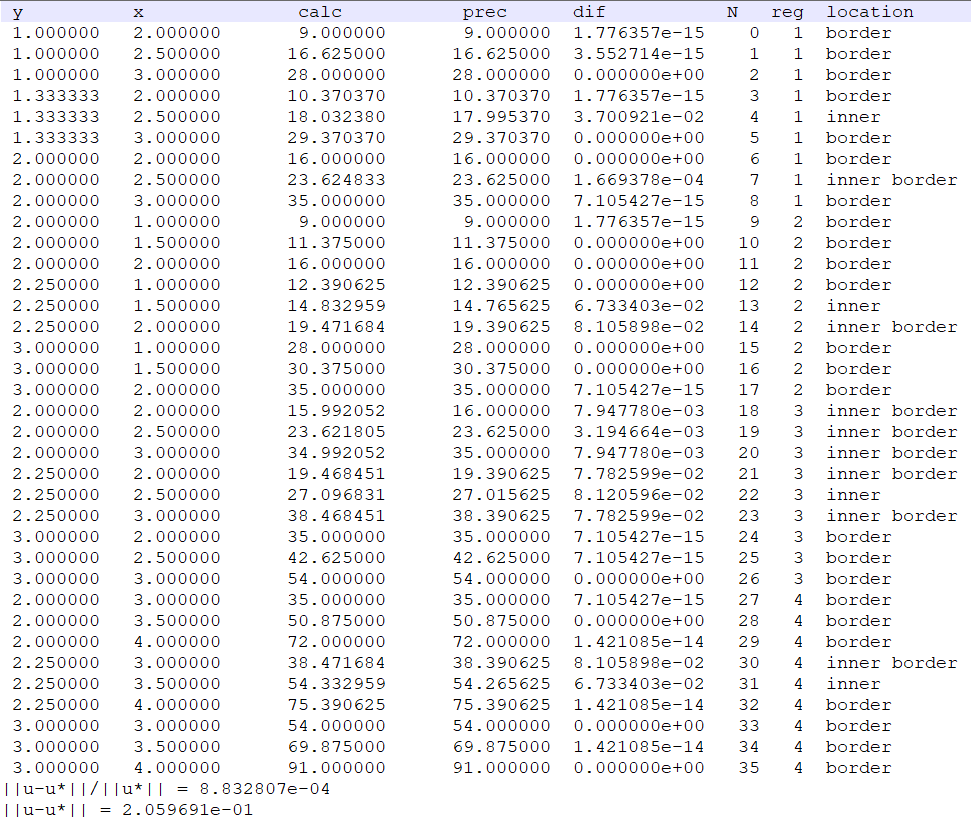
**Область имеет Т-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  4 | ***Файл regions.txt***  4  -1-  2 3 1 2  1 2 2 2  1 1 1 -3  -2-  1 2 2 3  1 2 3 2  1 -3 1 1  -3-  2 3 2 3  1 2 3 2  -2 -4 -1 1  -4-  3 4 2 3  1 2 3 2  -3 1 1 1 |

* λ = 1, γ = 1,



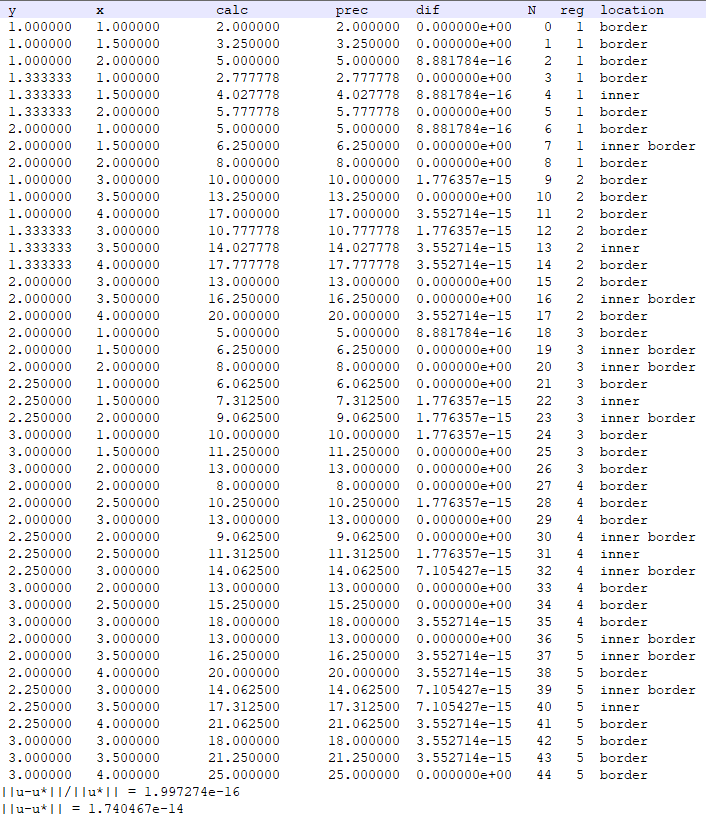
* λ = 1, γ = 1,



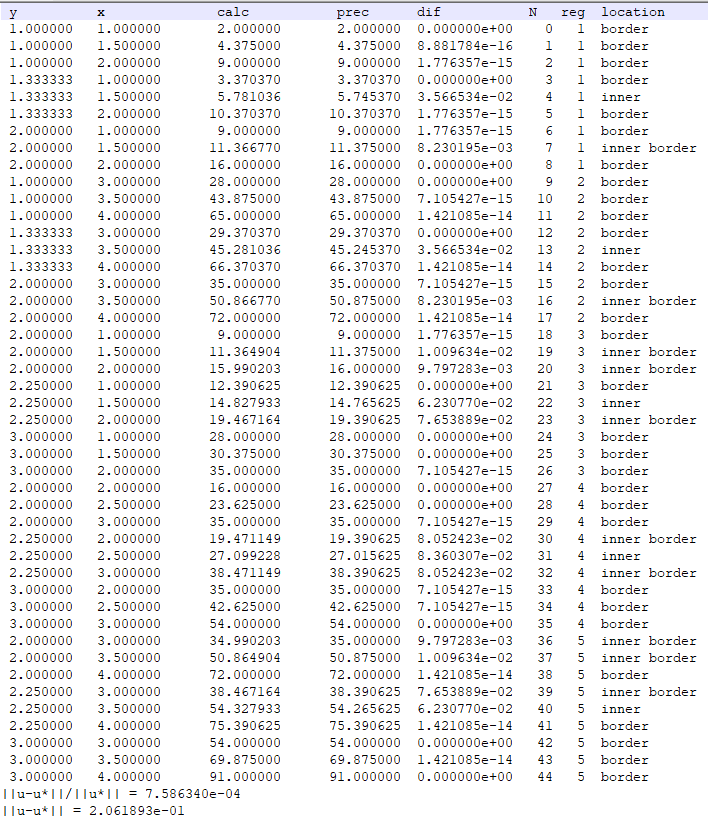
**Область имеет П-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  4  5 | ***Файл regions.txt***  5  -1-  1 2 1 2  1 2 2 2  1 1 1 -3  -2-  3 4 1 2  1 2 2 2  1 1 1 -5  -3-  1 2 2 3  1 2 3 2  1 -4 -1 1  -4-  2 3 2 3  1 2 3 2  -3 -5 1 1  -5-  3 4 2 3  1 2 3 2  -4 1 -2 1 |

* λ = 1, γ = 1,



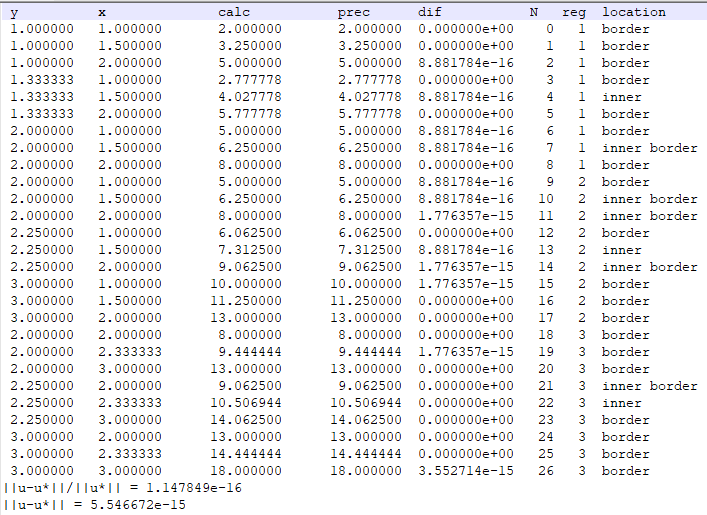
* λ = 1, γ = 1,



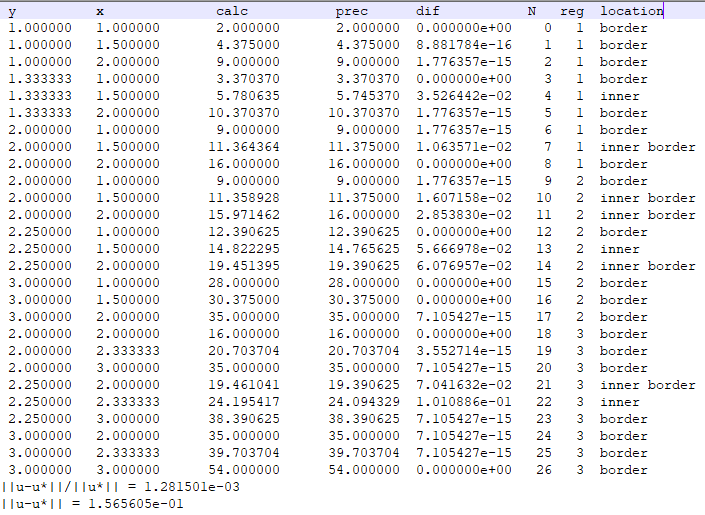
**Область имеет Г-образную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3 | ***Файл regions.txt***  3  -1-  1 2 1 2  1 2 2 2  1 1 1 -2  -2-  1 2 2 3  1 2 3 2  1 -3 -1 1  -3-  2 3 2 3  2 2 3 2  -2 1 1 1 |

* λ = 1, γ = 1,



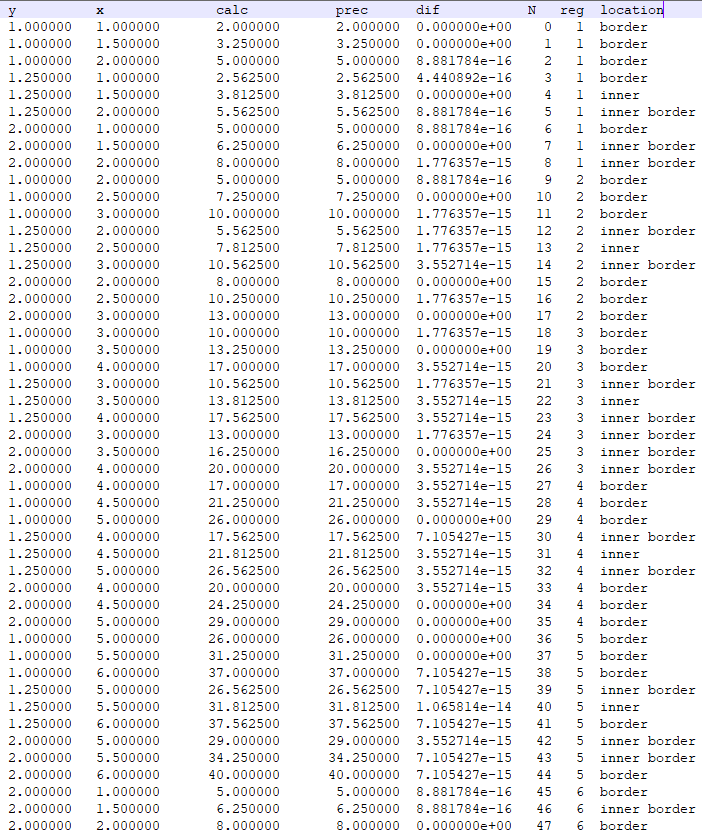
* λ = 1, γ = 1,

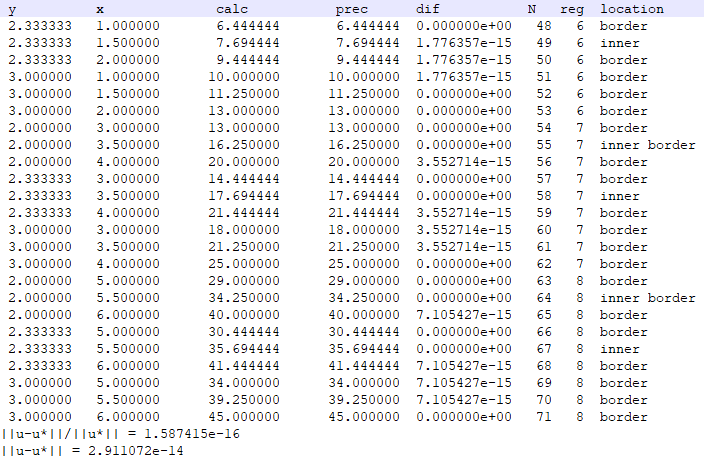


**Область имеет Ш-образную форму**

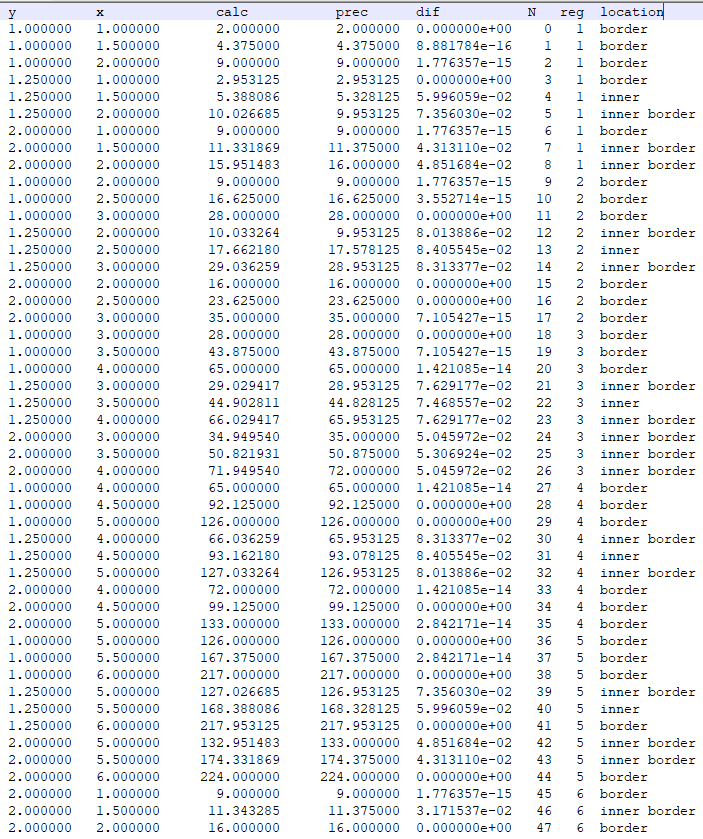
|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  3  4  5  6  7  8  4  5  6 | ***Файл regions.txt***  8  -1-  1 2 1 2  1 2 3 2  1 -2 1 -6  -2-  2 3 1 2  1 2 3 2  -1 -3 1 1  -3-  3 4 1 2  1 2 3 2  -2 -4 1 -7  -4-  4 5 1 2  1 2 3 2  -3 -5 1 1  -5-  5 6 1 2  1 2 3 2  -4 1 1 -8  -6-  1 2 2 3  1 2 2 2  1 1 -1 1  -7-  3 4 2 3  1 2 2 2  1 1 -3 1  -8-  5 6 2 3  1 2 2 2  1 1 -5 1 |

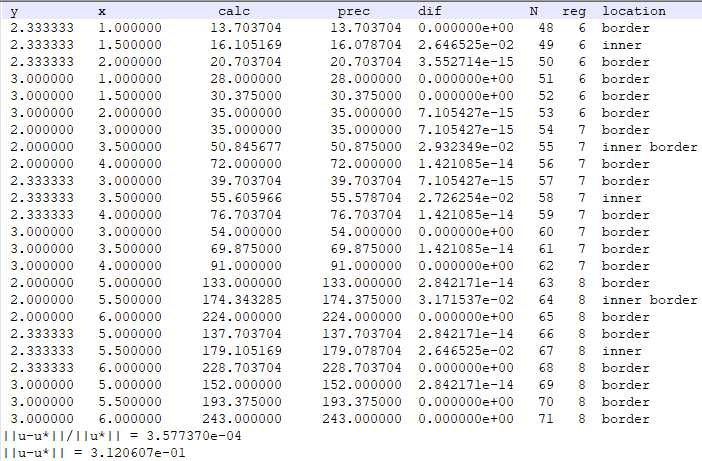
* λ = 1, γ = 1,





* λ = 1, γ = 1,

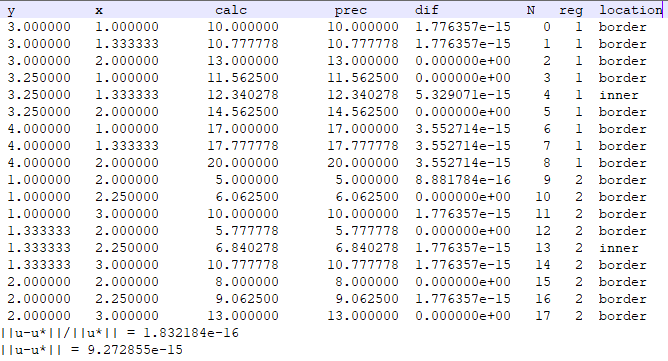




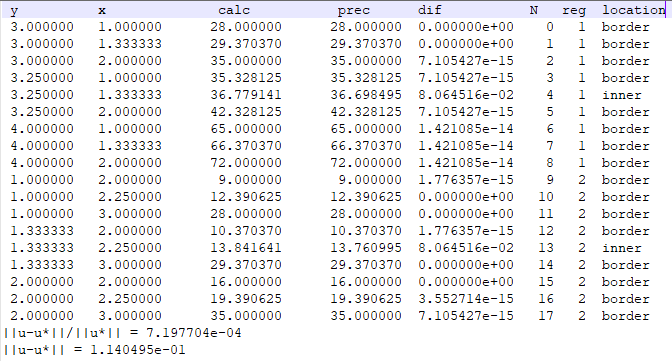
**Область имеет разрывную форму**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3  1  2  4 | ***Файл regions.txt***  2  -1-  1 2 3 4  2 2 3 2  1 1 1 1  -2-  2 3 1 2  3 2 2 2  1 1 1 1 |

* λ = 1, γ = 1,



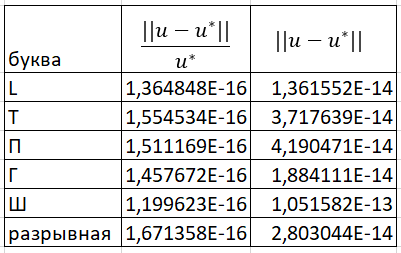
* λ = 1, γ = 1,



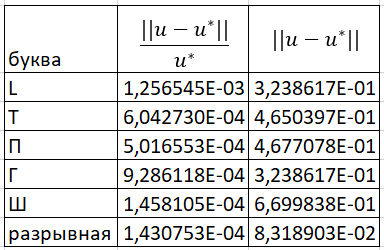
1. **Выводы**

*Результаты проверки работоспособности программы для расчетных областей различной формы на равномерной сетке:*

* λ = 1, γ = 1,



* λ = 1, γ = 1,

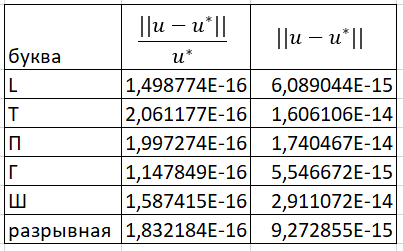


Равномерная сетка имеет третий порядок аппроксимации, поэтому при наблюдается появление погрешности вычислений.

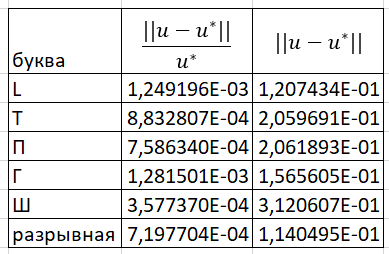
**При тестировании для расчетных областей различной формы на равномерной сетке мы получили примерно одинаковую погрешность решения.**

*Результаты проверки работоспособности программы для расчетных областей различной формы на неравномерной сетке:*

* λ = 1, γ = 1,



* λ = 1, γ = 1,



Равномерная сетка имеет второй порядок аппроксимации, поэтому при наблюдается появление погрешности вычислений.

**При тестировании для расчетных областей различной формы на неравномерной сетке мы получили примерно одинаковую погрешность решения.**