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

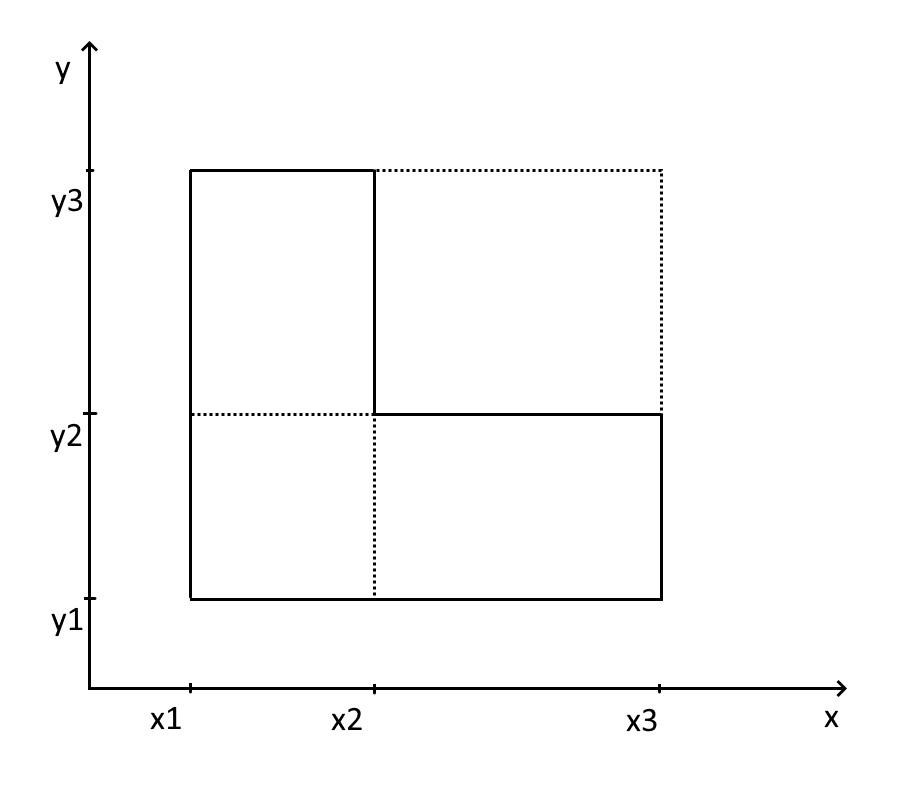
2021

1. **Цель работы**

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

1. **Задание**

Область имеет L-образную 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 краевыми условиями

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 .

**Учет вторых краевых условий:**

так как расчетная область представляет собой многоугольник, со сторонами, параллельными координатным осям, то направление нормали к границам, на которых заданы вторые краевые условия совпадает с одной из координатных линий, методы аппроксимации производной по нормали можно свести следующей формуле:

1. **Текст программы**

***Файл “vector.h”***

#pragma once

#include <vector>

#include <iomanip>

#include <fstream>

using namespace std;

// Умножение вектора на число

vector<double> operator \* (double val, const vector<double>& vec)

{

size\_t n = vec.size();

vector<double> res(n);

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

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

return res;

}

// Сложение векторов

vector<double> operator + (const vector<double>& vec1, const vector<double>& vec2)

{

size\_t n = vec1.size();

if (n != vec2.size())

throw("a.size() != b.size()");

vector<double> res(n);

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

res[i] = vec1[i] + vec2[i];

return res;

}

// Вычитание векторов

vector<double> operator - (const vector<double>& vec1, const vector<double>& vec2)

{

size\_t n = vec1.size();

if (n != vec2.size())

throw("a.size() != b.size()");

vector<double> res(n);

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

res[i] = vec1[i] - vec2[i];

return res;

}

// Скалярное произведение векторов

double operator \*(const vector<double>& vec1, const vector<double>& vec2)

{

size\_t n = vec1.size();

if (n != vec2.size())

throw("vec1.size() != vec2.size()");

double res = 0;

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

res += vec1[i] \* vec2[i];

return res;

}

// Норма вектора

double Norm(const vector<double>& vec)

{

return sqrt(vec \* vec);

}

***Файл “SLAE.h”***

#pragma once

#include <vector>

#include "Vector.h"

using namespace std;

class SLAE

{

public:

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

vector<int> index; // Индексы столбцов

vector<double> f; // Вектор правой части

const int D = 5; // Количество диагоналей матрицы

int N = 0; // Размерность матрицы

int M = 0; // Расстояние до крайних диагоналей

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

SLAE(const int& N, const int& N\_X)

{

M = N\_X - 2;

this->N = N;

index.resize(D);

index[0] = -(2 + M);

index[1] = -1;

index[2] = 0;

index[3] = 1;

index[4] = 2 + M;

matrix.resize(D);

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

matrix[i].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 i = 0; i < D; i++)

{

k = index[i];

if(k < 0)

for(int j = abs(k); j < n; j++)

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

else

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

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

}

}

// Получение относительной невязки системы

double RelativeResidual(vector<double>& vec)

{

vector<double> mult(N);

Multiplication(vec, mult);

mult = f - mult;

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];

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);

//cout << k << " " << residual << endl;

}

}

};

***Файл “EllipticalProblev.h”***

#pragma once

#include <vector>

#include <fstream>

#include <string>

#include "SLAE.h"

#include "Test.h"

using namespace std;

class EllipticalProblem

{

public:

vector<double> x\_reg; // Границы области по X

vector<double> y\_reg; // Границы области по Y

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

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

int N\_X; // Количество узлов по X

int N\_Y; // Количество узлов по Y

int x\_bord; // Индекс внутренней границы

// L-области по X

int y\_bord; // Индекс внутренней границы

// L-области по Y

const int N\_BORD = 6; // Количество ребер

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

SLAE\* slae; // Система

Test test; // Тестовая информация

EllipticalProblem()

{

x\_reg = vector<double>(3);

y\_reg = vector<double>(3);

ReadCoordLines("coords.txt");

double h;

// Генерация координат узлов по X

h = (x\_reg[1] - x\_reg[0]) / x\_bord;

for(int i = 0; i <= x\_bord; i++)

x\_node[i] = x\_reg[0] + h \* i;

h = (x\_reg[2] - x\_reg[1]) / (N\_X - x\_bord - 1);

for(int i = 0; i <= N\_X - x\_bord - 1; i++)

x\_node[i + x\_bord] = x\_reg[1] + h \* i;

// Генерация координат узлов по Y

h = (y\_reg[1] - y\_reg[0]) / y\_bord;

for(int i = 0; i <= y\_bord; i++)

y\_node[i] = y\_reg[0] + h \* i;

h = (y\_reg[2] - y\_reg[1]) / (N\_Y - y\_bord - 1);

for(int i = 0; i <= (N\_Y - y\_bord - 1); i++)

y\_node[i + y\_bord] = y\_reg[1] + h \* i;

ReadBordConditions("borders.txt");

// Формирование индексов границ ребер с соответсвующими

// краевыми условиями

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

{

borders[i][1] = CorrespondX(borders[i][1]);

borders[i][2] = CorrespondX(borders[i][2]);

borders[i][3] = CorrespondY(borders[i][3]);

borders[i][4] = CorrespondY(borders[i][4]);

}

// Инициализация СЛАУ

slae = new SLAE(N\_X \* N\_Y, N\_X);

// Инициализация тестовых данных

test = Test(2);

}

~EllipticalProblem()

{

delete slae;

}

// Функция считывания границ области из файла FILE\_NAME

void ReadCoordLines(const string& FILE\_NAME)

{

ifstream fin(FILE\_NAME);

// Считываем границы области

for(int i = 0; i < 3; i++)

fin >> x\_reg[i];

for(int i = 0; i < 3; i++)

fin >> y\_reg[i];

// Считывание координатных линий по X

int count = 0; // Число узлов по X

int t1, t2;

fin >> t1;

count += t1;

fin >> t2;

count += t2;

N\_X = count + 1;

x\_bord = t1;

x\_node.resize(N\_X);

// Считывание координатных линий по Y

count = 0; // Число узлов по Y

fin >> t1;

count += t1;

fin >> t2;

count += t2;

N\_Y = count + 1;

y\_bord = t1;

y\_node.resize(N\_Y);

fin.close();

}

int CorrespondX(const int& I)

{

switch(I)

{

case(0): return 0;

case(1): return x\_bord;

case(2): return N\_X - 1;

}

}

int CorrespondY(const int& I)

{

switch(I)

{

case(0): return 0;

case(1): return y\_bord;

case(2): return N\_Y - 1;

}

}

// Функция считывания индексов для описания краевых условий

// из файла FILE\_NAME

void ReadBordConditions(const string& FILE\_NAME)

{

ifstream fin(FILE\_NAME);

borders.resize(N\_BORD);

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

{

borders[i].resize(5);

for(int j = 0; j < 5; j++)

fin >> borders[i][j];

}

fin.close();

}

// Формирование матрицы системы

void FormMatrix()

{

for(int n = 0; n < slae->N; n++)

{

// Индексы центрального узла

int x\_cent = n % N\_X;

int y\_cent = floor(n / N\_X);

// Обработка некраевых узлов внутри L-формы

if(x\_cent < N\_X - 1 && x\_cent > 0 &&

y\_cent < y\_bord && y\_cent > 0 ||

x\_cent < x\_bord && x\_cent > 0 &&

y\_cent < N\_Y - 1 && y\_cent > 0)

{

// Приросты по X

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

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

// Приросты по Y

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

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

// Нижний узел

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

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

// Левый узел

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

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

// Центральный узел

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

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

// Правый узел

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

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

// Верхний узел

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

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

// Вектор правой части

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

}

// Обработка краевого узла

else if(x\_cent <= x\_bord || y\_cent <= y\_bord)

{

// Обход по всем ребрам

for(int b = 0; b < N\_BORD; b++)

{

// Условие узла нахождения на ребре b

if(x\_cent >= borders[b][1] && x\_cent <= borders[b][2] &&

y\_cent >= borders[b][3] && y\_cent <= borders[b][4])

{

// Первое краевое условие

if(borders[b][0] == 0)

{

slae->matrix[2][n] = 1.0;

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

}

// Второе краевое условие

else if(borders[b][0] == 1)

{

// Если ребро параллельно оси X

if(borders[b][3] == borders[b][4])

{

double h = x\_node[x\_cent + 1] - x\_node[x\_cent - 1];

slae->matrix[1][n] = -test.lambda() / h;

slae->matrix[3][n] = +test.lambda() / h;

// Если нормаль направлена вниз

if(y\_cent == 0)

{

slae->matrix[1][n] \*= -1;

slae->matrix[3][n] \*= -1;

}

}

// Если ребро параллельно оси Y

else if(borders[b][1] == borders[b][2])

{

double h = y\_node[y\_cent + 1] - y\_node[y\_cent - 1];

slae->matrix[0][n] = -test.lambda() / h;

slae->matrix[4][n] = +test.lambda() / h;

// Если нормаль направлена влево

if(x\_cent == 0)

{

slae->matrix[0][n] \*= -1;

slae->matrix[4][n] \*= -1;

}

}

slae->matrix[2][n] = test.gamma();

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

}

break;

}

}

}

// Обработка квадрата за пределами L-формы

else

slae->matrix[2][n] = 1.0;

}

}

// Вывод решения

void PrintSolution()

{

int w = ceil(log10(N\_X \* N\_Y)) + 2;

cout << " x y calc prec dif ";

for(int i = 0; i < w - 1; i++)

cout << " ";

cout << "N location" << endl << fixed;

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

{

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

{

int n = j \* N\_X + i;

//if (i % 2 == 0 && j % 2 == 0)

{

cout << setw(9) << y\_node[j];

cout << setw(11) << x\_node[i];

double t = slae->xk[n];

cout << setw(15) << t;

double tt = 0;

if (i <= x\_bord || j <= y\_bord) tt = test.u(x\_node[i], y\_node[j]);

cout << setw(15) << tt;

cout << setw(14) << scientific <<

abs(t - tt);

cout << fixed << setw(w) << n;

if (i < N\_X - 1 && i > 0 &&

j < y\_bord && j > 0 ||

i < x\_bord && i > 0 &&

j < N\_Y - 1 && j > 0)

cout << " inner";

else if (i <= x\_bord || j <= y\_bord)

cout << " border";

else

cout << " outer";

cout << endl;

}

}

}

}

};

***Файл “Test.h”***

#pragma once

#include <vector>

using namespace std;

class Test

{

public:

// Функция - полином какого порядка

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();

case(5): return (X \* X + Y \* Y) \* sin(X \* Y) \* lambda() + u(X, Y) \* gamma();

};

}

double lambda()

{

return 1;

}

double gamma()

{

return 1;

}

double theta()

{

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;

case(5): return sin(X \* Y);

};

}

};

***Файл “Main.cpp”***

#include "EllipticalProblem.h"

using namespace std;

void main()

{

EllipticalProblem ep = EllipticalProblem();

ep.FormMatrix();

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

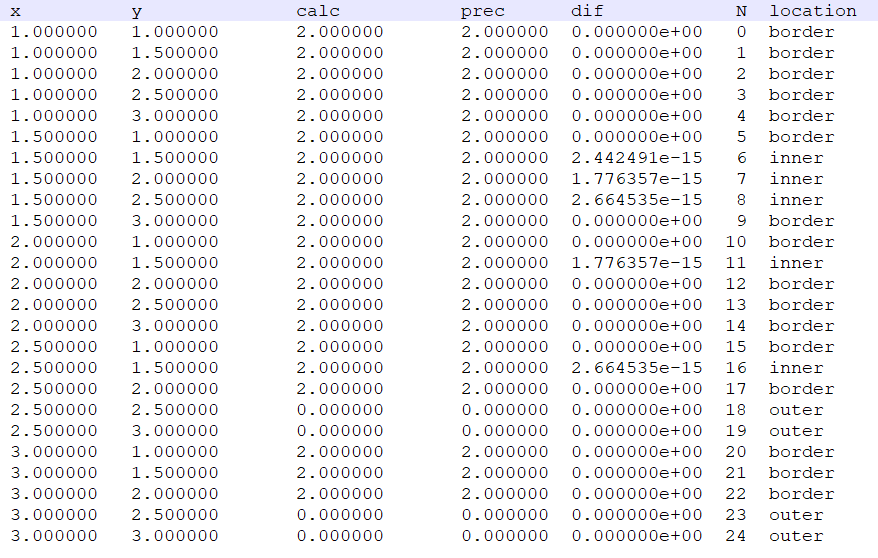
ep.PrintSolution();

}

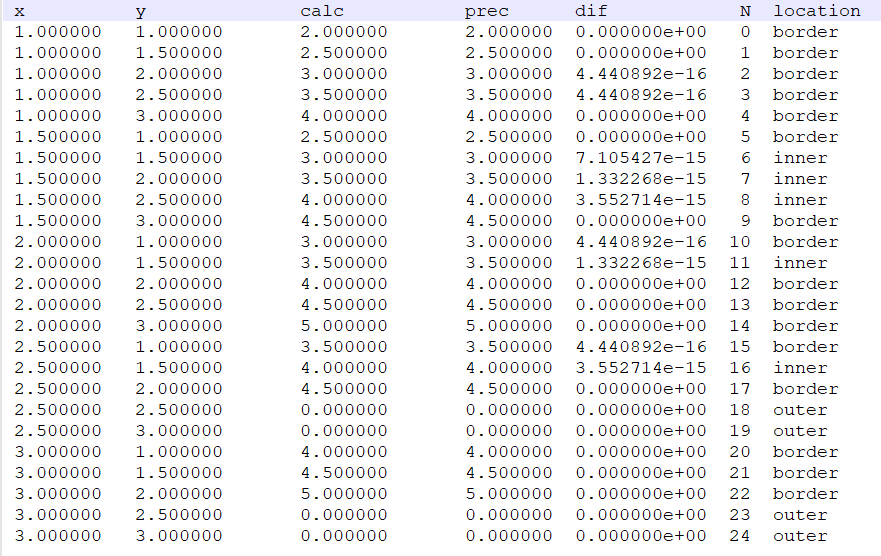
1. **Исследование порядка аппроксимации на равномерной сетке**

|  |  |
| --- | --- |
| 0  1  1  1.5  1.5  2  2  2.5  2.5  3  3 | ***Файл cords.txt***  1 2 3  1 2 3  2 2 2 2  ***Файл borders.txt***  0 0 2 0 0  0 2 2 0 1  0 1 2 1 1  0 1 1 1 2  0 0 1 2 2  0 0 0 0 2 |

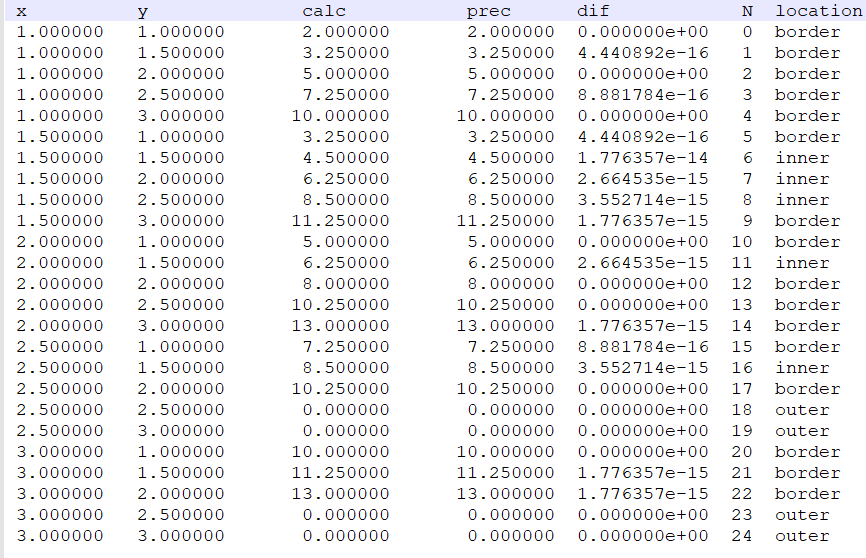
* λ = 1, γ = 1,



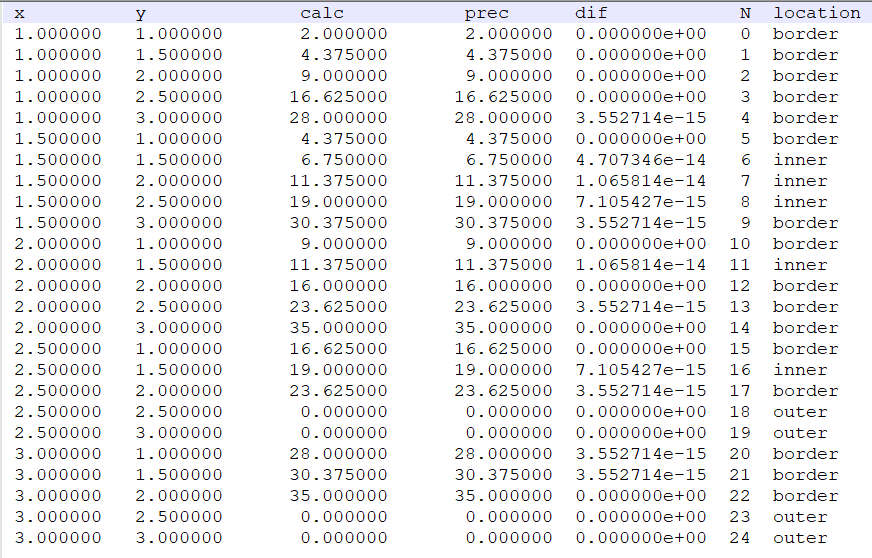
* λ = 1, γ = 1,



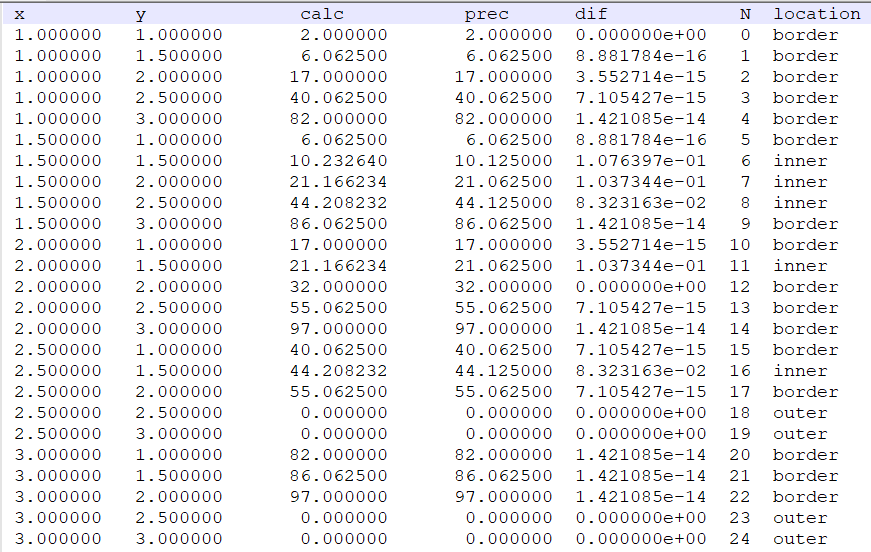
* λ = 1, γ = 1,



* λ = 1, γ = 1,



* λ = 1, γ = 1,

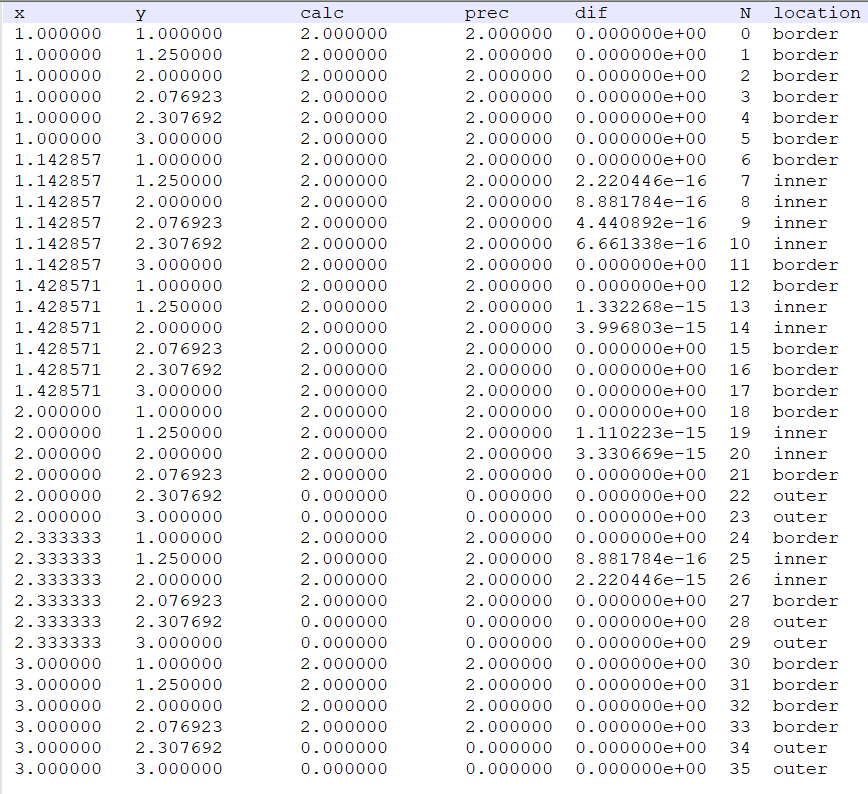


При увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

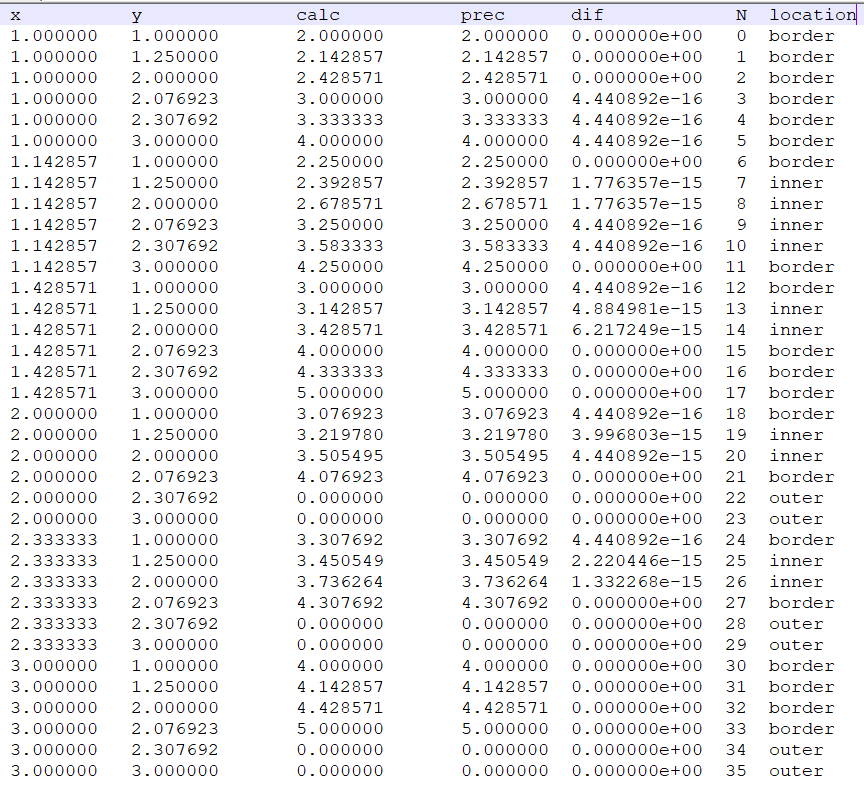
1. **Исследование порядка аппроксимации на неравномерной сетке**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3 | ***Файл cords.txt***  1 2 3  1 2 3  2 3 2 2  3 2 3 3  ***Файл borders.txt***  0 0 2 0 0  0 2 2 0 1  0 1 2 1 1  0 1 1 1 2  0 0 1 2 2  0 0 0 0 2 |

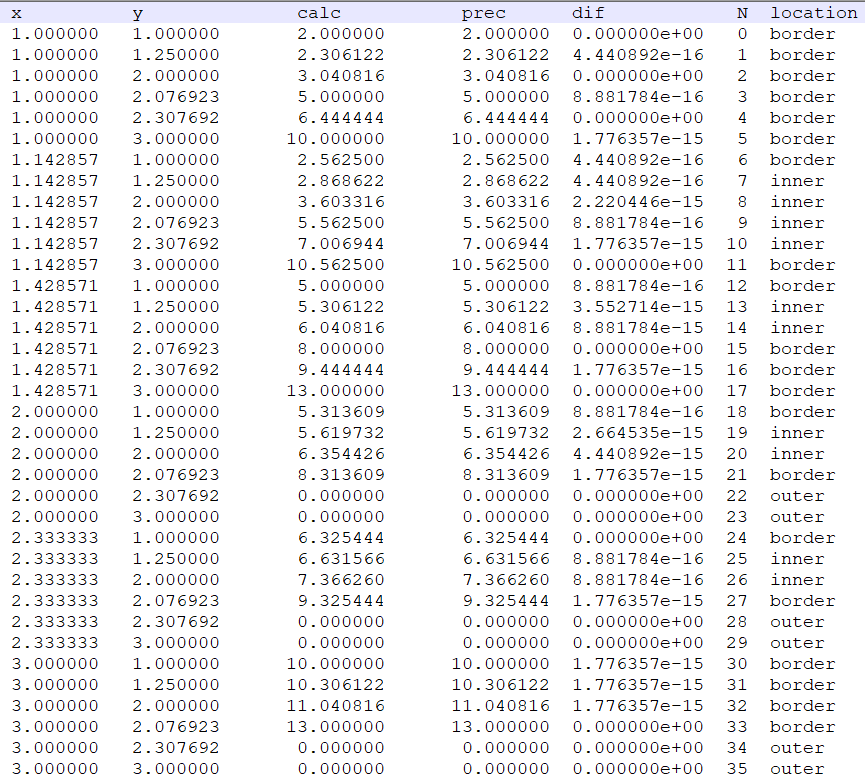
* λ = 1, γ = 1,



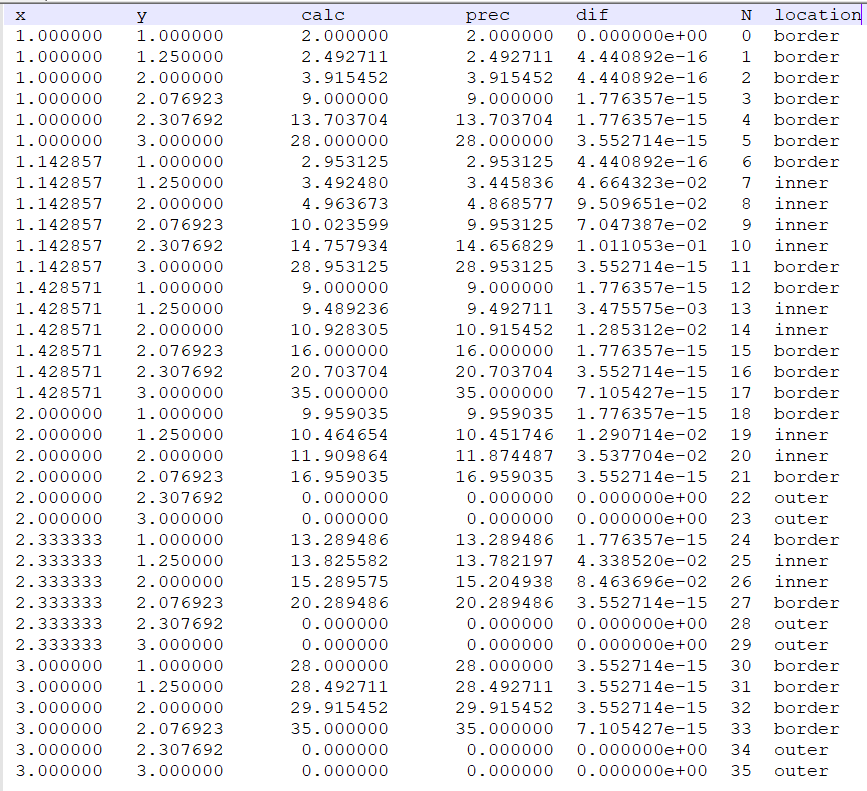
* λ = 1, γ = 1,



* λ = 1, γ = 1,



* λ = 1, γ = 1,



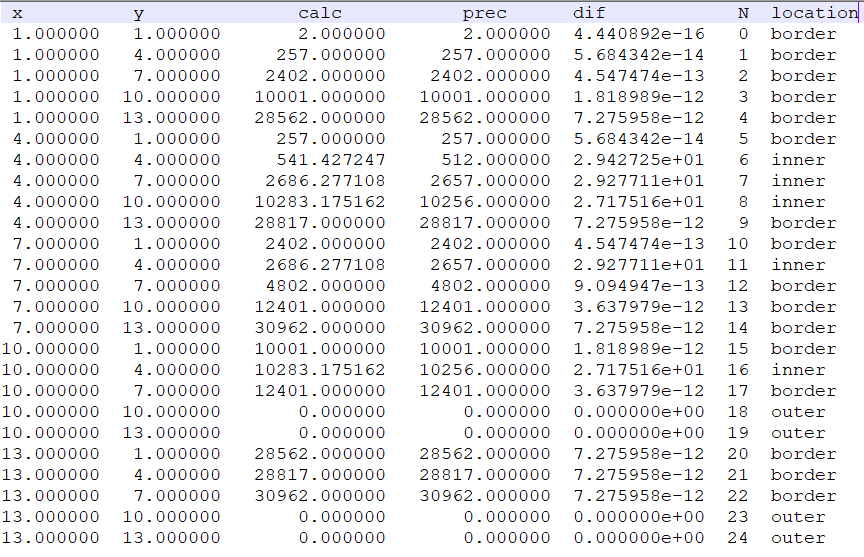
При увеличении степени искомой функции, начиная с , происходит увеличение погрешности.

1. **Исследование порядка сходимости на равномерной сетке**

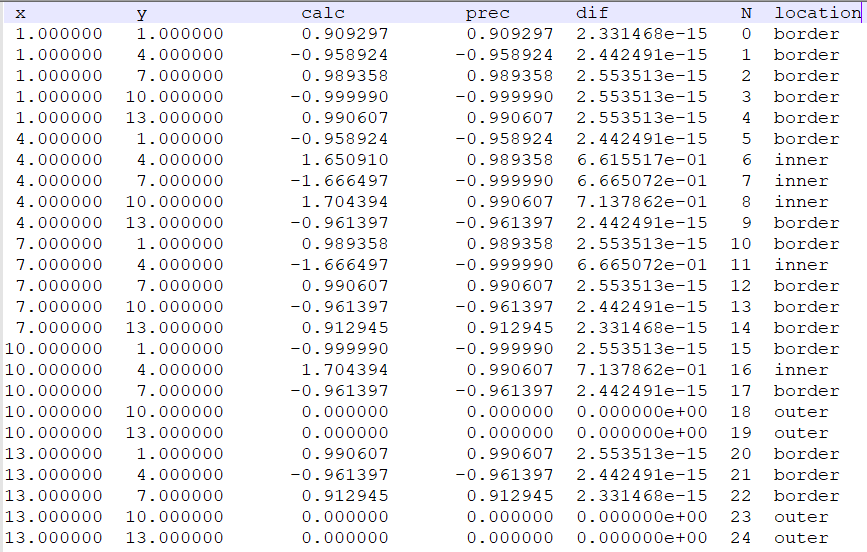
|  |  |
| --- | --- |
| 0  1  1  4  4  7  7  10  11  13  15 | ***Файл cords.txt***  1 7 13  1 7 13  1 2 1 2  1 2 1 2  ***Файл borders.txt***  0 0 2 0 0  0 2 2 0 1  0 1 2 1 1  0 1 1 1 2  0 0 1 2 2  0 0 0 0 2 |

**Будем использовать данную сетку для следующих двух тестов.**

* λ = 1, γ = 1,

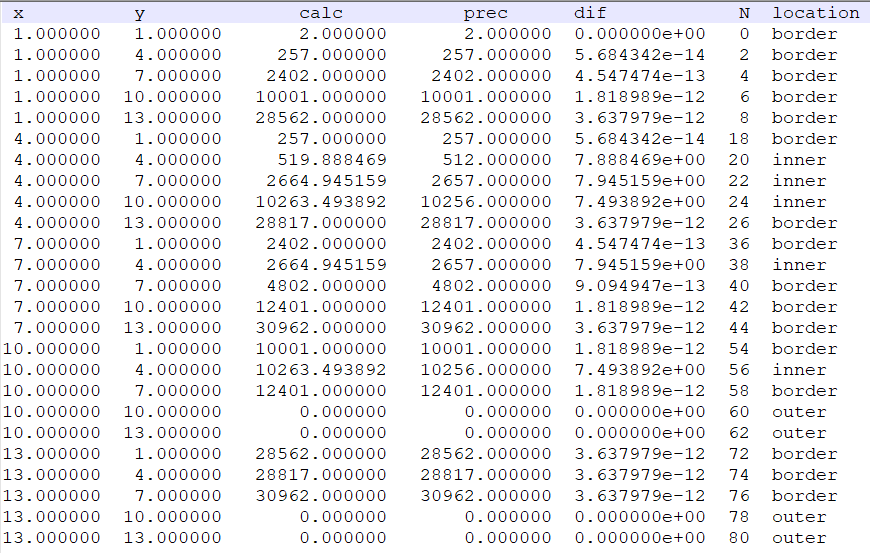


* λ = 1, γ = 1,

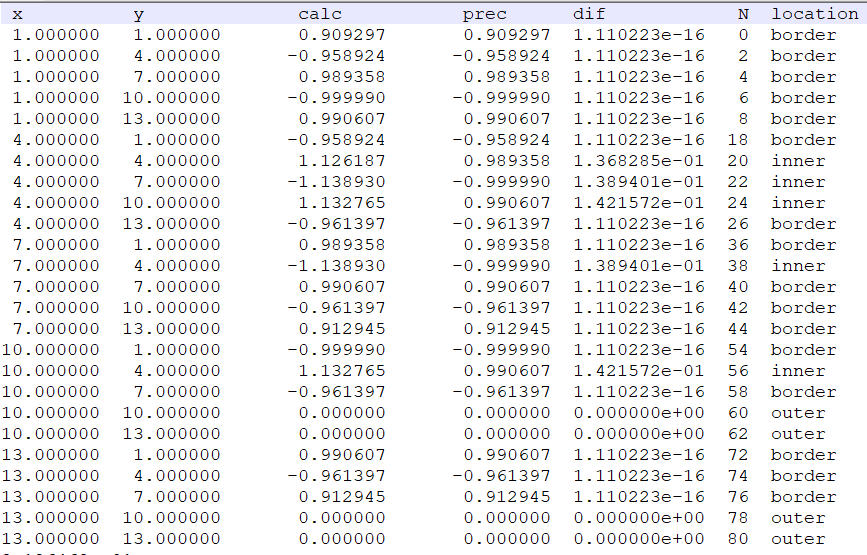


**Поделим сетку в два раза по x и y**

* λ = 1, γ = 1,

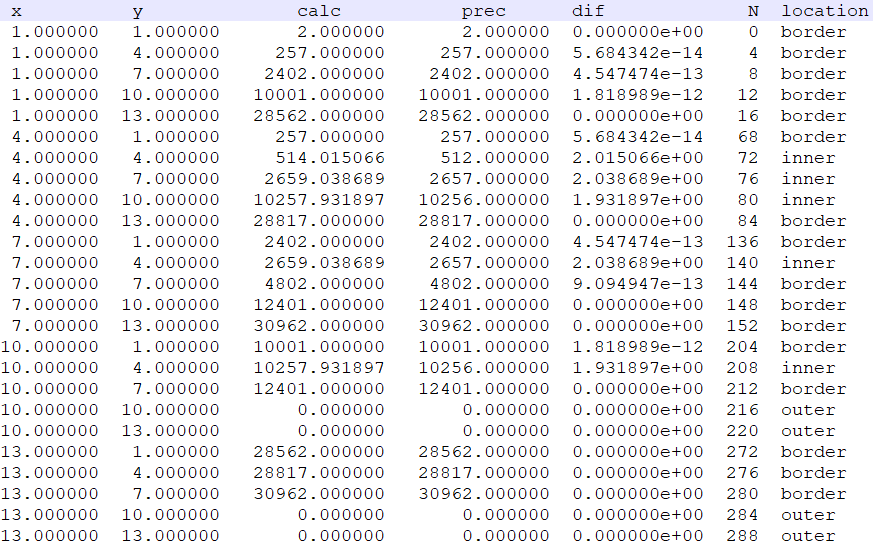


* λ = 1, γ = 1,

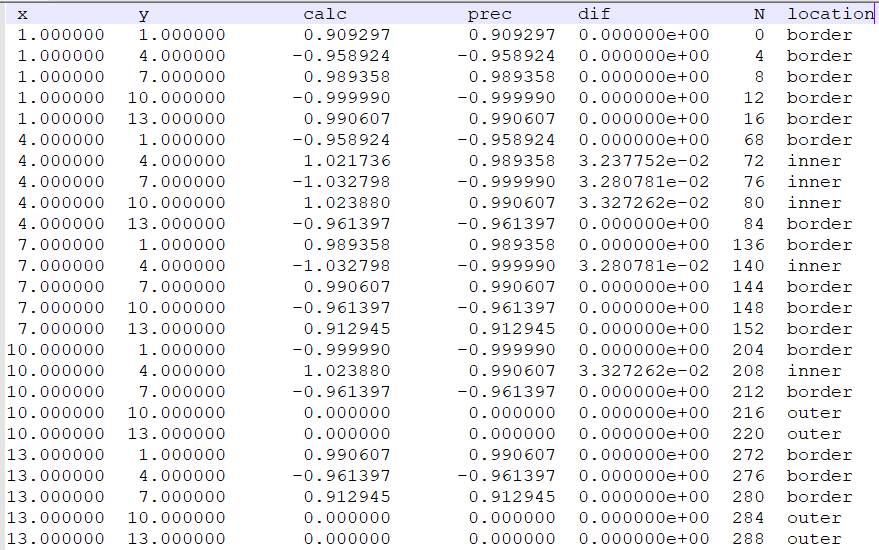


**Поделим сетку еще в два раза по x и y**

* λ = 1, γ = 1,

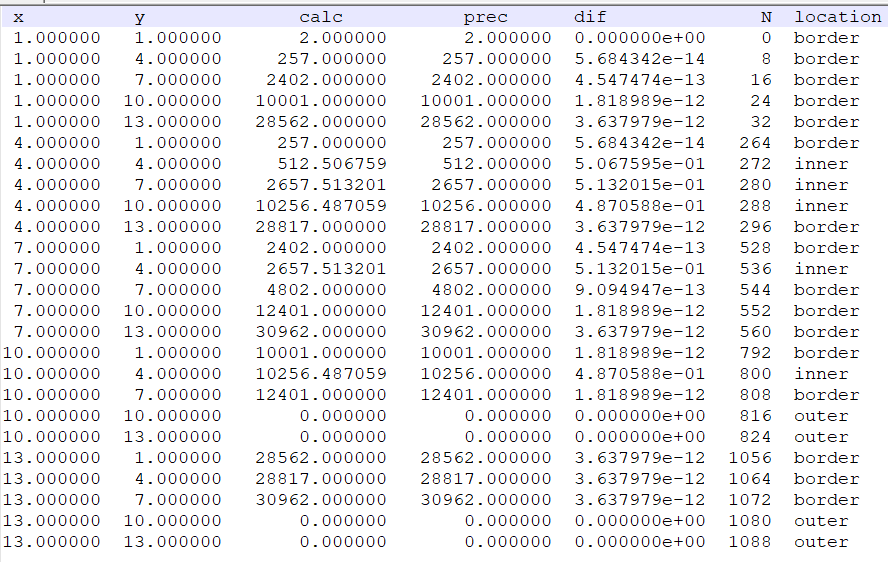


* λ = 1, γ = 1,

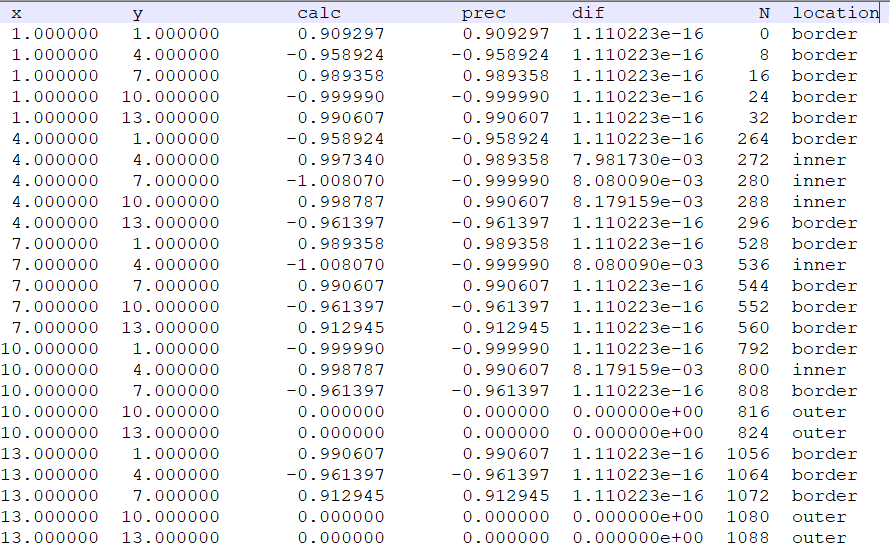


**Поделим сетку еще в два раза по x и y**

* λ = 1, γ = 1,

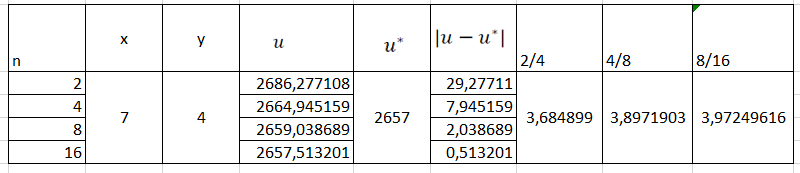


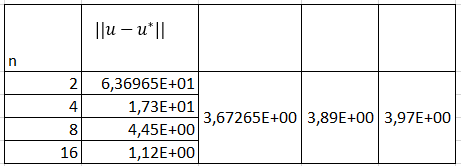
* λ = 1, γ = 1,



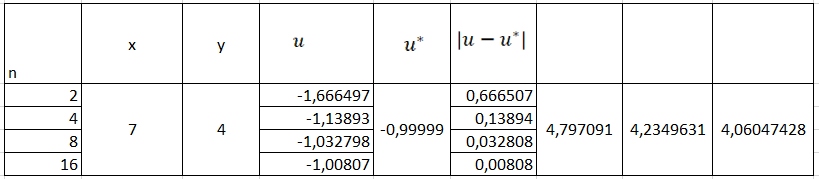
**Рассмотрим значения численного и аналитического решения**

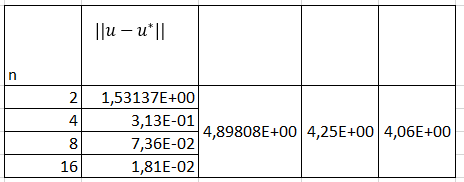
* λ = 1, γ = 1,





* λ = 1, γ = 1,

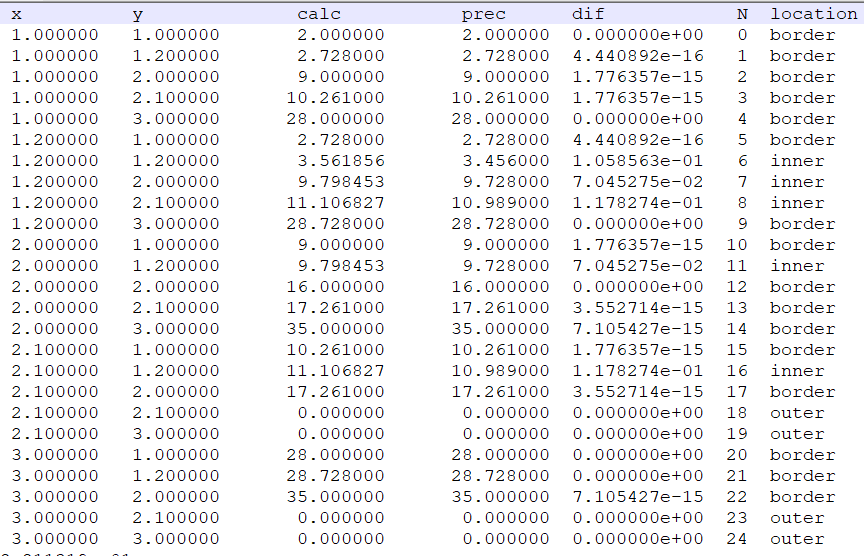




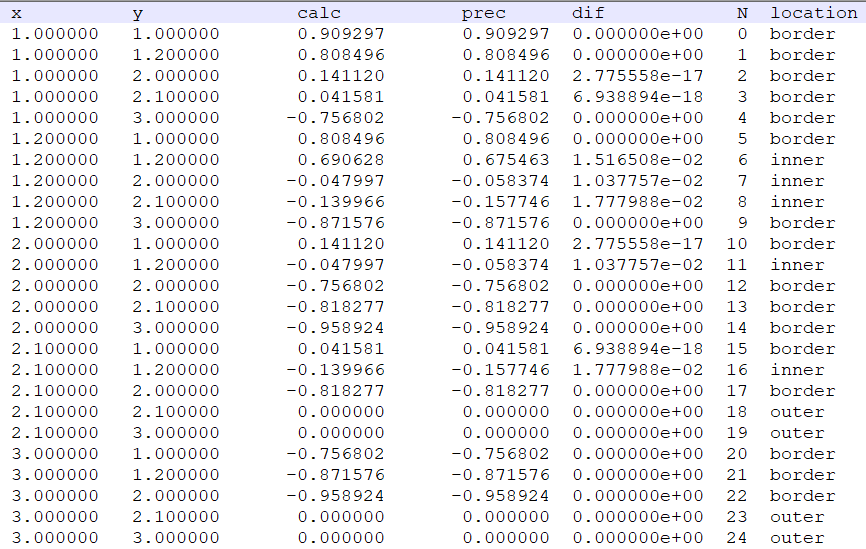
1. **Исследование порядка сходимости на неравномерной сетке**

|  |  |
| --- | --- |
| 0  1  1  2  2  3  3 | ***Файл cords.txt***  1 2 3  1 2 3  4 2 9 2  4 2 9 2  ***Файл borders.txt***  0 0 2 0 0  0 2 2 0 1  0 1 2 1 1  0 1 1 1 2  0 0 1 2 2  0 0 0 0 2 |

* λ = 1, γ = 1,

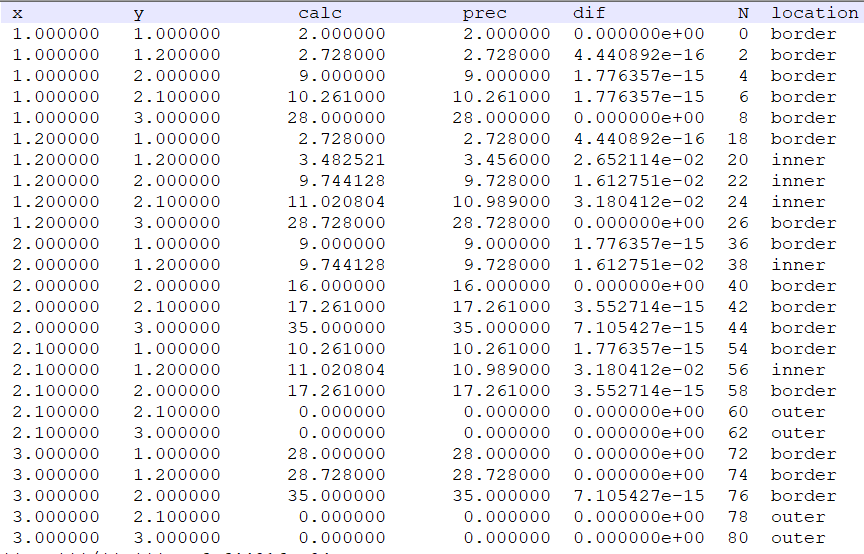


* λ = 1, γ = 1,

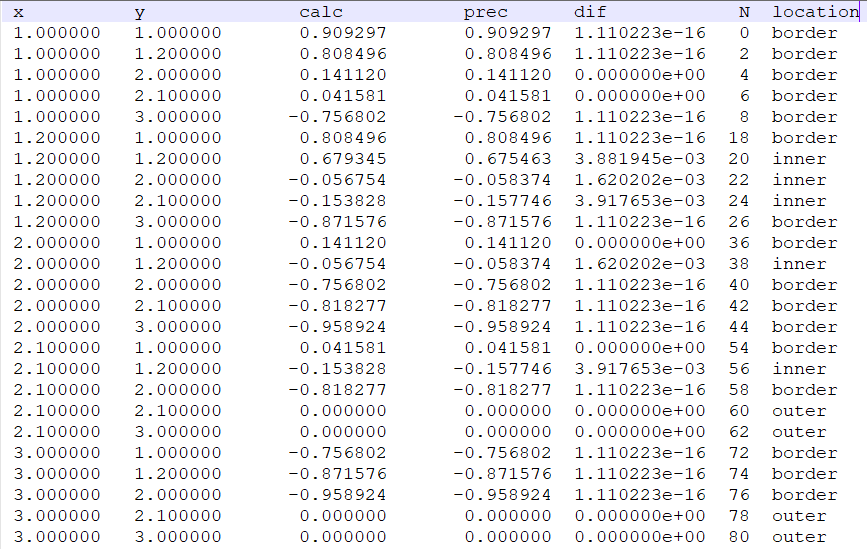


**Поделим сетку в два раза по x и y ().**

* λ = 1, γ = 1,

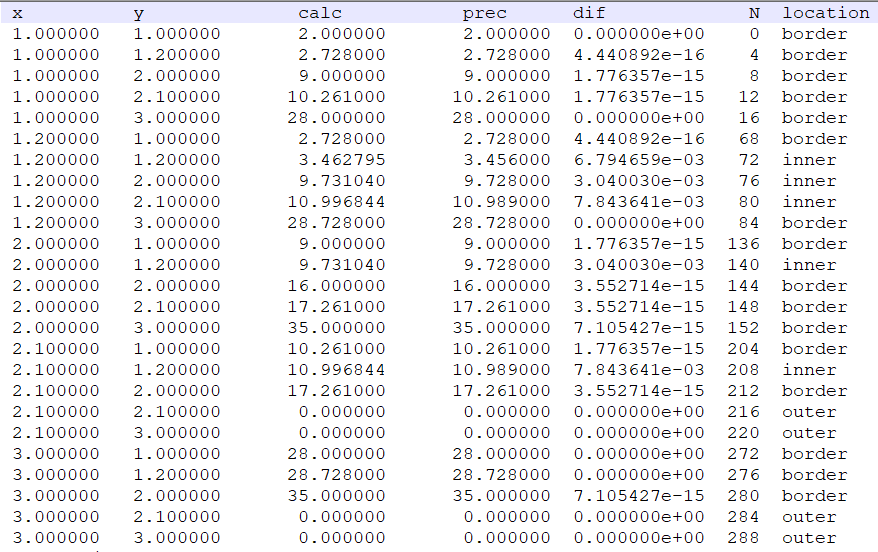


* λ = 1, γ = 1,

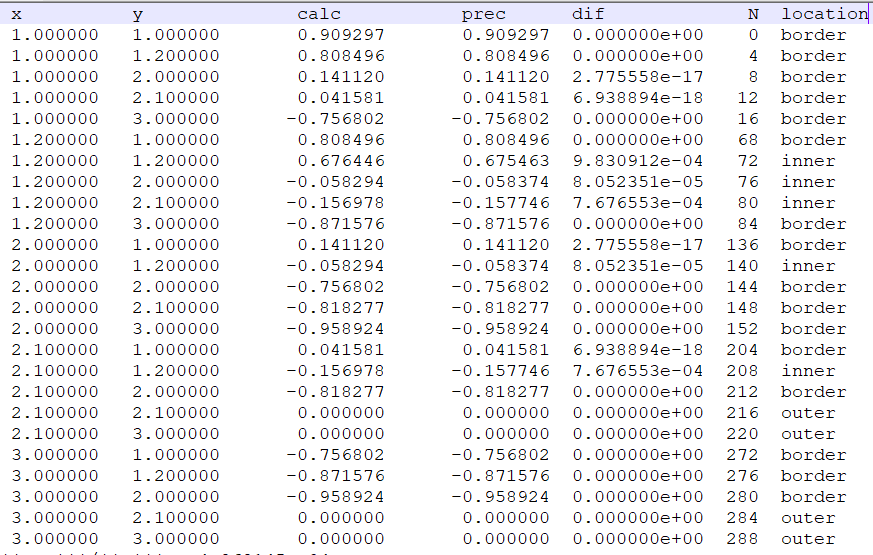


**Поделим сетку в два раза по x и y ().**

* λ = 1, γ = 1,

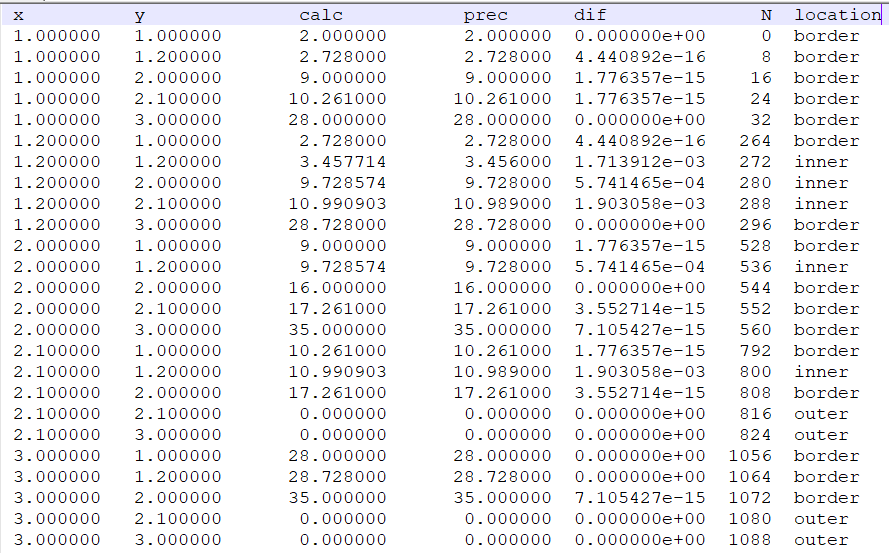


* λ = 1, γ = 1,

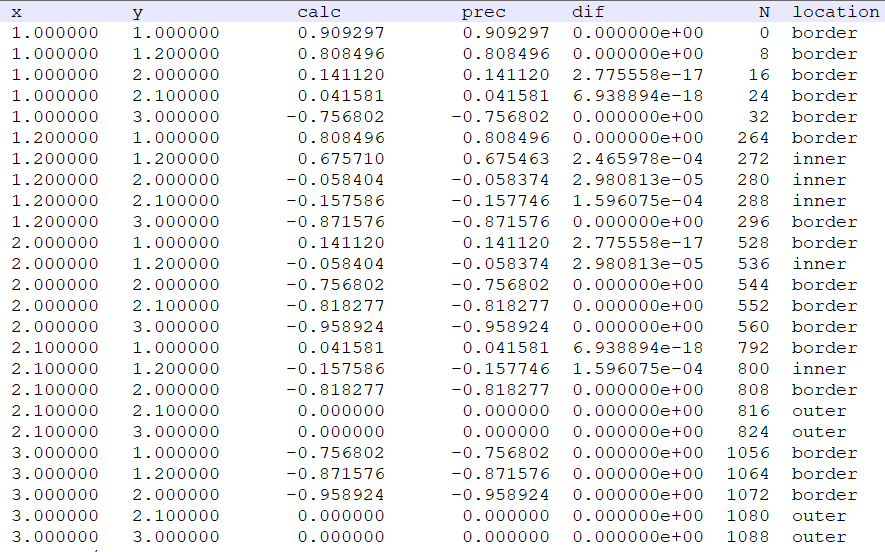


**Поделим сетку в два раза по x и y ().**

* λ = 1, γ = 1,

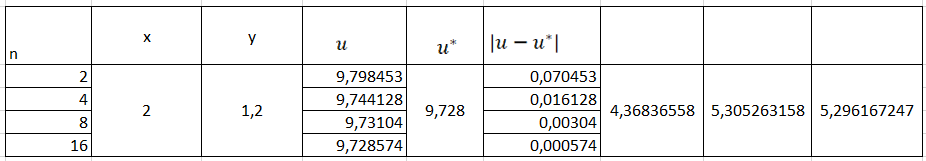


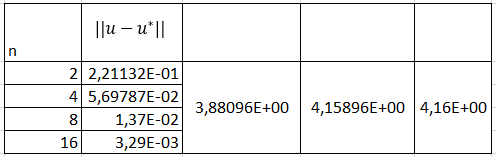
* λ = 1, γ = 1,



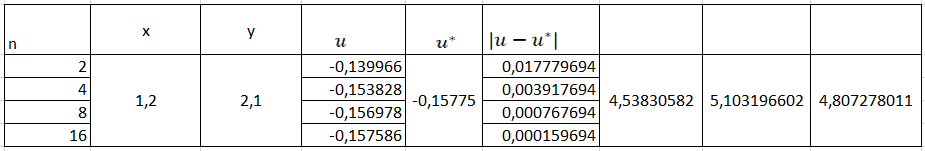
**Рассмотрим значения численного и аналитического решения**

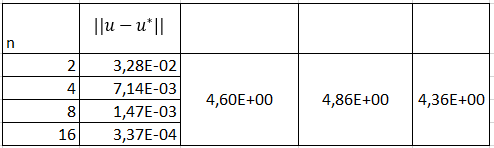
* λ = 1, γ = 1,





* λ = 1, γ = 1,





1. **Выводы**
2. **Исследование на порядок аппроксимации**

В результате исследования на порядок аппроксимации можно сказать, что при увеличении степени - искомой функции, на равномерной сетке начиная с , на неравномерной сетке начиная с , происходит увеличение погрешности. Это связано с тем, что частные вторые производные полиномов высших степеней – нелинейные функции. Таким образом, на **равномерной сетке третий порядок аппроксимации, а на неравномерной – второй.**

1. **Исследование на порядок сходимости**

В результате исследования на равномерной сетке, мы получили, что при дроблении сетки в 2 раза погрешность решения падает в 4 раза, следовательно, **порядок сходимости на равномерной сетке равен 2.**

В результате исследования на неравномерной сетке, мы получили, что при дроблении сетки в 2 раза для полинома 3 степени погрешность падает в 4 раза, а для не полиномиального решения – в 5 раз, следовательно, **порядок сходимости на неравномерной сетке равен 2.**