Министерство науки и высшего образования РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Курсовая работа

по уравнениям математической физики

Факультет: ПМИ

Группа: ПМ-63

Студент: Майер В.А.

Варианты: 9

Преподаватель: Патрушев И.И.

Задорожный А.Г.

Персова М.Г.

Новосибирск

2019

Оглавление

[1. Постановка задачи 3](#_Toc10620307)

[2. Теоретическая часть 3](#_Toc10620308)

[2.1 Дискретизация по времени 3](#_Toc10620309)

[2.2 Вариационная постановка 4](#_Toc10620310)

[2.3 Конечноэлементная дискретизация 4](#_Toc10620311)

[2.4 Локальные матрицы и вектора конечных элементов 5](#_Toc10620312)

[3. Описание разработанной программы 6](#_Toc10620313)

[3.1 Задача области 6](#_Toc10620314)

[3.2 Хранение граничных условий 6](#_Toc10620315)

[3. 3 Хранение матрицы 6](#_Toc10620316)

[3.4 Правая часть уравнения 7](#_Toc10620317)

[3.5 Коэффициенты гиперболического уравнения 7](#_Toc10620318)

[3.6 Начальные условия 7](#_Toc10620319)

[3.7 Указание базиса 7](#_Toc10620320)

[3.8 Запуск программы 7](#_Toc10620321)

[3.9 Решатель 7](#_Toc10620322)

[4. Исследования и Тестирование программы 8](#_Toc10620323)

[4.1 Итерационный процесс по пространству без погрешности 8](#_Toc10620324)

[4.2 Итерационны процесс по пространству с погрешностью 8](#_Toc10620325)

[4.3 Зависимость от параметров гиперболического уравнения функция без времени 8](#_Toc10620326)

[4.4 Неравномерная сетка на полиноме 4ой степени 9](#_Toc10620327)

[4.5 Неравномерная сетка на полиноме 4ой степени 2 краевые 9](#_Toc10620328)

[4.6 Неравномерная сетка на полиноме 4ой степени 3 краевые 9](#_Toc10620329)

[4.7 Равномерная сетка на полиноме 4ой степени 10](#_Toc10620330)

[4.8 Зависимость от времени 11](#_Toc10620331)

[4.9 Дробление сетки по времени 11](#_Toc10620332)

[4.10 Экспонента 12](#_Toc10620333)

[5. Вывод 12](#_Toc10620334)

[6. Код программы 13](#_Toc10620335)

# Постановка задачи

Решить методом конечных элементов двухмерную гиперболическую задачу в декартовых координатах. Уравнение имеет вид.

Первые краевые:

Вторые краевые:

Третьи краевые:

Вид разностной схемы по времени: неявная, четырёхслойная

Вид конечных элементов: прямоугольники

Вид базисных функций: би квадратичный

Формат хранения матрицы СЛАУ: разряженный строчно-столбцовый

Метод решения СЛАУ: LOS, MSG, BCG\_STAB с LU предобуславливанием

# Теоретическая часть

## 2.1 Дискретизация по времени

Введем сетку по времени.

Будем рассматривать точки на шаге i, следовательно нам необходимо задавать 3 начальных значений u, поскольку представление нашей функции в виде полинома Лагранджа:

Выразим коэффициенты при аппроксимации первой производной r, второй k

После подстановки разностной схемы в уравнение, получаем:

## 2.2 Вариационная постановка

Выполним вариационную постановку методом Бубнова-Галеркина.

В общем виде постановка Бубнова-Галеркина для операторного уравнения Aq = b записывается в следующем виде:

Где

Для гиперболический задачи, постановка Бубного-Галеркина выглядит следующим образом:

Применив формулу Грина к этому выражению, получим:

## 2.3 Конечноэлементная дискретизация

Разобьем область Ω на непересекающиеся подобласти – конечные элементы: . В соответствии с заданием А граничные поверхности представим в виде объединения соответствующих отрезков прямоугольников: Тогда мы получаем:

В области Ω выбираем квадратичный финитный базис {}, обладающими свойствами:

- Базисная функция ассоциирована с узлом конечноэлементной сетки

- Базисная функция отличная от нуля только на тех конечных элементах, которые содержат точку

Рассмотрим отдельно интегралы по конечным элементам, по отрезкам, где заданы вторые краевые условия и отрезкам, где заданы третьи краевые условия

## 2.4 Локальные матрицы и вектора конечных элементов

В качестве локальных базисных функций на элементе выбираем 9 базисных функций:

*X1(x) = Y1(x) =*

*X2(x) = Y2(x) =*

*X3(x) = Y3(x) =*

Для конечного элемента, элементы матрицы массы жесткости принимают вид:

Элементы матрицы массы будут вычисляться:

Правая часть уравнения принимает вид:

Обозначим для сокращения

После вычисления, итоговые матрицы примут вид:

*–* компоненты локальных матриц жёсткости и массы соответствующих одномерных конченых элементов с квадратичными базисными функциями

# Описание разработанной программы

## 3.1 Задача области

Сетка является прямоугольной и задается двумя массивами.

Массивом точек по оси x (SetX) и массивом точек по оси y (SetY)

Сетка по времени также задается массивом точек (SetT)

vector<double> x; //Cетка по x

vector<double> y; //Сетка по y

vector<double> t; //Сетка по t

## 3.2 Хранение граничных условий

Граница хранит

TypeBorder type; //Тип краевого условия

double b; //бетта для третьих краевых

function<double(double, double, double)>& U; //функция краевого условия

Граничные функции хранятся как лямбда-выражения. Также для задания граничного условия, необходимо указать тип граничного условия :

Border::TypeBorder::First;

Border::TypeBorder::Second;

Border::TypeBorder::Third;

Для задания третьего граничного условия необходимо так же указать значения коэффициента бетта, если он не будет указан, то он будет считаться равным 1.

Для нормальной работы программы необходимо указать 4 граничных условия(грани квадрата). Если указать меньшее количество, то появится исключение, говорящее о необходимости ввести еще, если было введено больше, то будет учитываться только первые 4. Граничные условия указываются по порядку начиная от левой грани против часовой стрелки.

## 3. 3 Хранение матрицы

Матрица хранится в разряженном формате. Для простоты обращения была написана функция AF, которая позволяет получить ссылку на элемент матрицы, используя индексы плотного формата.

vector<double> di; //Диагональные элементы матрицы А

vector<double> ggu; //Верхний треугольник матрицы А в разреженном формате

vector<double> ggl; //Нижний треугольний матрицы А в разреженном формате

vector<size\_t> ig; // Массив индексов

vector<size\_t> jg; // Другой массив индексов

В случае если такого элемента в матрице нет, то возвращается сслыка на пустые байты.

//Ображение к элементку в разряженной матрице

double& AF(int i, int j)

## 3.4 Правая часть уравнения

Вектор правой части хранится как лямбда-выражение зависящее от x, y, t

function<double(double, double, double)> f;

## 3.5 Коэффициенты гиперболического уравнения

Коэффициенты уравнения являются константами

double lambda; //Лямбда

double sigma; //Сигма

double hi; //Хи

## 3.6 Начальные условия

Поскольку используется 4-х слойная схема, то для начала расчета необходимо указать как минимум 3 начальных значений. В случае если указано меньше возникает соответствующее исключение.

Начальное значение задается как вектор, со значениями в узлах сетки и дополнительных узлах (квадратичный базис).

Функция задачи начального значения

void AddSolution(vector<double> &q)

{

u.push\_back(q);

}

Если указано больше чем 3 начальных значений будут использоваться последние 3.

## 3.7 Указание базиса

Так же для начала работы необходимо указать базис, поскольку в варианте только единственный базис(Лагранджа квадратичный),то возможно выбрать только его.

## 3.8 Запуск программы

Запуск осуществляется функцией Start().

Для запуска необходимо:

* Задать базисы
* Задать константы
* Задать краевые условия
* Задать форму(сетку)
* Задать правую часть уравнения
* Задать начальные условия

Потом вызываем функция Start

И получаем результат по ссылки, путем вызова функции GetResult

Так же есть возможность задавать константы и сетки из файла

## 3.9 Решатель

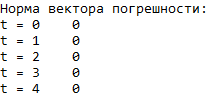
Для решения СЛАУ используется метод сопряжённых градиентов, локально оптимальная схема с LU предобуславлеванием, написанная на дисциплине Численные методы. Они вызываются с из динамической библиотеки, так же возможно использовать, метод бисопряжённых градиентов так же с LU пердобуславлеванием.

# Исследования и Тестирование программы

## 4.1 Итерационный процесс по пространству без погрешности

1 краевые

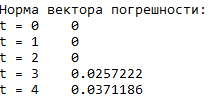
Норма вектора невязки



## 4.2 Итерационны процесс по пространству с погрешностью

1 краевые

Норма вектора невязки



Поскольку на полиноме 3 степени считалось без погрешности, а на полиноме 4 погрешность появляется, следовательно: ***Порядок аппроксимации*** данного метода по пространству равен 3

## 4.3 Зависимость от параметров гиперболического уравнения функция без времени

1 краевые

σ = 0 χ = 0 λ = 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| λ | Норма вектора невязки |  | σ, χ | Норма вектора невязки |
| 1 |  |  | 1, 1 |  |
| 5 |  |  | 5, 5 |  |
| 25 |  |  | 25, 25 |  |
| 100 |  |  | 100, 100 |  |
|  |  |  | λ = 0  σ= 1, χ= 1 |  |

## 4.4 Неравномерная сетка на полиноме 4ой степени

1 краевые

Неравномерная сетка по y

Коэффициент релаксации 0.5

|  |  |
| --- | --- |
| Количество конечных элементов по y | Норма вектора погрешности |
| 4 | 1.11316 |
| 8 | 0.221546 |
| 16 | 0.0818673 |
| 32 | 0.0186512 |

Порядок сходимости:

2.32898

1.43625

2.13402

## 4.5 Неравномерная сетка на полиноме 4ой степени 2 краевые

1 краевые и 2 краевые (Вторые краевыве слева сверхи и справа. Снизу первые краевые)

Неравномерная сетка по y

Коэффициент релаксации 0.5

|  |  |
| --- | --- |
| Количество конечных элементов по y | Норма вектора погрешности |
| 4 | 1.67565 |
| 8 | 8.09823e-10 |
| 16 | 2.74985e-08 |
| 32 | 0.0133606 |

Порядок сходимости:

30,94640

-5,08560

-18,89020

## 4.6 Неравномерная сетка на полиноме 4ой степени 3 краевые

1 краевые и 3 краевые (Вторые краевыве слева сверхи и справа. Снизу первые краевые)

Неравномерная сетка по y

Коэффициент релаксации 0.5

|  |  |
| --- | --- |
| Количество конечных элементов по y | Норма вектора погрешности |
| 4 | 1.33424 |
| 8 | 0.23739 |
| 16 | 0.136337 |
| 32 | 0.110099 |

Порядок сходимости:

2,49069

0,800082

0,308376

## 4.7 Равномерная сетка на полиноме 4ой степени

1 краевые

|  |  |
| --- | --- |
| Количество конечных элементов на ось | Норма вектора погрешности |
| 4 | 1.97849 |
| 8 | 0.0711037 |
| 16 | 0.00468841 |
| 32 | 0.000294295 |
| 64 | 1.83903e-05 |

Порядок сходимости:

4.79833

3.92275

3.99376

4,000025

2 краевые

|  |  |
| --- | --- |
| Количество конечных элементов на ось | Норма вектора погрешности |
| 4 | 1.95313 |
| 8 | 0.244141 |
| 16 | 0.0305176 |
| 32 | 0.00381473 |
| 64 | 0.000477053 |

Порядок сходимости:

2,9888

3,0000

2,9999

2,99936

3 краевые

|  |  |
| --- | --- |
| Количество конечных элементов на ось | Норма вектора погрешности |
| 4 | 1.52183 |
| 8 | 0.191708 |
| 16 | 0.0240048 |
| 32 | 0.00300147 |
| 64 | 0.000375191 |

Порядок сходимости:

2,9888

2,99752

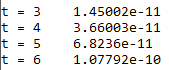
2,99958

2,999999

## 4.8 Зависимость от времени

t = {0, 1, 2, 3, 4, 5, 6}

1 краевые



## 4.9 Дробление сетки по времени

t = [0, 10]

1 краевые

*Дробление сетки по t*

|  |  |
| --- | --- |
| Количество конечных элементов по t | Норма вектора погрешности |
| 8 | 15.292 |
| 16 | 3.68988 |
| 32 | 0.892903 |
| 64 | 0.22984 |
| 128 | 0.0557547 |
| 256 | 0.0131135 |
| 512 | 0.00314803 |

Порядок сходимости по времени:

2.05113

2.04700

1.95787

2.04346

2.08804

2.05853

*Дробление сетки по x,y*

|  |  |
| --- | --- |
| Количество конечных элементов по x, y | Норма вектора погрешности |
| 4 | 406.821 |
| 8 | 406.918 |
| 16 | 406.929 |
| 32 | 406.93 |

Порядок сходимости по времени: 0

Поскольку, на полиноме 3 степени по времени считалось без погрешности, а на полиноме 4 степени она появилось, следовательно ***порядок аппроксимации*** по времени данного метода является 3.

## 4.10 Экспонента

t = [0,1]

1 краевые

|  |  |
| --- | --- |
| Количество конечных элементов по t | Норма вектора погрешности |
| 8 | 0.43071 |
| 16 | 0.118328 |
| 32 | 0.0322181 |
| 64 | 0.0102159 |
| 128 | 0.00531086 |
| 256 | 0.00440167 |

Порядок сходимости по времени: 2

2 краевые (левая границы 1 краевые, остальные 2)

|  |  |
| --- | --- |
| Количество конечных элементов по t | Норма вектора погрешности |
| 8 | 6.21536 |
| 16 | 1.686 |
| 32 | 0.430288 |
| 64 | 0.10173 |
| 128 | 0.0195564 |

Порядок сходимости по времени: 2

3 краевые (левая границы 1 краевые, остальные 3)

|  |  |
| --- | --- |
| Количество конечных элементов по t | Норма вектора погрешности |
| 8 | 4.60536 |
| 16 | 1.2432 |
| 32 | 0.31748 |
| 64 | 0.0746489 |
| 128 | 0.01458 |

Порядок сходимости по времени: 2

# Вывод

Теоретические и практические порядки сходимости и аппроксимации по времени и пространству совпали.

# Код программы

#include "Includes.h"

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

typedef size\_t(\*MSGSolver)(std::vector<size\_t> &iig, std::vector<size\_t> &ijg, std::vector<double> &gl, std::vector<double> &gu, std::vector<double> &di, std::vector<double> &F, std::vector<double> &X, double EPS);

typedef void(\*LDU)(std::vector<double> al, std::vector<double> au, std::vector<double> di, std::vector<size\_t> &ia, std::vector<double> &F, std::vector<double> &X);

vector<double> operator\*(vector<vector<double>> a, vector<double> b)

{

vector<double> c(b.size());

double sum = 0;

int m = a.size();

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

{

sum = 0;

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

sum += a[i][j] \* b[j];

c[i] = sum;

}

return c;

}

vector<double> operator+(vector<double> a, vector<double> b)

{

vector<double> c(a.size());

double sum = 0;

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

c[i] = a[i] + b[i];

return c;

}

vector<double> operator-(vector<double> a, vector<double> b)

{

vector<double> c(a.size());

double sum = 0;

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

c[i] = a[i] - b[i];

return c;

}

vector<double> operator\*(vector<double> a, double b)

{

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

a[i] \*= b;

return a;

}

vector<double> operator/(vector<double> a, double b)

{

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

a[i] /= b;

return a;

}

//Разряженный формат (если нет, то плотный)

#define A1

class FEM

{

public:

#pragma region Public struct

//Базисы

struct Basis

{

enum BasisType

{

Lagrange,

Hermite

};

BasisType type;

int order;

};

//Граничные условия

struct Border

{

public:

enum TypeBorder

{

First,

Second,

Third

};

TypeBorder type;

double b;

function<double(double, double, double)>& U;

//Задание первой или второй краевой функции

Border(function<double(double, double, double)> &f, TypeBorder t) : U(f)

{

type = t;

b = 1.0;

}

//Задание третьей краевой функции

Border(function<double(double, double, double)> &f, TypeBorder t, double betta) : U(f)

{

type = t;

b = betta;

}

};

#pragma endregion

FEM() { lambda = 1;};

~FEM() {};

double eps; //Погрешнасть решения

int maxIter; //Максимальное еоличестов итераций на один слой

int lastIter; //Текущее количество итераций на слое

int lastIter2; //Текущее количество итераций на слое

long long tq1, tq2, tq3;

vector<double> q1;

vector<double> q2;

vector<double> q;

//Запуск метода конечных элементов

void Start()

{

checkdata();

Preprocessing();

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

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

LDU ldu = (LDU)GetProcAddress(hinstLib, "LDU");

MSGSolver MSG\_LOS\_BCGSTAB = (MSGSolver)GetProcAddress(hinstLib, "MSGSolver");

MSGSolver LOS = (MSGSolver)GetProcAddress(hinstLib, "LOS");

MSGSolver BSG\_STAB = (MSGSolver)GetProcAddress(hinstLib, "BSG\_STAB");

vector<double> localF(pow(basis.order + 1, 2));

Matrix Glocal(9);

Matrix Mlocal(9);

double hx, hy;

vector<double> ts(4);

vector<double> th(4);

vector<double> q1(9); //значение q на k - 1 шаге по времени

vector<double> q2(9); //значение q на k - 2 шаге по времени

vector<double> q3(9); //значение q на k - 3 шаге по времени

double mid\_x, mid\_y;

Matrix Alocal;

vector<double> localb;

auto qlast = q;

for (size\_t k = u.size(); k < t.size(); k++)

{

ClearLastIter();

lastIter = 0;

q = u[u.size() - 1];

qlast = q;

ts = { ((t[k - 2] - t[k]) \* (t[k - 1] - t[k])) / ((t[k - 3] - t[k - 2]) \* (t[k - 3] - t[k - 1]) \* (t[k - 3] - t[k])),

((t[k - 3] - t[k]) \* (t[k] - t[k - 1])) / ((t[k - 3] - t[k - 2])\* (t[k - 2] - t[k - 1]) \* (t[k - 2] - t[k])),

((t[k - 3] - t[k]) \* (t[k - 2] - t[k])) / ((t[k - 1] - t[k - 3]) \* (t[k - 1] - t[k - 2]) \* (t[k - 1] - t[k])),

1.0 / (t[k] - t[k - 3]) + 1.0 / (t[k] - t[k - 2]) + 1.0 / (t[k] - t[k - 1])};

th = { -2.0 \* (t[k - 2] + t[k - 1] - 2.0 \*t[k]) / ((t[k - 3] - t[k - 2]) \* (t[k - 3] - t[k - 1]) \* (t[k - 3] - t[k])),

2.0 \* (t[k - 3] + t[k - 1] - 2.0\* t[k]) / ((t[k - 3] - t[k - 2])\*(t[k - 2] - t[k - 1]) \* (t[k - 2] - t[k])),

2.0 \* (t[k - 3] + t[k - 2] - 2.0\*t[k]) / ((t[k - 3] - t[k - 1]) \* (t[k - 1] - t[k - 2]) \* (t[k - 1] - t[k])),

2.0 \* (t[k - 3] + t[k - 2] + t[k - 1] - 3.0 \* t[k]) / ((t[k - 3] - t[k]) \* (t[k] - t[k - 2]) \* (t[k] - t[k - 1]))};

do

{

for (size\_t i = 0; i < x.size() - 1; i++)

{

for (size\_t j = 0; j < y.size() - 1; j++)

{

hx = x[i + 1] - x[i];

hy = y[j + 1] - y[j];

for (size\_t i = 0; i < 9; i++)

for (size\_t j = 0; j < 9; j++)

Glocal.x[i][j] = (hy / hx \* G.x[mu(i)][mu(j)] \* M.x[nu(i)][nu(j)] + hx / hy \* G.x[nu(i)][nu(j)] \* M.x[mu(i)][mu(j)]);

for (size\_t i = 0; i < 9; i++)

for (size\_t j = 0; j < 9; j++)

Mlocal.x[i][j] = hx \* hy \* M.x[mu(i)][mu(j)] \* M.x[nu(i)][nu(j)];

CreateLocalA(Glocal, Mlocal, Alocal, ts[3], th[3]);

CreateLocalF(i, j, k, localF);

CreateLocalQ(i, j, k, q1, q2, q3);

CreateLocalb(Mlocal, localb, localF, q1, q2, q3, ts[2], ts[1], ts[0], th[2], th[1], th[0]);

AddLocalMatrix(i, j, Alocal, localb);

}

}

AddBorderInMatrix(k);

//Prof2Raz();

//high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

////lastIter = MSG(ig, jg, ggl, ggu, di, b, q1, eps);

//lastIter = LOS(ig, jg, ggl, ggu, di, b, q1, eps);

//high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

//tq1 = duration\_cast<microseconds>(t2 - t1).count();

//for (auto& i : A)

//{

// for (auto& j : i)

// cout << j << " ";

// cout << endl;

//}

//cout << Norm(A \* q - b) / Norm(b) << endl;

#ifdef A1

MultAF(q, localb);

if (Norm(localb - b) / Norm(b) < eps)

break;

qlast = q;

//LOS(ig, jg, ggl, ggu, di, b, q, eps);

//BSG\_STAB(ig, jg, ggl, ggu, di, b, q, eps);

MSG\_LOS\_BCGSTAB(ig, jg, ggl, ggu, di, b, q, eps);

#else

if (Norm(A \* q - b) / Norm(b) < eps)

break;

qlast = q;

gause(A, b, q);

#endif // A1

} while (Norm(q - qlast) / Norm(q) < eps);

u.push\_back(q);

}

}

//Высвобождение памяти //кототрого нет!!!

void Clear()

{

}

void GetResult(vector<vector<double>> &U)

{

U = u;

}

#pragma region SetFunctions

//Задать базис

void SetBasis(Basis b)

{

basis = b;

}

//Задать Сигму

void SetSigma(double Sigma) { sigma = Sigma; }

//Задать Лямбду

void SetLambda(double l) { lambda = l; }

//Задать Хи

void SetHi(double Hi) { hi = Hi; }

//Доьавление решения

void AddSolution(vector<double> &q)

{

u.push\_back(q);

}

//Добавление стороны

void AddBorder(Border b)

{

border.push\_back(b);

}

//Задание правой части

void SetF(function<double(double, double, double)> func)

{

f = func;

}

//SetX

void SetX(vector<double> X) { x = X; }

//SetY

void SetY(vector<double> Y) { y = Y; }

//SetT

void SetT(vector<double> T) { t = T; }

#pragma endregion

private:

#pragma region Privat struct

struct ShapeIndex

{

int ixMax; //Максимальный индекс

int iyMax;

};

struct Matrix

{

Matrix()

{

n = 0;

}

Matrix(int N)

{

n = N;

x.resize(n);

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

x[i].resize(n);

}

Matrix(vector<vector<double>> x0)

{

x = x0;

n = x0.size();

}

vector<vector<double>> x;

size\_t n;

void Clear()

{

x.clear();

}

void resize(int N)

{

n = N;

x.resize(n);

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

x[i].resize(n);

}

Matrix operator\*(double b)

{

Matrix a(x);

for (size\_t i = 0; i < a.x.size(); i++)

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

a.x[i][j] \*= b;

return a;

}

Matrix operator/(double b)

{

Matrix a(x);

for (size\_t i = 0; i < a.x.size(); i++)

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

a.x[i][j] /= b;

return a;

}

Matrix operator+(Matrix b)

{

Matrix c(x);

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

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

c.x[i][j] += b.x[i][j];

return c;

}

Matrix operator-(Matrix b)

{

Matrix c(x);

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

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

c.x[i][j] -= b.x[i][j];

return c;

}

void operator\*=(double b)

{

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

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

x[i][j] \*= b;

}

vector<double> operator\*(vector<double> b)

{

vector<double> a(n);

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

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

a[i] += x[i][j] \* b[j];

return a;

}

};

#pragma endregion

//Дано

Basis basis; //Базис

vector<Border> border; //Границы

vector<double> x; //Cетка по x

vector<double> y; //Сетка по y

vector<double> t; //Сетка по t

double lambda; //Лямбда

double sigma; //Сигма

double hi; //Хи

function<double(double, double, double)> f;

//Результат

vector<vector<double>> u; //Искомая функция

//Переходные вычисления

int m;

#ifndef A1

vector<vector<double>> A; //Матрица левой части в диагональном виде

#endif // !A1

vector<double> b; //Вектора правой части

Matrix M;

Matrix G;

vector<double> di; //Диагональные элементы матрицы А

vector<double> ggu; //Верхний треугольник матрицы А в разреженном формате

vector<double> ggl; //Нижний треугольний матрицы А в разреженном формате

vector<size\_t> ig; // Массив индексов

vector<size\_t> jg; // Другой массив индексов

ShapeIndex index;

//Обработка введеных данных

void Preprocessing()

{

if (t.size() < u.size())

throw "Установлино неправильное число времен";

if (u.size() < 3)

throw "Введено недостаточно начальных условий";

index.ixMax = 2 \* x.size() - 1;

index.iyMax = 2 \* y.size() - 1;

m = index.ixMax \* index.iyMax;

b.resize(m);

q.resize(m);

int order = basis.order + 1;

di.resize(m);

ig.resize(m + 1);

#ifdef A1

for (size\_t i = 0, i1 = 0; i < x.size() - 1; i++, i1 += 2)

for (size\_t j = 0, j1 = 0; j < y.size() - 1; j++, j1 += 2)

for (size\_t j2 = 0; j2 < 3; j2++)

for (size\_t i2 = 0; i2 < 3; i2++)

for (size\_t j3 = 0; j3 < 3; j3++)

for (size\_t i3 = 0; i3 < 3; i3++)

CreateA1Elem((i1 + i3) + (j1 + j3) \* index.ixMax, (i1 + i2) + (j1 + j2) \* index.ixMax);

#else

A.resize(m);

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

A[i].resize(m);

#endif // A1

CreateMG();

}

//Создание матрицы массы и матрицы жесткости для 1D

void CreateMG()

{

G.x = vector<vector<double>>{ {7, -8, 1}, {-8, 16, -8}, {1, -8, 7}};

G = G \* (lambda / 3.0);

G.n = 3;

M.x = vector<vector<double>>{ {4, 2, -1}, {2, 16, 2}, {-1, 2, 4}};

M = M / 30.0;

M.n = 3;

}

//Создание локальной матрицы A

inline void CreateLocalA(Matrix &G, Matrix &M, Matrix &Alocal, double ts, double th)

{

Alocal = G + M \* (ts \* sigma + th \* hi);

}

//Создание локаольного вектора B

inline void CreateLocalb(Matrix &M, vector<double>& blocal, vector<double> &localf,

vector<double> &qlast1, vector<double> &qlast2, vector<double> &qlast3,

double &ts1, double &ts2, double &ts3, double &th1, double &th2, double &th3)

{

blocal = M \* (localf - (((qlast1 \* ts1 + qlast2 \* ts2 + qlast3 \* ts3) \* sigma) + ((qlast1 \* th1 + qlast2 \* th2 + qlast3 \* th3) \* hi)));

}

//Провкрка введенных данных

void checkdata()

{

}

//Добавление локальных матриц в глобальную

void AddLocalMatrix(int i, int j, Matrix &localA, vector<double> &localb)

{

vector<int> local2full;

for (size\_t j1 = 0, j2 = 2 \* j; j1 < 3; j1++, j2++)

for (size\_t i1 = 0, i2 = 2 \* i; i1 < 3; i1++, i2++)

local2full.push\_back(i2 + j2 \* index.ixMax);

for (size\_t j = 0; j < 9; j++)

{

for (size\_t i = 0; i < 9; i++)

#ifdef A1

AF(local2full[j], local2full[i]) += localA.x[j][i];

#else

A[local2full[j]][local2full[i]] += localA.x[j][i];

#endif // A1

b[local2full[j]] += localb[j];

}

}

//Очистка A и b c предыдущей итерации

void ClearLastIter()

{

#ifdef A1

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

{

ggu[i] = 0;

ggl[i] = 0;

}

for (auto& i : di)

i = 0;

#else

for(auto& i : A)

for (auto& j : i)

j = 0;

#endif // A1

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

b[i] = 0;

}

//Очищение места где должны быть краевые условия

void SetThirdBorderA(double hx, int p1, int p2, int p3)

{

#ifdef A1

AF(p1,p1) += M.x[0][0] \* hx; AF(p1,p2) += M.x[0][1] \* hx; AF(p1,p3) += M.x[0][2] \* hx;

AF(p2,p1) += M.x[1][0] \* hx; AF(p2,p2) += M.x[1][1] \* hx; AF(p2,p3) += M.x[1][2] \* hx;

AF(p3,p1) += M.x[2][0] \* hx; AF(p3,p2) += M.x[2][1] \* hx; AF(p3,p3) += M.x[2][2] \* hx;

#else

A[p1][p1] += M.x[0][0] \* hx; A[p1][p2] += M.x[0][1] \* hx; A[p1][p3] += M.x[0][2] \* hx;

A[p2][p1] += M.x[1][0] \* hx; A[p2][p2] += M.x[1][1] \* hx; A[p2][p3] += M.x[1][2] \* hx;

A[p3][p1] += M.x[2][0] \* hx; A[p3][p2] += M.x[2][1] \* hx; A[p3][p3] += M.x[2][2] \* hx;

#endif // A1

}

//Добавление краевых условаий в матрицу А и b

void AddBorderInMatrix(int k)

{

//ClearBorder();

int p;

int p1;

int p2;

//Левая сторона не 1 краевое

if (border[0].type != border[0].First)

for (size\_t i = 0; i < y.size() - 1; i++)

{

p = (2 \* i) \* index.ixMax;

p1 = (2 \* i + 1) \* index.ixMax;

p2 = (2 \* i + 2) \* index.ixMax;

double hy = y[i + 1] - y[i];

if (border[0].type == border[0].Third)

{

hy \*= border[0].b;

SetThirdBorderA(hy, p, p1, p2);

}

vector<double> loc2b = M \* vector<double>{ border[0].U(x[0], y[i], t[k]),

border[0].U(x[0], (y[i + 1] + y[i]) / 2.0, t[k]),

border[0].U(x[0], y[i + 1], t[k])};

b[p] += loc2b[0] \* hy;

b[p1] += loc2b[1] \* hy;

b[p2] += loc2b[2] \* hy;

}

//Нижняя сторона не 1 краевое

if (border[1].type != border[1].First)

for (size\_t i = 0; i < x.size() - 1; i++)

{

p = 2 \* i;

p1 = 2 \* i + 1;

p2 = 2 \* i + 2;

double hx = x[i + 1] - x[i];

if (border[1].type == border[1].Third)

{

hx \*= border[1].b;

SetThirdBorderA(hx, p, p1, p2);

}

vector<double> loc2b = M \* vector<double>{ border[1].U(x[i], y[0], t[k]),

border[1].U((x[i + 1] + x[i]) / 2, y[0], t[k]),

border[1].U(x[i + 1], y[0], t[k])};

b[p] += loc2b[0] \* hx;

b[p1] += loc2b[1] \* hx;

b[p2] += loc2b[2] \* hx;

}

//Правая сторона не 1 краевое

if (border[2].type != border[2].First)

for (size\_t i = 0; i < y.size() - 1; i++)

{

p = ((2 \* i + 1)\* index.ixMax - 1);

p1 = ((2 \* i + 2)\* index.ixMax - 1);

p2 = ((2 \* i + 3)\* index.ixMax - 1);

double hy = y[i + 1] - y[i];

if (border[2].type == border[2].Third)

{

hy \*= border[2].b;

SetThirdBorderA(hy, p, p1, p2);

}

vector<double> loc2b = M \* vector<double>{ border[2].U(x[x.size() - 1], y[i], t[k]),

border[2].U(x[x.size() - 1], (y[i + 1] + y[i]) / 2.0, t[k]),

border[2].U(x[x.size() - 1], y[i + 1], t[k])};

b[p] += loc2b[0] \* hy;

b[p1] += loc2b[1] \* hy;

b[p2] += loc2b[2] \* hy;

}

//Верхняя сторона не 1 краевое

if (border[3].type != border[3].First)

for (size\_t i = 0; i < x.size() - 1; i++)

{

p = index.ixMax \* (index.iyMax - 1) + 2 \* i;

p1 = index.ixMax \* (index.iyMax - 1) + 2 \* i + 1;

p2 = index.ixMax \* (index.iyMax - 1) + 2 \* i + 2;

double hx = x[i + 1] - x[i];

if (border[3].type == border[3].Third)

{

hx \*= border[3].b;

SetThirdBorderA(hx, p, p1, p2);

}

vector<double> loc2b = M \* vector<double>{ border[3].U(x[i], y[y.size() - 1], t[k]),

border[3].U((x[i + 1] + x[i]) / 2, y[y.size() - 1], t[k]),

border[3].U(x[i + 1], y[y.size() - 1], t[k])};

b[p] += loc2b[0] \* hx;

b[p1] += loc2b[1] \* hx;

b[p2] += loc2b[2] \* hx;

}

//Если левая сторона 1 краевое

if(border[0].type == border[0].First)

for (size\_t i = 0; i < index.iyMax; i++)

{

p = i \* index.ixMax;

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

#ifdef A1

AF(p, j, 0);

AF(p, p) = 1;

#else

A[p][j] = 0;

A[p][p] = 1;

#endif // A1

if (i % 2 == 0)

b[p] = border[0].U(x[0], y[i / 2], t[k]);

else

b[p] = border[0].U(x[0], (y[i / 2] + y[i / 2 + 1]) / 2, t[k]);

}

//Если нижняя сторона 1 краевое

if (border[1].type == border[1].First)

for (size\_t i = 0; i < index.ixMax; i++)

{

p = i;

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

#ifdef A1

AF(p, j, 0);

AF(p, p) = 1;

#else

A[p][j] = 0;

A[p][p] = 1;

#endif // A1

if (i % 2 == 0)

b[p] = border[1].U(x[i / 2], y[0], t[k]);

else

b[p] = border[1].U((x[i / 2] + x[i / 2 + 1]) / 2, y[0], t[k]);

}

//Если правая сторона 1 краевое

if (border[2].type == border[2].First)

for (size\_t i = 0; i < index.iyMax; i++)

{

p = ((i + 1)\* index.ixMax - 1);

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

#ifdef A1

AF(p, j, 0);

AF(p, p) = 1;

#else

A[p][j] = 0;

A[p][p] = 1;

#endif // A1

if (i % 2 == 0)

b[p] = border[2].U(x[(index.ixMax - 1) / 2], y[i / 2], t[k]);

else

b[p] = border[2].U(x[(index.ixMax - 1) / 2], (y[i / 2] + y[i / 2 + 1]) / 2, t[k]);

}

//Если верхняя сторона 1 краевое

if (border[3].type == border[3].First)

for (size\_t i = 0; i < index.ixMax; i++)

{

p = index.ixMax \* (index.iyMax - 1) + i;

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

#ifdef A1

AF(p, j, 0);

AF(p, p) = 1;

#else

A[p][j] = 0;

A[p][p] = 1;

#endif // A1

if (i % 2 == 0)

b[p] = border[3].U(x[i / 2], y[(index.iyMax - 1) / 2], t[k]);

else

b[p] = border[3].U((x[i / 2] + x[i / 2 + 1]) / 2, y[(index.iyMax - 1) / 2], t[k]);

}

}

//Норма

double Norm(vector<double> b)

{

double sum = 0;

for (auto &i : b)

sum += pow(i, 2);

return sqrt(sum);

}

//Перенос в 2D матрицу G и M из 1D

inline int mu(int i){return (i % 3);}

inline int nu(int i) { return i / 3;}

//Создаем на каждом шаге локальный вектор правой части

void CreateLocalF(int i, int j, int k, std::vector<double>& localF)

{

double mid\_x = (x[i + 1] + x[i]) / 2.0;

double mid\_y = (y[j + 1] + y[j]) / 2.0;

localF[0] = f(x[i], y[j], t[k]);

localF[1] = f(mid\_x, y[j], t[k]);

localF[2] = f(x[i + 1], y[j], t[k]);

localF[3] = f(x[i], mid\_y, t[k]);

localF[4] = f(mid\_x, mid\_y, t[k]);

localF[5] = f(x[i + 1], mid\_y, t[k]);

localF[6] = f(x[i], y[j + 1], t[k]);

localF[7] = f(mid\_x, y[j + 1], t[k]);

localF[8] = f(x[i + 1], y[j + 1], t[k]);

}

//Создание на каждом шаге локальный вектор решений на предыдущих итерациях

void CreateLocalQ(int i, int j, int k, std::vector<double>& q1, std::vector<double>& q2, std::vector<double>& q3)

{

for (size\_t j1 = 0, j2 = 2 \* j, p = 0; j1 < 3; j1++, j2++)

for (size\_t i1 = 0, i2 = 2 \* i; i1 < 3; i1++, i2++, p++)

{

q1[p] = u[k - 1][i2 + j2 \* index.ixMax];

q2[p] = u[k - 2][i2 + j2 \* index.ixMax];

q3[p] = u[k - 3][i2 + j2 \* index.ixMax];

}

}

#ifdef A1

//Добавлеие элемента в разряженную матрицу

void CreateA1Elem(int i, int j)

{

if (i == j)

return;

if (j > i)

{

int c = j;

j = i;

i = c;

}

for (size\_t k = ig[i]; k < ig[i + 1]; k++)

if (jg[k] == j)

return;

for (size\_t i1 = i + 1; i1 < ig.size(); i1++)

ig[i1]++;

jg.push\_back(0);

for (int k = ig[ig.size() - 1] - 1; k >= ig[i + 1]; k--)

jg[k] = jg[k - 1];

ggu.push\_back(0);

ggl.push\_back(0);

for (int k = ig[i]; k < ig[i + 1] - 1; k++)

{

if (jg[k] > j)

{

for (size\_t p = ig[i + 1] - 1; p > k; p--)

jg[p] = jg[p - 1];

jg[k] = j;

return;

}

}

jg[ig[i + 1] - 1] = j;

}

//Ображение к элементку в разряженной матрице

double& AF(int i, int j)

{

if (i == j)

return di[i];

if (i > j)

{

for (int k = ig[i]; k < ig[i + 1]; k++)

{

if (jg[k] == j)

return ggl[k];

}

}

else

{

for (int k = ig[j]; k < ig[j + 1]; k++)

{

if (jg[k] == i)

return ggu[k];

}

}

double p = 0;

return p;

}

//Умножение матрицы в разряженном формате на вектор

void MultAF(vector<double> &b, vector<double> &result)

{

result.resize(m);

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

{

result[i] = di[i] \* b[i];

for (int k = ig[i]; k < ig[i + 1]; k++)

{

int j = jg[k];

result[i] += ggl[k] \* b[j];

result[j] += ggu[k] \* b[i];

}

}

}

//Задать значение элементу матрицы

void AF(int i, int j, double c)

{

if (i == j)

{

di[i] = c;

return;

}

if (i > j)

{

for (int k = ig[i]; k < ig[i + 1]; k++)

{

if (jg[k] == j)

{

ggl[k] = c;

return;

}

}

}

else

{

for (int k = ig[j]; k < ig[j + 1]; k++)

{

if (jg[k] == i)

{

ggu[k] = c;

return;

}

}

}

}

#endif // A1

};

/\*Source.cpp\*/

#include "FEM.h"

//Константы точности

#define EPS 1e-14 //Погрешность невязки

#define DELTA 1e-14 //Погрешность разности шага решений

#define MAXITER 100000 //Максимальное количество итераций на каждый метод

//X и T

#define X {1, 3, 5, 7}

#define Y {1, 2, 3}

#define T {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

//Значение констант

#define lambda 3 //Лябда

#define sigma 1 //Функция сигма от x t и dudx

#define hi 0.1

//3 краевое

//#define betta 1

//#define U x \* x \* t \* t \* t + y \* y \* y \* t \* t

//

//#define DUDT 3 \* x \* x \* t \* t + 2 \* y \* y \* y \* t

//#define D2UDT2 6 \* x \* x \* t + 2 \* y \* y \* y

//

//#define DUDX 2 \* x \* t \* t \* t

//#define DUDY 3 \* y \* y \* t \* t

//

//#define DivGrad 2 \* t \* t \* t + 6 \* y \* t \* t

//#define Print

vector<double> createwline(double w, double a, double b, double n)

{

vector<double> c;

double r = (b - a)\* (1 - w) / (1 - pow(w, n));

c.push\_back(a);

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

{

r \*= w;

a += r;

c.push\_back(a);

}

return c;

}

vector<double> createline(double a, double b, double n)

{

vector<double> c;

double r = (b - a) / n;

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

c.push\_back(a + i \* r);

return c;

}

int main()

{

FEM m;

int n = 128;

int n0 = n;

int nt = 8;

vector<double> x = createline(0, 1, 10);

vector<double> y = createline(0, 1, 10);

vector<double> t = createline(0, 1, n);

vector<int> bord = Bord;

//for (size\_t i = 0; i < 100; i++)

//{

// x.push\_back(i);

// y.push\_back(i);

//}

#pragma region Функции

//Искомая функция

function<double(double, double, double)> u = [](double x, double y, double t) { return U; };

function<double(double, double, double)> dudt = [](double x, double y, double t) {return DUDT; };

function<double(double, double, double)> d2udt2 = [](double x, double y, double t) {return D2UDT2; };

function<double(double, double, double)> divgrad = [](double x, double y, double t) {return DivGrad; };

function<double(double, double, double)> dudx = [](double x, double y, double t) {return DUDX; };

function<double(double, double, double)> dudy = [](double x, double y, double t) {return DUDY; };

#pragma endregion

#pragma region Задаем базисы

FEM::Basis basis;

basis.type = basis.Lagrange;

basis.order = 2;

m.SetBasis(basis);

#pragma endregion

#pragma region Задаем константы

int NX = x.size();

int NY = y.size();

m.SetLambda(lambda);

m.SetSigma(sigma);

m.SetHi(hi);

m.eps = EPS;

m.maxIter = MAXITER;

#pragma endregion

#pragma region Задаем краевые условия

function<double(double, double, double t)> f1 = [&](double z, double y, double t) { return u(x[0], y, t) ; };

function<double(double, double, double t)> f2 = [&](double x, double z, double t) { return u(x, y[0], t) ; };

function<double(double, double, double t)> f3 = [&](double z, double y, double t) { return u(x[NX-1], y, t) ; };

function<double(double, double, double t)> f4 = [&](double x, double z, double t) { return u(x, y[NY-1], t) ; };

function<double(double, double, double t)> f12 = [&](double z, double y, double t) { return lambda \* (-dudx(x[0], y, t)); };

function<double(double, double, double t)> f22 = [&](double x, double z, double t) { return lambda \* (-dudy(x, y[0], t)); };

function<double(double, double, double t)> f32 = [&](double z, double y, double t) { return lambda \* (dudx(x[NX - 1], y, t)); };

function<double(double, double, double t)> f42 = [&](double x, double z, double t) { return lambda \* (dudy(x, y[NY - 1], t)); };

function<double(double, double, double t)> f13 = [&](double z, double y, double t) { return lambda / betta \* (-dudx(x[0], y, t)) + u(x[0], y, t); };

function<double(double, double, double t)> f23 = [&](double x, double z, double t) { return lambda / betta \* (-dudy(x, y[0], t)) + u(x, y[0], t); };

function<double(double, double, double t)> f33 = [&](double z, double y, double t) { return lambda / betta \* (dudx(x[NX - 1], y, t)) + u(x[NX - 1], y, t); };

function<double(double, double, double t)> f43 = [&](double x, double z, double t) { return lambda / betta \* (dudy(x, y[NY - 1], t)) + u(x, y[NY - 1], t); };

FEM::Border b1(f1, FEM::Border::First),

b2(f2, FEM::Border::First),

b3(f3, FEM::Border::First),

b4(f4, FEM::Border::First),

b12(f12, FEM::Border::Second),

b22(f22, FEM::Border::Second),

b32(f32, FEM::Border::Second),

b42(f42, FEM::Border::Second),

b13(f13, FEM::Border::Third, betta),

b23(f23, FEM::Border::Third, betta),

b33(f33, FEM::Border::Third, betta),

b43(f43, FEM::Border::Third, betta);;

switch (bord[0])

{

case 1: m.AddBorder(b1); break;

case 2: m.AddBorder(b12); break;

case 3: m.AddBorder(b13); break;

default: m.AddBorder(b1); break;

}

switch (bord[1])

{

case 1: m.AddBorder(b2); break;

case 2: m.AddBorder(b22); break;

case 3: m.AddBorder(b23); break;

default:m.AddBorder(b2); break;

}

switch (bord[2])

{

case 1: m.AddBorder(b3); break;

case 2: m.AddBorder(b32); break;

case 3: m.AddBorder(b33); break;

default: m.AddBorder(b3); break;

}

switch (bord[3])

{

case 1: m.AddBorder(b4); break;

case 2: m.AddBorder(b42); break;

case 3: m.AddBorder(b43); break;

default: m.AddBorder(b4); break;

}

#pragma endregion

#pragma region Задаем форму

m.SetX(x);

m.SetY(y);

m.SetT(t);

#pragma endregion

#pragma region Задаем правую часть уравнения

function<double(double, double, double)> F = [&](double x, double y, double t) {return lambda \* -1 \* divgrad(x, y, t) + sigma \* dudt(x, y, t) + hi \* d2udt2(x, y, t); };

m.SetF(F);

#pragma endregion

#pragma region Задаем начальное значени

for (size\_t k = 0; k < 3; k++)

{

vector<double> q;

for (size\_t i = 0, i1 = 0; i < y.size(); i++)

{

for (size\_t j = 0; j < x.size(); j++, i1++)

{

q.push\_back(u(x[j], y[i], t[k]));

if (j + 1 < x.size())

q.push\_back(u((x[j] + x[j + 1]) / 2, y[i], t[k]));

}

if(i + 1 < y.size())

for (size\_t j = 0; j < x.size(); j++, i1++)

{

q.push\_back(u(x[j], (y[i] + y[i + 1]) / 2, t[k]));

if(j + 1 < x.size())

q.push\_back(u((x[j] + x[j + 1]) / 2, (y[i] + y[i + 1]) / 2, t[k]));

}

}

m.AddSolution(q);

}

#pragma endregion

#pragma region Вывод

m.Start();

vector<vector<double>> res;

m.GetResult(res);

int w = 15

#ifdef Print

cout << "Решение" << endl;

for (size\_t k = 0; k < res.size(); k++)

{

cout << "t = " << t[k] << endl;

cout << setw(w) << "y\\x";

for (size\_t j = 0; j < 2 \* x.size() - 1; j++)

{

if (j % 2)

cout << setw(w) << (x[j / 2] + x[(j + 1) / 2]) / 2 << " ";

else

cout << setw(w) << x[j / 2] << " ";

}

cout << endl;

for (size\_t i = 0, i1 = 0; i < 2 \* y.size() - 1; i++)

{

if (i % 2)

cout << setw(w) << (y[i / 2] + y[(i + 1) / 2]) / 2 << " ";

else

cout << setw(w) << y[i / 2] << " ";

for (size\_t j = 0; j < 2 \* x.size() - 1; j++, i1++)

cout << setw(w) << res[k][i1] << " ";

cout << endl;

}

cout << endl;

}

cout << endl << "Погрешность" << endl;

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

{

cout << "t = " << t[i] << endl;

cout << setw(w) << "y\\x";

for (size\_t j = 0; j < 2 \* x.size() - 1; j++)

{

if (j % 2)

cout << setw(w) << (x[j / 2] + x[(j + 1) / 2]) / 2 << " ";

else

cout << setw(w) << x[j / 2] << " ";

}

cout << endl;

for (size\_t j = 0, j1 = 0; j < 2 \* y.size() - 1; j++)

{

{

if (j % 2)

{

cout << setw(w) << (y[j / 2] + y[(j + 1) / 2]) / 2 << " ";

for (size\_t k = 0; k < 2 \* x.size() - 1; k++, j1++)

if (k % 2)

cout << setw(w) << u((x[k / 2] + x[(k + 1) / 2]) / 2, (y[j / 2] + y[(j + 1) / 2]) / 2, t[i]) - res[i][j1] << " ";

else

cout << setw(w) << u(x[k / 2], (y[j / 2] + y[(j + 1) / 2]) / 2, t[i]) - res[i][j1] << " ";

}

else

{

cout << setw(w) << y[j / 2] << " ";

for (size\_t k = 0; k < 2 \* x.size() - 1; k++, j1++)

if (k % 2)

cout << setw(w) << u((x[k / 2] + x[(k + 1) / 2]) / 2, y[j / 2], t[i]) - res[i][j1] << " ";

else

cout << setw(w) << u(x[k / 2], y[j / 2], t[i]) - res[i][j1] << " ";

}

cout << endl;

}

}

normvect.push\_back(sqrt(norm));

norm = 0;

cout << endl;

}

#endif // Print

double norm = 0;

double normt = 0;

vector<double> normvect(0);

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

{

for (size\_t j = 0, j1 = 0; j < 2 \* y.size() - 1; j++)

{

if (j % 2)

{

for (size\_t k = 0; k < 2 \* x.size() - 1; k++, j1++)

if (k % 2)

{

if (j % (n / n0) == 0 && k % (n / n0) == 0) norm += pow(u((x[k / 2] + x[(k + 1) / 2]) / 2, (y[j / 2] + y[(j + 1) / 2]) / 2, t[i]) - res[i][j1], 2);

}

else

{

if (j % (n / n0) == 0 && k % (n / n0) == 0) norm += pow(u(x[k / 2], (y[j / 2] + y[(j + 1) / 2]) / 2, t[i]) - res[i][j1], 2);

}

}

else

{

for (size\_t k = 0; k < 2 \* x.size() - 1; k++, j1++)

if (k % 2)

{

if (j % (n / n0) == 0 && k % (n / n0) == 0) norm += pow(u((x[k / 2] + x[(k + 1) / 2]) / 2, y[j / 2], t[i]) - res[i][j1], 2);

}

else

{

if (j % (n / n0) == 0 && k % (n / n0) == 0) norm += pow(u(x[k / 2], y[j / 2], t[i]) - res[i][j1], 2);

}

}

//cout << endl;

}

if (i % (n / nt) == 0) normt += norm;

normvect.push\_back(sqrt(norm));

norm = 0;

}

std::cout << "!!!!!!!!! " << sqrt(normt) << endl;

std::cout << "Норма вектора погрешности: " << endl;

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

{

// std::cout << "t = " << t[i] << " " << normvect[i] << endl;

}

#pragma endregion

system("pause");

return 0;

}

/\*dllmain.cpp\*/

// dllmain.cpp : Определяет точку входа для приложения DLL.

#include "stdafx.h"

#include "LDU.cpp"

Matrix::Gause::Gause<double> gause;

Matrix::LDU::LDU<double> ldu;

extern "C"

{

\_declspec(dllexport) void Gause(std::vector<std::vector<double>> &A, std::vector<double> &F, std::vector<double> &X)

{

gause.A = A;

gause.f = F;

Matrix::Gause::CountUpTrianMatrix(gause);

Matrix::Gause::GausBack(gause);

X = gause.x;

}

\_declspec(dllexport) void LDU(std::vector<double> al, std::vector<double> au, std::vector<double> di, std::vector<size\_t> &ia, std::vector<double> &F, std::vector<double> &X)

{

ldu.n = di.size();

ldu.m = al.size();

ldu.x = X;

Matrix::LDU::CountLDU(ldu, al, au, di, ia);

Matrix::LDU::CountX(ldu, F);

X = ldu.x;

}

\_declspec(dllexport) size\_t MSGSolver(std::vector<size\_t> &iig,std::vector<size\_t> &ijg,std::vector<double> &gl,std::vector<double> &gu,std::vector<double> &di, std::vector<double> &F, std::vector<double> &X, double EPS)

{

Matrix::MSG\_LOS\_BCGSTAB::Matrix<double> msg;

size\_t n, m;

n = di.size();

m = gl.size();

msg.Init(n, m, iig, ijg, gl, gu, di, F, EPS);

msg.preconditioning(3);

msg.ConjugateGradient(3);

X = msg.Result();

return msg.K();

}

\_declspec(dllexport) size\_t LOS(std::vector<size\_t> &iig, std::vector<size\_t> &ijg, std::vector<double> &gl, std::vector<double> &gu, std::vector<double> &di, std::vector<double> &F, std::vector<double> &X, double EPS)

{

Matrix::MSG\_LOS\_BCGSTAB::Matrix<double> msg;

size\_t n, m;

n = di.size();

m = gl.size();

msg.Init(n, m, iig, ijg, gl, gu, di, F, EPS);

msg.preconditioning(3);

msg.LocalOptimalScheme(3);

X = msg.Result();

return msg.K();

}

\_declspec(dllexport) size\_t BSG\_STAB(std::vector<size\_t> &iig, std::vector<size\_t> &ijg, std::vector<double> &gl, std::vector<double> &gu, std::vector<double> &di, std::vector<double> &F, std::vector<double> &X, double EPS)

{

Matrix::MSG\_LOS\_BCGSTAB::Matrix<double> msg;

size\_t n, m;

n = di.size();

m = gl.size();

msg.Init(n, m, iig, ijg, gl, gu, di, F, EPS);

msg.preconditioning(3);

msg.BSG\_STAB(3);

X = msg.Result();

return msg.K();

}

}

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

setlocale(LC\_ALL, "Russian");

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

/\*LDU.cpp\*/

// LDU.cpp : Определяет экспортированные функции для приложения DLL.

//

#include "stdafx.h"

namespace Matrix

{

enum class Errors

{

noError,

ErrorOpenFile,

MatrixNotCount

};

enum class Format

{

LDU,

Profile

};

template<class T>

class Matrix

{

public:

Matrix() {}

~Matrix() {}

std::vector<T> al, au, di, f, x;

std::vector<size\_t> ia;

size\_t n, m;

Format format;

Errors error;

private:

};

namespace Gause

{

template<class T>

class Gause : Matrix<T>

{

public:

std::vector<std::vector<T>> A;

std::vector<double> f, x;

Gause()

{

this->format = Format::Profile;

error = Errors::noError;

}

~Gause() {}

Errors error;

private:

size\_t n; //Размерность матрицы

};

template<class T>

void GausBack(Gause<T> &matrix)

{

std::vector<T> &X = matrix.x;

std::vector<T> &F = matrix.f;

std::vector<std::vector<T>> &A = matrix.A;

auto n = F.size();

X.resize(n);

if (matrix.error != Errors::noError)

return;

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

{

double sum = 0;

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

sum += A[i][j] \* X[j];

X[i] = (F[i] - sum) / A[i][i];

}

}

template<class T>

void CountUpTrianMatrix(Gause<T> &matrix)

{

if (matrix.error != Errors::noError)

return;

std::vector<std::vector<T>> &A = matrix.A;

std::vector<T> &F = matrix.f;

auto n = F.size();

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

{

//Опредиление главного Элемента в столбце

T Max = A[i][i];

size\_t index = i;

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

{

if (abs(A[j][i]) > abs(Max))

{

Max = A[j][i];

index = j;

}

}

//Проверка единственность

if (!Max)

{

matrix.error = Errors::MatrixNotCount;

return;

}

//Свап строки

if (i != index)

{

A[index].swap(A[i]);

T buf = F[i];

F[i] = F[index];

F[index] = buf;

}

//Постройка верхней треугольной матрицы

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

{

if (!A[j][i])

continue;

T c = A[j][i] / Max;

A[j][i] = 0.;

for (size\_t l = i + 1; l < n; l++)

A[j][l] -= A[i][l] \* c;

F[j] -= F[i] \* c;

}

}

}

template<class T>

Gause<T> Create(std::vector<double> &al, std::vector<double> &au, std::vector<double> &di, std::vector<size\_t> &ia)

{

Gause<T> m;

std::vector<std::vector<T>> &A = m.A;

if (!di.size())

{

m.Error = Errors::MatrixNotCount;

return;

}

else

{

A.resize(m.n);

for (size\_t i = 0; i < m.n; A[i++].resize(m.n));

}

m.n = di.size();

auto n = m.n;

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

A[i][i] = di[i];

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

{

for (size\_t m = ia[i], j = i - ia[i + 1] + m; m < ia[i + 1]; m++, j++)

{

A[i][j] = al[m];

A[j][i] = au[m];

}

}

}

}

namespace LDU

{

template<class T>

class LDU: Matrix<T>

{

public:

LDU<T>()

{

this->format = Format::LDU;

}

~LDU<T>() {}

std::vector<T> al, au, di, f, x;

std::vector<size\_t> ia;

size\_t n, m;

Format format;

Errors error;

};

template<class T>

void CountX(LDU<T> &matrix, std::vector<T> F)

{

size\_t n = matrix.n;

std::vector<T> &D = matrix.di;

std::vector<T> &U = matrix.au;

std::vector<T> &L = matrix.al;

std::vector<T> &X = matrix.x;

Errors Error = matrix.error;

std::vector<size\_t> &ia = matrix.ia;

for (auto &elem : X)

elem = 0;

if (Error != Errors::noError)

return;

//Поиск Y

std::vector<T> &Y = X;

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

{

double sum = 0;

for (size\_t m = ia[i],

j = i - ia[i + 1] + m;

m < ia[i + 1]; m++)

sum += Y[j++] \* L[m];

Y[i] += F[i] - sum;

}

//Поиск Z

std::vector<T> &Z = X;

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

Z[i] /= D[i];

//Поиск X

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

for (int j = ia[i + 1] - ia[i] - 1,

j1 = i - j - 1,

j2 = ia[i + 1] - j - 1;

j >= 0; j--)

X[j1++] -= X[i] \* U[j2++];

}

template<class T>

void CountLDU(LDU<T> &matrix)

{

size\_t n = matrix.n;

size\_t m = matrix.m;

auto &D = matrix.di;

auto &U = matrix.au;

auto &L = matrix.al;

auto &Error = matrix.error;

auto &ia = matrix.ia;

double CompareNumber;

if (std::is\_same<T, double>::value)

CompareNumber = pow(10, 15);

else

CompareNumber = pow(10, 7);

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

{

double sum = 0;

for (size\_t j = ia[i], j1 = i - ia[i + 1] + j; j < ia[i + 1]; j++)

{

double sum1 = 0, sum2 = 0;

for (size\_t k = min(j - ia[i], ia[i - ia[i + 1] + j + 1] - ia[i - ia[i + 1] + j]),

i1 = j - k,

i2 = i - ia[i + 1] + j - k,

i3 = ia[i - ia[i + 1] + j + 1] - k;

k > 0; k--)

{

sum1 += L[i1] \* D[i2] \* U[i3];

sum2 += L[i3++] \* D[i2++] \* U[i1++];

}

L[j] = (L[j] - sum1) / D[j1];

U[j] = (U[j] - sum2) / D[j1];

sum += L[j] \* D[j1++] \* U[j];

}

//Если сделать проверка на inf после деления на 0, то можем получить не то что нам нужно

//Т.к. при разность (D[i] - sum) может получиться не 0, а число близкое к нему(разность нецелого типа)

//и в таком случае при делении числа на эту разность может получиться не inf.

if (abs((D[i] - sum)) < abs(D[i] / CompareNumber))

{

Error = Errors::MatrixNotCount;

return;

}

D[i] -= sum;

}

}

template <class T>

void CountLDU(LDU<T> &matrix, std::vector<T> &al, std::vector<T> &au, std::vector<T> &di, std::vector<size\_t> &ia)

{

size\_t n = matrix.n;

size\_t m = matrix.m;

auto &D = matrix.di;

auto &U = matrix.au;

auto &L = matrix.al;

D.resize(n);

U.resize(m);

L.resize(m);

auto &Error = matrix.error;

matrix.ia = ia;

double CompareNumber;

if (std::is\_same<T, double>::value)

CompareNumber = pow(10, 15);

else

CompareNumber = pow(10, 7);

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

{

double sum = 0;

for (size\_t j = ia[i], j1 = i - ia[i + 1] + j; j < ia[i + 1]; j++)

{

double sum1 = 0, sum2 = 0;

for (size\_t k = min(j - ia[i], ia[i - ia[i + 1] + j + 1] - ia[i - ia[i + 1] + j]),

i1 = j - k,

i2 = i - ia[i + 1] + j - k,

i3 = ia[i - ia[i + 1] + j + 1] - k;

k > 0; k--)

{

sum1 += L[i1] \* D[i2] \* U[i3];

sum2 += L[i3++] \* D[i2++] \* U[i1++];

}

L[j] = (al[j] - sum1) / D[j1];

U[j] = (au[j] - sum2) / D[j1];

sum += L[j] \* D[j1++] \* U[j];

}

//Если сделать проверка на inf после деления на 0, то можем получить не то что нам нужно

//Т.к. при разность (D[i] - sum) может получиться не 0, а число близкое к нему(разность нецелого типа)

//и в таком случае при делении числа на эту разность может получиться не inf.

if (abs((D[i] - sum)) < abs(D[i] / CompareNumber))

{

Error = Errors::MatrixNotCount;

return;

}

D[i] = di[i] - sum;

}

}

template<class U>

bool Open(std::string path, std::vector<U> &a, size\_t len)

{

std::ifstream ia;

ia.exceptions(std::ifstream::failbit | std::ifstream::badbit);

try

{

ia.open(path);

a.resize(len);

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

ia >> a[i];

ia.close();

}

catch (std::ifstream::failure e)

{

std::cout << "Error Open File: " << path << std::endl;

return false;

}

return true;

}

template<class T>

void OpenFile(LDU<T> &matrix, std::string path)

{

int error = 0;

size\_t n, m;

if (path.size())

path += "/";

std::vector<size\_t> info(2);

error +=

!Open(path + "info.txt", info, 2);

n = info[0], m = info[1];

matrix.x.resize(n);

error +=

!Open(path + "al.txt", matrix.al, m) +

!Open(path + "au.txt", matrix.au, m) +

!Open(path + "di.txt", matrix.di, n) +

!Open(path + "ia.txt", matrix.ia, n + 1) +

!Open(path + "F.txt", matrix.f, n);

if(error)

matrix.error = Errors::ErrorOpenFile;

}

template<class T>

LDU<T> Create(std::string path)

{

LDU<T> m;

OpenFile(m, path);

CountLDU(m);

return m;

}

template<class T>

LDU<T> Create(std::vector<double> &al, std::vector<double> &au, std::vector<double> &di, std::vector<size\_t> &ia)

{

LDU<T> m;

CountLDU(m, al, au, di, ia);

return m;

}

}

namespace MSG\_LOS\_BCGSTAB

{

template<class T>

class Matrix

{

public:

Matrix() { Maxiter = 10000; }

~Matrix() {}

void Init(size\_t n, size\_t m, std::vector<size\_t> iig, std::vector<size\_t> ijg, std::vector<double> gl, std::vector<double> gu, std::vector<double> Di, std::vector<double> F, double EPS)

{

N = n;

M = m;

e = EPS;

ig = iig;

jg = ijg;

result.resize(N);

ggl = gl;

ggu = gu;

di = Di;

f = F;

}

//Set matrix dimension

///<param name = "n"> New matrix dimension</param>

void SetN(size\_t n) { N = n; }

//Get matrix dimention

size\_t n() { return N; }

//Возращает резултат операций

std::vector<T> Result() { return result; }

//Возращает количество операций

size\_t K() { return k; }

//Set max iteration

///<param name = "Max"> New max ieration </param>

void Setmaxiter(size\_t Max) { Maxiter = Max; }

//Get max iteration

size\_t maxiter() { return Maxiter; }

///<param name = "i"> i = 0 без предобуславливония

///i = 1 диагональное

///i = 2 Холесского</param>

void preconditioning(int i)

{

switch (i)

{

//Без предобуславливония

case 0:

break;

//Диагональное

case 1:

// factorizationLU(false);

factorizationLLT();

break;

//Холесского

// case 2:

// factorizationLLT(true);

//break;

//LU

case 3:

factorizationLU(true);

break;

default:

break;

}

}

//i = 0 без предобуславливония

//i = 1 диагональное

//i = 2 Холесского</param>

void preconditioning()

{

preconditioning(3);

}

void BSG\_STAB(int metod)

{

NormF = Norm(f);

switch (metod)

{

case 0:

{

std::vector<T> Best = result;

r = Residual(f, MultMatrixVector(Best));

auto p = r;

z = r;

auto s = r;

std::vector<T> d;

T rpScolar = Scolar(r, p);

T rpLastScolar;

T alfa, betta;

T CoudA;

T min = 0.;

while (checkEnd(rpScolar, CoudA))

{

d = MultMatrixVector(z);

alfa = rpScolar / Scolar(s, d);

Best = Sum(Best, MultVectorOnT(z, alfa));

r = Residual(r, MultVectorOnT(d, alfa));

d = MultTMatrixVector(s);

p = Residual(p, MultVectorOnT(d, alfa));

rpLastScolar = rpScolar;

rpScolar = Scolar(r, p);

betta = rpScolar / rpLastScolar;

z = Sum(r, MultVectorOnT(z, betta));

z = Sum(p, MultVectorOnT(s, betta));

k++;

if (min > CoudA || !min)

{

min = CoudA;

result = Best;

}

k++;

}

break;

}

case 1:

case 2:

break;

case 3:

{

auto Best = result;

x = Up(U, D, result);

r = Residual(f, MultMatrixVector(x));

r = Down(L, r);

z = r;

auto p = r;

auto s = r;

T rpScolar = Scolar(r, p);

T rpLastScolar;

T alfa;

T betta;

NormF = Norm(f);

std::vector<T> d;

std::vector<T> buf;

std::vector<T> buf1;

T Coud;

T min = 0.;

while (checkEnd(rpScolar, Coud))

{

d = Up(U, D, z);

d = MultMatrixVector(d);

d = Down(L, d);

double c = Scolar(s, d);

if (!c)

break;

alfa = rpScolar / c;

buf1 = MultVectorOnT(z, alfa);

x = Sum(x, buf1);

buf1 = MultVectorOnT(d, alfa);

r = Residual(r, buf1);

buf = Up(L, s);

buf = MultTMatrixVector(buf);

buf = Down(U, D, buf);

buf1 = MultVectorOnT(buf, alfa);

p = Residual(p, buf1);

rpLastScolar = rpScolar;

rpScolar = Scolar(p, r);

betta = rpScolar / rpLastScolar;

buf1 = MultVectorOnT(z, betta);

z = Sum(r, buf1);

buf1 = MultVectorOnT(s, betta);

s = Sum(p, buf1);

if (min > Coud || !min)

{

min = Coud;

Best = x;

}

k++;

}

result = Up(U, D, Best);

//for (auto &i : result)

// i = 0;

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

//{

// for (size\_t j = ig[i]; j < ig[i + 1]; j++)

// result[jg[j]] += U[j] \* Best[i];

// result[i] += D[i] \* Best[i];

//}

break;

}

default:

break;

}

}

void ConjugateGradient(int metod)

{

NormF = Norm(f);

switch (metod)

{

//Без

case 0:

{

std::vector<T> Best = result;

f = MultTMatrixVector(f);

r = Residual(f, MultTMatrixVector(MultMatrixVector(Best)));

z = r;

T rLastScolar = Scolar(r);

T CoudA;

T min = 0.;

while (checkEnd(rLastScolar, CoudA))

{

std::vector<T> d = MultTMatrixVector(MultMatrixVector(z));

a = rLastScolar / Scolar(d, z);

Best = Sum(Best, MultVectorOnT(z, a));

rLast = r;

r = Residual(r, MultVectorOnT(d, a));

T lol = Scolar(r);

b = lol / rLastScolar;

rLastScolar = lol;

z = Sum(r, MultVectorOnT(z, b));

if (min > CoudA || !min)

{

min = CoudA;

result = Best;

}

k++;

}

break;

}

// Диаганальное

case 1:

//LLT

case 2:

{

std::vector<T> d = Residual(f, MultMatrixVector(result)); //r = U^-t \* A ^t \* L^ -T \* L-1(f - Ax)

d = Down(L, D, d);// L d1 = d

d = Up(L, D, d); //L^t d1 = d

d = MultTMatrixVector(d); // A^t \* d = d1

r = Down(L, D, d); // U^t r = d

z = r;

x.resize(N);

#pragma region Умножение\_x-\_=\_Ux

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

{

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

x[i] += L[j] \* result[jg[j]];

x[i] += D[i] \* result[i];

}

#pragma endregion

k = 0;

//xbest = x;

std::vector<T> Best = x;

T rLastScolar = Scolar(r);

T CoudA;

T min = 0.;

while (checkEnd(rLastScolar, CoudA))

{

std::vector<T> d = Up(L, D, z);

d = MultMatrixVector(d);

d = Down(L, D, d);

d = Up(L, D, d);

d = MultTMatrixVector(d);

d = Down(L, D, d);

T s = Scolar(d, z);

if (!s)

break;

a = rLastScolar / s;

x = Sum(x, MultVectorOnT(z, a));

rLast = r;

r = Residual(r, MultVectorOnT(d, a));

T rScolar = Scolar(r);

b = rScolar / rLastScolar;

z = Sum(r, MultVectorOnT(z, b));

rLastScolar = rScolar;

k++;

if (min > CoudA || !min)

{

min = CoudA;

Best = x;

}

}

result = Up(L, D, Best);

break;

}

//LU

case 3:

{

std::vector<T> d = Residual(f, MultMatrixVector(result)); //r = U^-t \* A ^t \* L^ -T \* L-1(f - Ax)

d = Down(L, d);// L d1 = d

d = Up(L, d); //L^t d1 = d

d = MultTMatrixVector(d); // A^t \* d = d1

r = Down(U, D, d); // U^t r = d

z = r;

x.resize(N);

#pragma region Умножение\_x-\_=\_Ux

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

{

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

x[i] += U[j] \* result[jg[j]];

x[i] += D[i] \* result[i];

}

#pragma endregion

k = 0;

xbest = x;

std::vector<T> Best = x;

T rLastScolar = Scolar(r);

T Coud;

T min = 0.;

while (checkEnd(rLastScolar, Coud))

{

std::vector<T> d = Up(U, D, z);

d = MultMatrixVector(d);

d = Down(L, d);

d = Up(L, d);

d = MultTMatrixVector(d);

d = Down(U, D, d);

T s = Scolar(d, z);

if (!s)

break;

a = rLastScolar / s;

x = Sum(x, MultVectorOnT(z, a));

r = Residual(r, MultVectorOnT(d, a));

T rScolar = Scolar(r);

b = rScolar / rLastScolar;

rLastScolar = rScolar;

z = Sum(r, MultVectorOnT(z, b));

if (min > Coud || !min)

{

min = Coud;

Best = x;

}

k++;

}

result = Up(U, D, Best);

break;

}

default:

break;

}

}

void LocalOptimalScheme(int metod)

{

NormF = Norm(f);

switch (metod)

{

case 0:

{

r = Residual(f, MultMatrixVector(result));

z = r;

p = MultMatrixVector(z);

k = 0;

T ScolarP = Scolar(p);

T ScolarR = Scolar(r);

//while (checkEnd(ScolarR, ScolarP, ScolarR))

while (checkEnd(r))

{

if (!ScolarP)

break;

a = Scolar(p, r) / ScolarP;

result = Sum(result, MultVectorOnT(z, a));

r = Residual(r, MultVectorOnT(p, a));

T gg = Scolar(r);

std::vector<T> d = MultMatrixVector(r);

b = -1.0 \* Scolar(p, d) / ScolarP;

z = Sum(r, MultVectorOnT(z, b));

p = Sum(d, MultVectorOnT(p, b));

ScolarP = Scolar(p);

k++;

}

break;

}

//DLLT

case 1:

//LLT

case 2:

{

r = Down(L, D, Residual(f, MultMatrixVector(result)));

z = Up(L, D, r);

p = Down(L, D, MultMatrixVector(z));

k = 0;

while (checkEnd(r))

{

T s = Scolar(p);

if (!s)

break;

a = Scolar(p, r) / s;

result = Sum(result, MultVectorOnT(z, a));

r = Residual(r, MultVectorOnT(p, a));

std::vector<T> d = Up(L, D, r);

d = MultMatrixVector(d);

d = Down(L, D, d);

b = -1.0 \* Scolar(p, d) / s;

z = Sum(Up(L, D, r), MultVectorOnT(z, b));

p = Sum(d, MultVectorOnT(p, b));

k++;

}

break;

}

//LU

case 3:

{

r = Down(L, Residual(f, MultMatrixVector(result)));

z = Up(U, D, r);

p = Down(L, MultMatrixVector(z));

k = 0;

while (checkEnd(r))

{

T s = Scolar(p);

if (!s)

break;

a = Scolar(p, r) / s;

result = Sum(result, MultVectorOnT(z, a));

r = Residual(r, MultVectorOnT(p, a));

std::vector<T> d = Up(U, D, r);

d = MultMatrixVector(d);

d = Down(L, d);

b = -1.0 \* Scolar(p, d) / s;

z = Sum(Up(U, D, r), MultVectorOnT(z, b));

p = Sum(d, MultVectorOnT(p, b));

k++;

}

break;

}

default:

break;

}

}

private:

std::vector<T> f; // Массив правой части

std::vector<T> di; //Диагональные элементы матрицы А

std::vector<T> ggu; //Верхний треугольник матрицы А в разреженном формате

std::vector<T> ggl; //Нижний треугольний матрицы А в разреженном формате

std::vector<T> L; // Матрица L

std::vector<T> U; // НЕ диагональыне элементы матрицы U

std::vector<T> D; // Диагональные элементы матрицы U

std::vector<size\_t> ig;// Массив индексов

std::vector<size\_t> jg; // Другой массив индексов

std::vector<T> result; //Результат x

std::vector<T> r, rLast, z, p, x, xbest;

size\_t N, M, Maxiter, k;

T e, a, b, NormF;

//Проверка окончания по невязки и по maxiter

///<param name = "ScolarR"> Входной параметр. Сколяр вектора R</param>

///<param name = "Return"> Выходной параметр. Невязка</param>

//Проверка конца у МСГ

bool checkEnd(T ScolarR, T &Return)

{

if (k == Maxiter)

return false;

Return = sqrt(ScolarR) / NormF;

if (e > Return)

return false;

return true;

}

///<param name = "ScolarR"> Входной параметр. Сколяр вектора R</param>

///<param name = "ScolarP"> Входной параметр. Сколяр вектора P</param>

///<param name = "ReturnR"> Выходной параметр. Сколяр вектора R</param>

//Проверка конца у ЛОС

bool checkEnd(T ScolarR, T ScolarP, T &ReturnR)

{

if (k == Maxiter)

return false;

T R = sqrt(ScolarR) / NormF;

T g = Scolar(r);

if (e > R)

return false;

a = Scolar(p, r) / Scolar(p);

T h = pow(a, 2) \* ScolarP;

ReturnR = ScolarR - h;

return true;

}

bool checkEnd(std::vector<T> r)

{

if (k == Maxiter)

return false;

if (e > Norm(r) / NormF)

return false;

return true;

}

//Прямой ход

std::vector<T> Down(std::vector<T> Matrix, std::vector<T> Diagonal, std::vector<T> R)

{

std::vector<T> x(N);

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

{

T sum = 0;

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

sum += Matrix[j] \* x[jg[j]];

x[i] = (R[i] - sum) / Diagonal[i];

}

return x;

}

std::vector<T> Down(std::vector<T> Matrix, std::vector<T> R)

{

std::vector<T> x(N);

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

{

T sum = 0;

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

sum += Matrix[j] \* x[jg[j]];

x[i] = R[i] - sum;

}

return x;

}

//Обратный ход

std::vector<T> Up(std::vector<T> Matrix, std::vector<T> Diagonal, std::vector<T> R)

{

for (int i = N - 1; i >= 0; i--)

{

R[i] /= Diagonal[i];

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

{

size\_t p = jg[j];

R[p] -= Matrix[j] \* R[i];

}

}

return R;

}

std::vector<T> Up(std::vector<T> Matrix, std::vector<T> R)

{

for (int i = N - 1; i >= 0; i--)

{

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

{

size\_t p = jg[j];

R[p] -= Matrix[j] \* R[i];

}

}

return R;

}

//Умноженик числа на вектор

std::vector<T> MultVectorOnT(std::vector<T> a, T b)

{

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

a[i] \*= b;

return a;

}

//Mult matrix and vector f

///<param name = "f"> Matrix be multtiplication on thix vector.</param>

std::vector<T> MultMatrixVector(std::vector<T> f)

{

std::vector<T> v(N);

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

{

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

{

v[i] += ggl[j] \* f[jg[j]];

v[jg[j]] += ggu[j] \* f[i];

}

v[i] += di[i] \* f[i];

}

return v;

}

std::vector<T> MultTMatrixVector(std::vector<T> f)

{

std::vector<T> v(N);

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

{

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

{

v[i] += ggu[j] \* f[jg[j]];

v[jg[j]] += ggl[j] \* f[i];

}

v[i] += di[i] \* f[i];

}

return v;

}

//Residual a - b

std::vector<T> Residual(std::vector<T> a, std::vector<T> b)

{

std::vector<T> v;

v.resize(a.size());

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

v[i] = a[i] - b[i];

return v;

}

//Sum a + b

std::vector<T> Sum(std::vector<T> a, std::vector<T> b)

{

std::vector<T> v;

v.resize(a.size());

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

{

v[i] = a[i] + b[i];

}

return v;

}

//Норма вектора а

inline T Norm(std::vector<T> a)

{

return sqrt(Scolar(a));

}

//Скалярное произведение 1 элемента

inline T Scolar(std::vector<T> a)

{

T sum = 0;

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

sum += pow(a[i], 2);

return sum;

}

//Скалярное произведение 2 элемента

inline T Scolar(std::vector<T> a, std::vector<T> b)

{

T sum = 0;

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

sum += a[i] \* b[i];

return sum;

}

//factorizationLU is a method in the Matrix class. МБ даже работает

///<param name = "h"> if this param true, then LU, else diagonal LU </param>

void factorizationLU(bool h)

{

if (h)

{

L.resize(ggl.size());

D.resize(di.size());

U.resize(ggu.size());

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

{

T sum = 0;

for (size\_t j = ig[i]; j < ig[i + 1]; j++)

{

T sum1 = 0, sum2 = 0;

int jj = jg[j];

for (size\_t k = ig[i], k2 = ig[jj]; k < j && k2 < ig[jj + 1];)

{

int p1 = jg[k];

int p2 = jg[k2];

if (p1 == p2)

{

sum1 += L[k] \* U[k2];

sum2 += L[k2] \* U[k];

k++; k2++;

}

else

{

if (p1 < p2)

k++;

else

k2++;

}

}

L[j] = (ggl[j] - sum1) / D[jg[j]];

U[j] = ggu[j] - sum2;

sum += L[j] \* U[j];

}

D[i] = di[i] - sum;

}

}

else

{

L.resize(ggl.size());

U.resize(ggu.size());

D = di;

}

}

//factorizationLU is a method in the Matrix class. МБ даже работает

///<param name = "h"> if this param true, then Holisskigo, else diagonal Holisskigo </param>

void factorizationLLT()

{

L.resize(ggl.size());

D.resize(N);

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

D[i] = sqrt(di[i]);

}

};

}

}