Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №1

по дисциплине

«Уравнения математической физики»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Преподаватели Постовалов Е. В.

Вариант 4 (T образный 1 и 3 краевые)

Новосибирск 2019

1. Цель работы

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

1. Задание

Уравнение:

Для функции

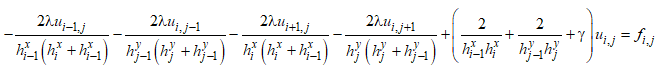
Краевые условия: 1

3:

Область имеет Т-образную форму

1. Анализ

Значение для узда внутри области рассчитывается по пятиточечной разностной схеме.



Значение для узла на границе с первой краевой задачей, уже указывает значение

Значение для узла на границе с третьей краевой задачей, указывает взаимоотношение между нормали и значением функции

После прохода по всем узлам получаем матрицу взаимоотношения узлов L и правую часть f

Решением этого СЛАУ так и будет значение u

Lu = f

Матрица будет пятидиагональном, поэтому оптимальным решателем будет метод Якоби или метод Зейделя.

*Для тестов, при 3 краевых условиях, левая границей является 1 краевой задачей, все остальные 3.*

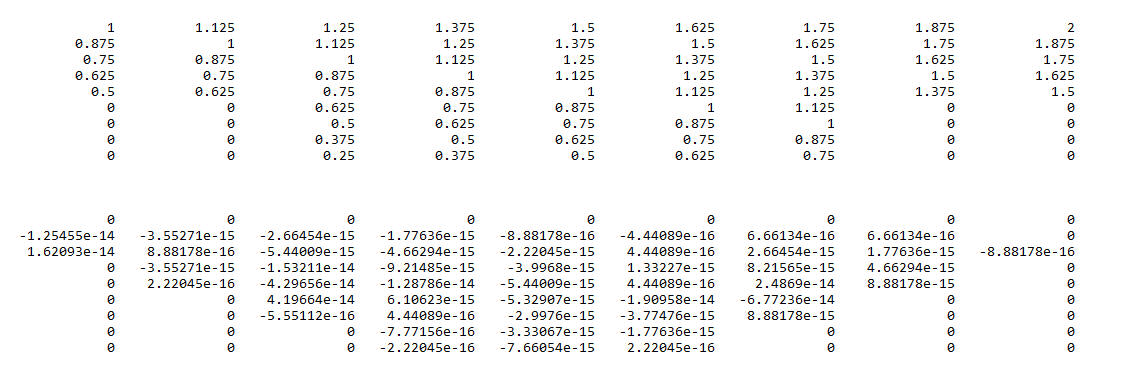
*Для тестов, при 1 краевых условиях, все границы являются задачами Дирихле.*

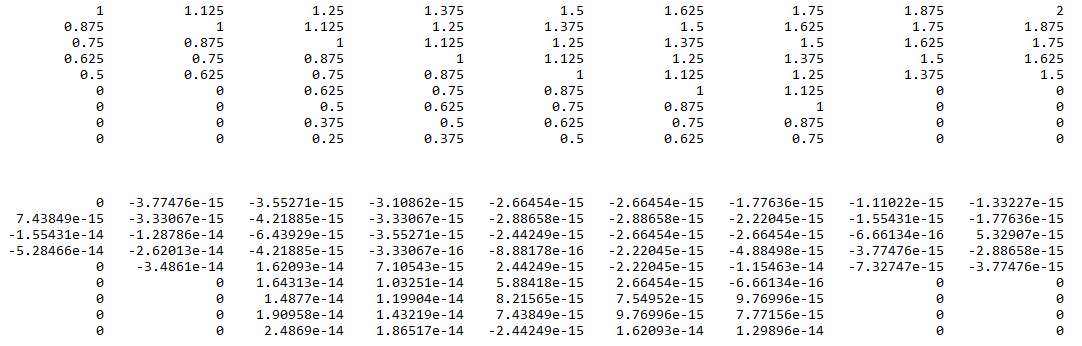
1. Тесты
2. *1 краевые полином 1 степени однородная*

Искомая функция:

Уравнение:

Краевые условия:



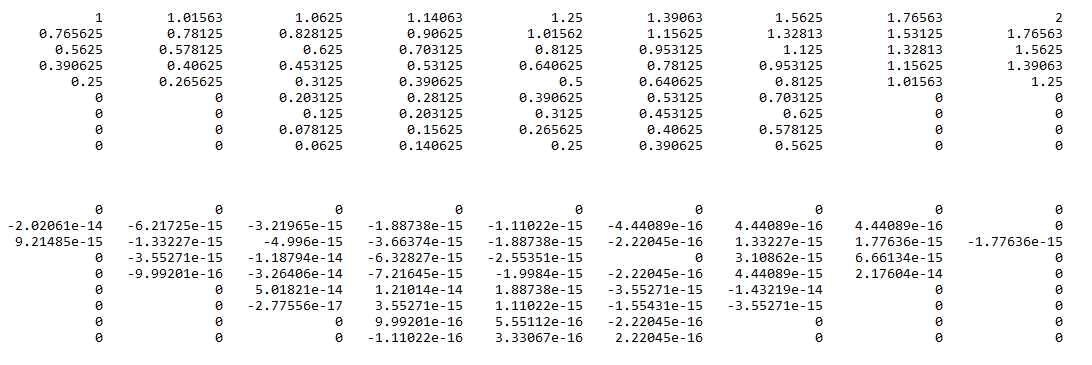


1. *1 краевые полином 2 степени однородная*

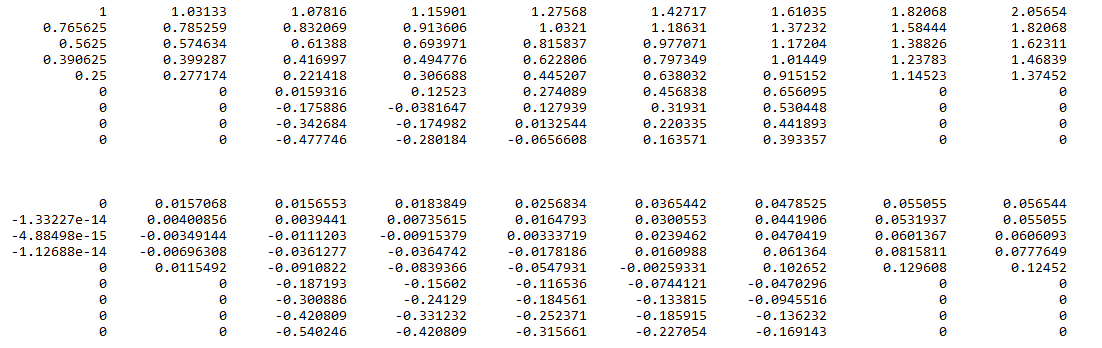
Искомая функция:

Уравнение:

Краевые условия:



*3 краевые*

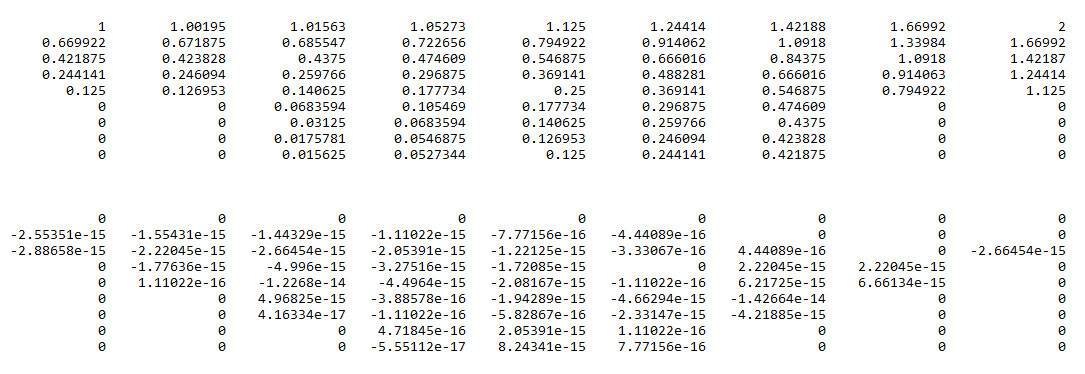


1. *1 краевые полином 3 степени однородная*

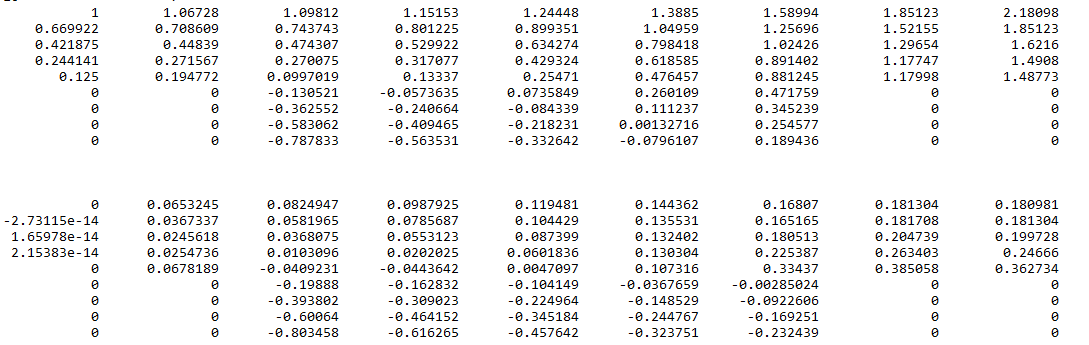
Искомая функция:

Уравнение:

Краевые условия:



*3 краевые*

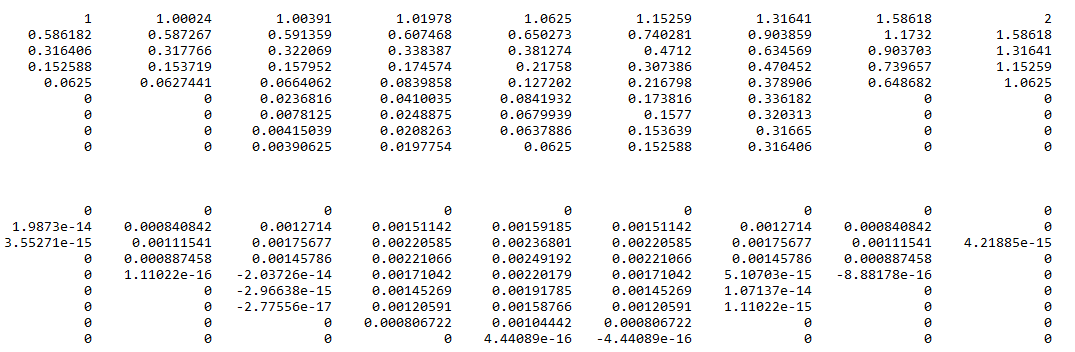


1. *1 краевые полином 4 степени однородная*

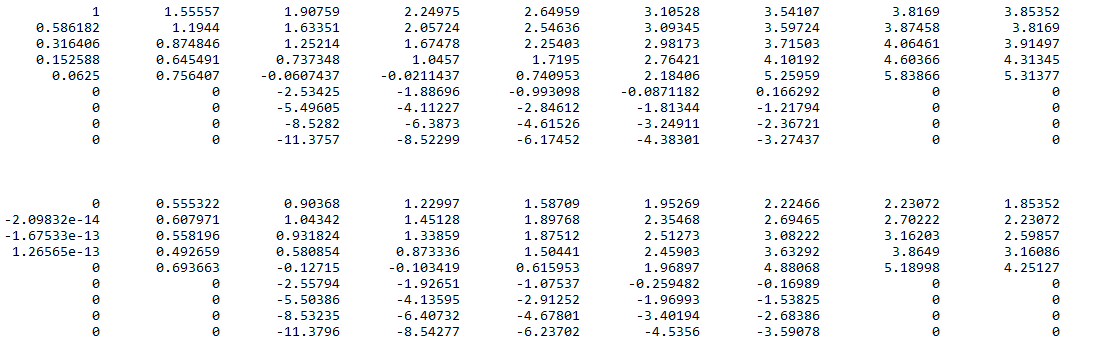
Искомая функция:

Уравнение:

Краевые условия:

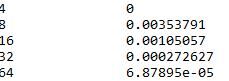


*3 краевые*

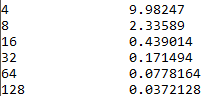


1. *1 краевые полином 4 степени для различных сеток*

Норма погрешности порядок сходимости

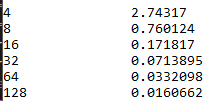
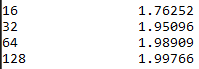
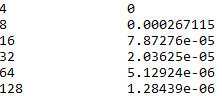


1. *3 краевые полином 2 степени для различных сеток*



1. *Экспанента для различных сеток*

*1 краевые порядок сходимости 3 краевые*



1. *Исследования на неравномерных сетках*

*Полином 2 степени x^2*Норма погрешности



*Экспоненциальная функция*

*Норма погрешности*



1. Текст программы

/\*Includeds.h\*/

#include<iostream>

#include<vector>

#include<algorithm>

#include<fstream>

#include<functional>

#include <numeric>

#include<Windows.h>

/\*Model.h\*/

#pragma once

#include"Includes.h"

using namespace std;

class Model

{

public:

//Форма

struct Shape

{

//Format matrix

// T - it's

// \* \* \* \* <- maxY

// \* \* \* \* <- YT

// \* \*

// \* \* <- minY

// A A A A

// | | | |

// m i a m

// i X X a

// n T T x

// X X

// or more point

enum Format

{

Rectangle,

T

};

Format type;

double minX, maxX;

double minY, maxY;

double iXT, aXT;

double YT;

};

//Граница

struct Border

{

enum Type

{

First,

Second,

Third

};

double b;

Type type;

function<double(double, double)> borderF;

};

Model();

~Model();

//Create Model

/// <param name="N">line</param>

/// <param name="M">column</param>

/// <param name="format">format of grid</param>

void CreateGrid(size\_t N = 4, size\_t M = 4);

/// <param name="shape">shape of area</param>

void SetShape(Shape shape);

/// <param name="border">border</param>

void AddBorder(Border border);

/// <param name="F">F from Lu = F. </param>

void SetF(function<double(double, double)> F);

/// <param name="lamda">New lamda </param>

void SetLamda(double lamda);

/// <param name="gamma">New gamma </param>

void SetGamma(double gamma);

void FDM();

vector<vector<double>> GetResult();

/// <param name="Kx">Коэфициент неравномерности по x </param>

/// <param name="Ky">Коэфициент неравномерности по y </param>

void SetKGrid(double Kx, double Ky);

double GethHI();

double GethHJ();

private:

struct ShapeIndex

{

size\_t maxX;

size\_t maxY;

size\_t iXT, aXT;

size\_t YT;

};

double lamda; //lamda

double gamma; //gamma

size\_t n, m; //Matrix n \* m

function<double(double, double)> F; //F function

vector<double> f; //f - local F

vector<double> u; //u - local U

vector<vector<double>> L; //L 5 diagonal matrix Lu = f

double hHI, hHJ; //первый шаг сетки

double hLT, hJT; //коэффициент неравномерности

Shape shape; //Shape of area

ShapeIndex index; //Shape in index

vector<Border> borders; //Border of area

void ConvertShape2Index();

void ConvertNeravShape2Index();

void ConvertL2MatrixFormat(); //Matrix - программа из прошлого года, которая решает диагональные матрицы методлом якоби и зегеля

vector<vector<double>> ConvertL2FullMatrix();

void CreateMatrixL();

};

*/\*Model.cpp\*/*

#include "Model.h"

#include "Includes.h"

#include "Matrix.cpp"

typedef void(\*Gause)(std::vector<std::vector<double>> &A, std::vector<double> &F, std::vector<double> &X);

Model::Model()

{

lamda = 1;

n = 4;

m = 4;

}

Model::~Model()

{

}

void Model::CreateGrid(size\_t N, size\_t M)

{

n = N;

m = M;

}

void Model::SetShape(Shape s)

{

shape = s;

}

void Model::AddBorder(Border border)

{

borders.push\_back(border);

}

void Model::SetF(function<double(double, double)> f)

{

F = f;

}

void Model::SetLamda(double l)

{

lamda = l;

}

void Model::SetGamma(double gamma)

{

this->gamma = gamma;

}

void Model::FDM()

{

//ConvertShape2Index();

ConvertNeravShape2Index();

CreateMatrixL();

Matrix<double> a;

auto hinstLib = LoadLibrary(TEXT("LDU.dll"));

Gause gause = (Gause)GetProcAddress(hinstLib, "Gause");

vector<vector<double>> A = ConvertL2FullMatrix();

#pragma region Мусор

//Matrix m;

//m.low\_diag.resize(0);

//m.high\_diag.resize(0);

//m.mid\_diag1.resize(0);

//m.mid\_diag2.resize(0);

//m.mid\_diag3.resize(0);

//

//m.max\_iter = 1000;

//m.block\_size = n;

////m.accuracy = 1e-10;

//for (size\_t i = 0; i < A.size(); i++)

// m.mid\_diag2.push\_back(A[i][i]);

//for (size\_t i = 0, j = 1; i < A.size() - 1; i++, j++)

//{

// m.mid\_diag1.push\_back(A[i][j]);

// m.mid\_diag3.push\_back(A[j][i]);

//}

//for (size\_t i = 0, j = n; i < A.size() - n; i++, j++)

//{

// m.low\_diag.push\_back(A[j][i]);

// m.high\_diag.push\_back(A[i][j]);

//}

//m.n = m.mid\_diag2.size();

//m.m = n;

//vector<double> x(m.mid\_diag2.size());

//vector<double> temp(m.mid\_diag2.size());

//vector<double> temp1(m.mid\_diag2.size() - 1);

//vector<double> temp2(m.mid\_diag2.size());

//u = m.method\_block\_relaxation(x, f, temp, 1, temp1, temp2);

#pragma endregion

//for (size\_t i = 0; i < A.size(); i++)

//{

// for (size\_t j = 0; j < A[i].size(); j++)

// {

// cout << A[i][j] << " ";

// }

// cout << "\t\t\t" << f[i];

// cout << endl;

//}

gause(A, f, u);

//for (size\_t i = 0; i < L.size(); i++)

//{

// for (size\_t j = 0; j < 5; j++)

// {

// cout << L[i][j] << " ";

// }

// cout << " " << f[i] << endl;

//}

//ConvertL2MatrixFormat();

//cout << "\n\n\n\n\n" << endl;

//for (size\_t i = 0; i < L.size(); i++)

//{

// for (size\_t j = 0; j < 5; j++)

// {

// cout << L[i][j] << " ";

// }

// cout << " " << f[i] << endl;

//}

//a.SetMatrix(L.size(), n, L, f);

//a.GaussSeidelMethod();

//a.JacobiMethod();

//do

//{

// //a.JacobiMethod();

// a.GaussSeidelMethod();

//} while (a.GetLastError() != 2);

//cout << endl;

//u = a.result;

}

vector<vector<double>> Model::GetResult()

{

//Conver line to matrix

vector<vector<double>> U(n, vector<double>(m));

for (int i = n - 1, k = 0; i >= 0; i--)

for (size\_t j = 0; j < m; j++, k++)

U[i][j] = u[k];

return U;

}

void Model::SetKGrid(double Kx, double Ky)

{

hLT = Kx;

hJT = Ky;

}

double Model::GethHI()

{

return hHI;

}

double Model::GethHJ()

{

return hHJ;

}

void Model::ConvertShape2Index()

{

if (!n && !m)

return;

index.maxX = m;

index.maxY = n;

index.YT = (int)((shape.YT - shape.minY) / (shape.maxY - shape.minY) \* n + 0.5); //0.5 к ближайшему

// сделать разный шаг

index.iXT = (int)((shape.iXT - shape.minX) / (shape.maxX - shape.minX) \* n + 0.5); //0.5 к ближайшему

index.aXT = (int)((shape.aXT - shape.minX) / (shape.maxX - shape.minX) \* n + 0.5); //0.5 к ближайшему

//сделать разный шаг

hHI = (shape.maxY - shape.minY) / n;

hHJ = (shape.maxX - shape.minX) / m;

//Можно в формулы подставить hHI и hHj будет немного быстрее, но непонятно откуда формулы

}

void Model::ConvertNeravShape2Index()

{

if (!n && !m)

return;

index.maxX = m;

index.maxY = n;

double k = 1;

double sum = 0;

for (size\_t i = 0; i < n; i++, k \*= hLT)

sum += k;

hHI = (shape.maxX - shape.minX) / sum;

sum = 0;

k = 1;

for (size\_t i = 0; i < m; i++, k \*= hJT)

sum += k;

hHJ = (shape.maxY - shape.minY) / sum;

sum = shape.minX;

k = hHI;

int j = 0;

for (; j < n; j++, k\*=hLT)

{

if (shape.iXT - sum < k / 2)

{

index.iXT = j;

break;

}

sum += k;

}

for (; j < n; j++, k\*=hLT)

{

if (shape.aXT - sum < k / 2)

{

index.aXT = j;

break;

}

sum += k;

}

sum = shape.minY;

k = hHJ;

for (size\_t i = 0; i < m; i++, k\*=hJT)

{

if (shape.YT - sum < k / 2)

{

index.YT = i;

break;

}

sum += k;

}

}

void Model::ConvertL2MatrixFormat()

{

double buf;

double size = L.size();

for (size\_t i = 0, j = size - 1; i < size - n; i++, j--)

{

L[i][4] = L[i + n][4];

L[j][0] = L[j - n][0];

}

for (size\_t i = 0, j = size - 1; i < n; i++, j--)

{

L[j][4] = 0;

L[i][0] = 0;

}

for (size\_t i = 0, j = size - 1; i < size - 1; i++, j--)

{

L[i][3] = L[i + 1][3];

L[j][1] = L[j - 1][1];

}

L[size - 1][3] = 0;

L[0][1] = 0;

}

vector<vector<double>> Model::ConvertL2FullMatrix()

{

vector<vector<double>> A(n\*m, vector<double> (n\*m));

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

{

if(L[i][0])

A[i][i + n] = L[i][0];

if(L[i][1])

A[i][i + 1] = L[i][1];

A[i][i] = L[i][2];

if(L[i][3])

A[i][i - 1] = L[i][3];

if(L[i][4])

A[i][i - n] = L[i][4];

}

return A;

}

void Model::CreateMatrixL()

{

m++;

n++;

L.resize(n \* m);

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

{

for (size\_t j = 0; j < m; j++)

{

L[i \* m + j].resize(5);

if (shape.type == shape.T && i < index.YT && (j < index.iXT || j > index.aXT))

{

L[i \* m + j][2] = 1;

f.push\_back(0);

}

else

{

if (i == index.maxY || !i || !j || j == index.maxX || shape.type == shape.T && (index.YT == i && (j <= index.iXT || j >= index.aXT) ||

i <= index.YT && (index.iXT == j || index.aXT == j )))

{

int k = 0;

if (shape.type == shape.Rectangle)

{

if (!j && (i != index.maxY && i ||

borders[0].type == borders[0].First ||

i == index.maxY && borders[3].type != borders[3].First ||

!i && borders[1].type != borders[1].First))

k = 0;

if (!i && (j && j != index.maxX ||

borders[1].type == borders[1].First ||

j == index.maxX && borders[2].type != borders[2].First ||

!j && borders[0].type != borders[0].First))

k = 1;

if (j == index.maxX && (i && i != index.maxY ||

borders[2].type == borders[2].First ||

i == index.maxY && borders[3].type != borders[3].First ||

!i && borders[1].type != borders[1].First))

k = 2;

if (i == index.maxX && (j && j != index.maxX ||

borders[3].type == borders[3].First ||

j == index.maxX && borders[2].type != borders[2].First ||

!j && borders[0].type != borders[0].First))

k = 3;

double sum1 = shape.minX;

double sum2 = shape.minY;

double kI = hHI;

double kJ = hHJ;

for (size\_t k1 = 0; k1 < i; k1++, kI\*= hLT)

sum1 += kI;

for (size\_t k1 = 0; k1 < j; k1++, kJ\*= hJT)

sum2 += kJ;

f.push\_back(borders[k].borderF(sum1, sum2));

if (borders[k].type == borders[k].First)

{

L[i \* m + j][2] = 1;

continue;

}

bool p = borders[k].type == borders[k].Second;

switch (k)

{

case 0:

if (p)

{

L[i \* m + j][2] = lamda / kI;

L[i \* m + j][1] = -lamda / kI;

}

else

{

L[i \* m + j][2] = -lamda / kI + borders[k].b;

L[i \* m + j][1] = lamda / kI;

}

break;

case 1:

if (p)

{

L[i \* m + j][2] = lamda / kJ;

L[i \* m + j][0] = -lamda / kJ;

}

else

{

L[i \* m + j][2] = -lamda / kJ + borders[k].b;

L[i \* m + j][0] = lamda / kJ;

}

break;

case 2:

kI /= hLT;

if (p)

{

L[i \* m + j][2] = -lamda / kI;

L[i \* m + j][3] = lamda / kI;

}

else

{

L[i \* m + j][2] = lamda / kI + borders[k].b;

L[i \* m + j][3] = -lamda / kI;

}

break;

case 3:

kJ /= hJT;

if (p)

{

L[i \* m + j][2] = -lamda / kJ;

L[i \* m + j][4] = lamda / kJ;

}

else

{

L[i \* m + j][2] = lamda / kJ + borders[k].b;

L[i \* m + j][4] = -lamda / kJ;

}

break;

}

continue;

}

if (shape.type == shape.T)

{

if (!j && (i != index.maxY && i ||

borders[0].type == borders[0].First ||

i == index.maxY && borders[3].type != borders[3].First ||

i == index.YT && borders[4].type != borders[4].First))

k = 0;

if (!i && (j != index.aXT && j != index.maxX ||

borders[1].type == borders[1].First ||

j == index.aXT && borders[6].type != borders[6].First ||

j == index.iXT && borders[5].type != borders[5].First))

k = 1;

if (j == index.maxX && (i != index.YT && i != index.maxY ||

borders[2].type == borders[2].First ||

i == index.maxY && borders[3].type != borders[3].First ||

i == index.YT && borders[7].type != borders[7].First))

k = 2;

if (i == index.maxX && (j && j != index.maxX ||

borders[3].type == borders[3].First ||

j == index.maxX && borders[2].type != borders[2].First ||

!j && borders[0].type != borders[0].First))

k = 3;

if (i == index.YT && j <= index.iXT && (j && j != index.iXT ||

borders[4].type == borders[4].First ||

!j && borders[0].type != borders[0].First ||

j == index.iXT && borders[5].type != borders[5].First))

k = 4;

if (j == index.iXT && i <= index.YT && (i && i != index.YT ||

borders[5].type == borders[5].First ||

!i && borders[1].type != borders[1].First ||

i == index.YT && borders[4].type != borders[4].First))

k = 5;

if (j == index.aXT && i <= index.YT && (i && i != index.YT ||

borders[6].type == borders[6].First ||

!i && borders[1].type != borders[1].First ||

i == index.YT && borders[7].type != borders[7].First))

k = 6;

if (i == index.YT && j >= index.aXT && (j != index.maxX && j != index.aXT ||

borders[7].type == borders[7].First ||

j == index.maxX && borders[2].type != borders[2].First ||

j == index.aXT && borders[6].type != borders[6].First))

k = 7;

double sum1 = shape.minX;

double sum2 = shape.minY;

double kI = hHI;

double kJ = hHJ;

for (size\_t k1 = 0; k1 < i; k1++, kI \*= hLT)

sum1 += kI;

for (size\_t k1 = 0; k1 < j; k1++, kJ \*= hJT)

sum2 += kJ;

f.push\_back(borders[k].borderF(sum1, sum2));

if (borders[k].type == borders[k].First)

{

L[i \* m + j][2] = 1;

continue;

}

bool p = borders[k].type == borders[k].Second;

switch (k)

{

case 0:

if (p)

{

L[i \* m + j][2] = lamda / kI;

L[i \* m + j][1] = -lamda / kI;

}

else

{

L[i \* m + j][2] = -lamda / kI + borders[k].b;

L[i \* m + j][1] = lamda / kI;

}

break;

case 1:

if (p)

{

L[i \* m + j][2] = lamda / kJ;

L[i \* m + j][0] = -lamda / kJ;

}

else

{

L[i \* m + j][2] = -lamda / kJ + borders[k].b;

L[i \* m + j][0] = lamda / kJ;

}

break;

case 2:

kI /= hLT;

if (p)

{

L[i \* m + j][2] = -lamda / kI;

L[i \* m + j][3] = lamda / kI;

}

else

{

L[i \* m + j][2] = lamda / kI + borders[k].b;

L[i \* m + j][3] = -lamda / kI;

}

break;

case 3:

kJ /= hJT;

if (p)

{

L[i \* m + j][2] = -lamda / kJ;

L[i \* m + j][4] = lamda / kJ;

}

else

{

L[i \* m + j][2] = lamda / kJ + borders[k].b;

L[i \* m + j][4] = -lamda / kJ;

}

break;

case 4:

if (p)

{

L[i \* m + j][2] = lamda / kJ;

L[i \* m + j][0] = -lamda / kJ;

}

else

{

L[i \* m + j][2] = -lamda / kJ + borders[k].b;

L[i \* m + j][0] = lamda / kJ;

}

break;

case 5:

if (p)

{

L[i \* m + j][2] = lamda / kI;

L[i \* m + j][1] = -lamda / kI;

}

else

{

L[i \* m + j][2] = -lamda / kI + borders[k].b;

L[i \* m + j][1] = lamda / kI;

}

break;

case 6:

kI /= hLT;

if (p)

{

L[i \* m + j][2] = -lamda / kI;

L[i \* m + j][3] = lamda / kI;

}

else

{

L[i \* m + j][2] = lamda / kI + borders[k].b;

L[i \* m + j][3] = -lamda / kI;

}

break;

case 7:

if (p)

{

L[i \* m + j][2] = lamda / kJ;

L[i \* m + j][0] = -lamda / kJ;

}

else

{

L[i \* m + j][2] = -lamda / kJ + borders[k].b;

L[i \* m + j][0] = lamda / kJ;

}

break;

}

continue;

}

}

else

{

double sum1 = shape.minX;

double sum2 = shape.minY;

double kI = hHI;

double kJ = hHJ;

for (size\_t k1 = 0; k1 < i; k1++, kI \*= hLT)

sum1 += kI;

for (size\_t k1 = 0; k1 < j; k1++, kJ \*= hJT)

sum2 += kJ;

L[i \* m + j][0] = -lamda / pow(kI, 2);

L[i \* m + j][1] = -lamda / pow(kJ, 2);

L[i \* m + j][2] = gamma + lamda \*(1.0 / pow(kJ, 2) + 1.0 / pow(kJ / hJT, 2) + 1.0 / pow(kI, 2) + 1.0 / pow(kI / hLT, 2));

L[i \* m + j][3] = -lamda / pow(kJ / hJT, 2);

L[i \* m + j][4] = -lamda / pow(kI / hLT, 2);

f.push\_back(F(sum1, sum2));

}

}

}

}

}