Министерство образования и науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №4

по дисциплине

«Уравнения математической физики»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Преподаватели Патрушев И. И.

Вариант 6

Новосибирск 2019

1. Цель работы

Изучить особенности реализации итерационных методов BCG, BCGStab, GMRES, для СЛАУ с несимметричными разряженными матрицами. Исследовать влияние предобусловливания на сходимость этих методов.

1. Задание

На начально этапе выберем начальное приближение

*На k-ой итерации метода*

*После итерационного процесса*

*.*

Остановка может происходить по числу итераций, по невязке, по отличию приближений и так далее. Поскольку метод является неустойчивым, то при его использовании дополнительно следует ограничивать сверху число итераций.

1. Сравнение LOS, LU и BCG

*800 конечный элементов*

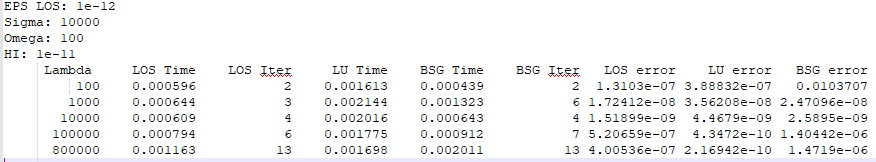
Область x [0, 20] y [0,20]

Искомая функция:

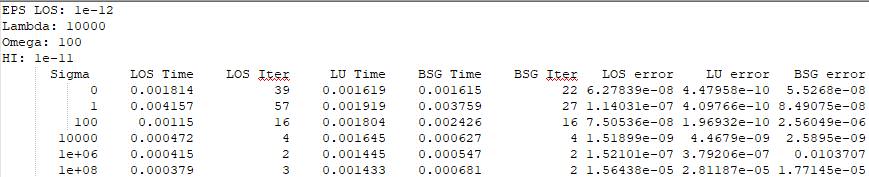
Уравнение:

Краевые условия:

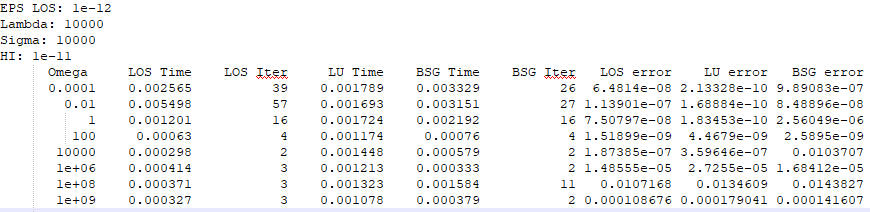
1. *Зависимость от лямбда*



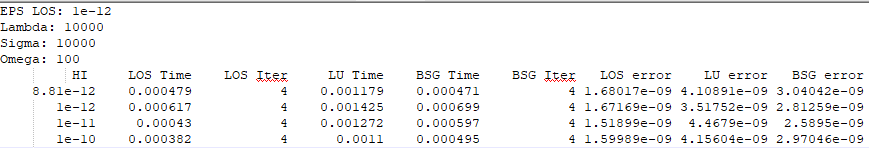
1. *Зависимость от сигмы*



1. *Зависимость от омеги*



1. *Зависимость от Хи*



*20000 конечный элементов*

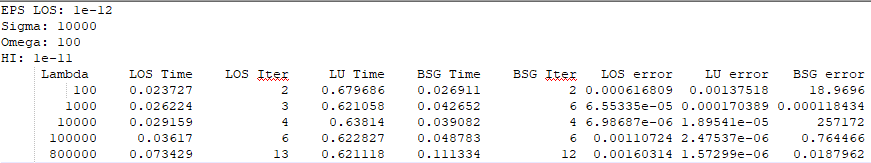
Область x [0, 100] y [0,100]

Искомая функция:

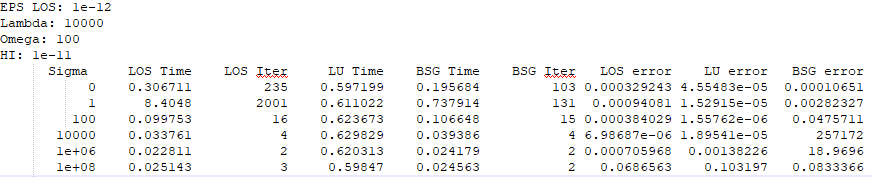
Уравнение:

Краевые условия:

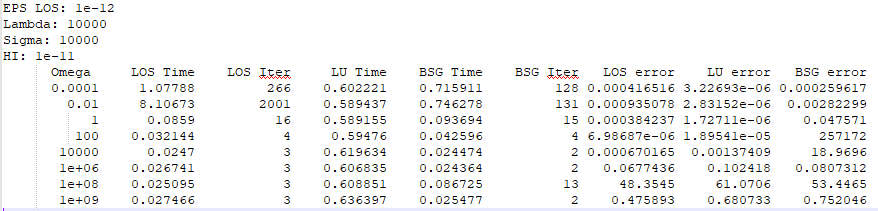
1. *Зависимость от лямбда*



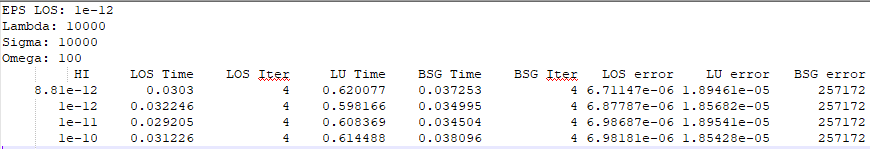
1. *Зависимость от сигмы*



1. *Зависимость от омеги*



1. *Зависимость от Хи*



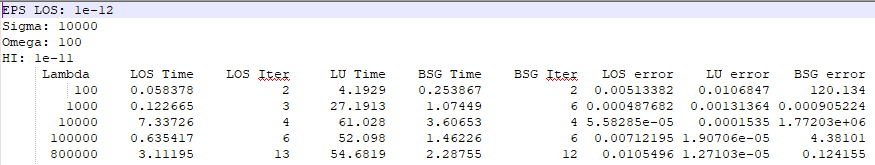
*45000 конечный элементов*

Область x [0, 150] y [0,150]

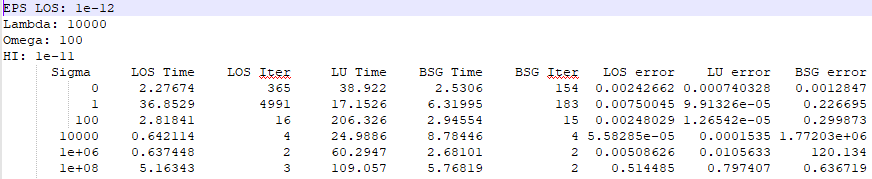
Искомая функция:

Уравнение:

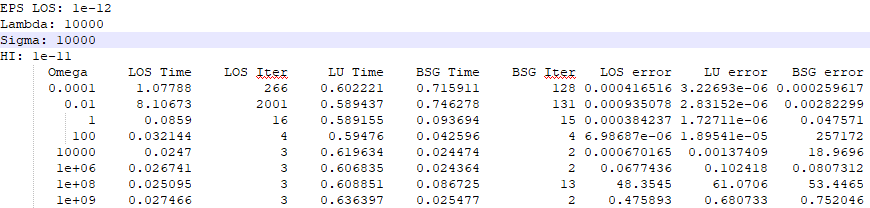
Краевые условия:

1) Зависимость от лямбда

2) Зависимость от сигмы



3) Зависимость от омеги



1. Текст программы

#/\*LDU.dll\*/

// 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]);

}

};

}

}

*/\*Includes.h\*/*

#pragma once

#include <iostream>

#include <vector>

#include <functional>

#include <sstream>

#include <iomanip>

#include <Windows.h>

#include <math.h>

using namespace std;

*/\*FEM.h\*/*

#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;

}

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;

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

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

//Задание первой краевой функции

Border(function<double(double, double)> &fs, function<double(double, double)> &fc, TypeBorder t) : Us(fs), Uc(fc)

{

type = t;

}

};

#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> localFs(pow(basis.order + 1, 2));

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

double hx, hy;

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

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

lastIter = 0;

ClearLastIter();

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

{

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

{

localFs[0] = fs(x[i], y[j]);

localFs[1] = fs(x[i + 1], y[j]);

localFs[2] = fs(x[i], y[j + 1]);

localFs[3] = fs(x[i + 1], y[j + 1]);

localFc[0] = fc(x[i], y[j]);

localFc[1] = fc(x[i + 1], y[j]);

localFc[2] = fc(x[i], y[j + 1]);

localFc[3] = fc(x[i + 1], y[j + 1]);

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

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

P = Gx \* (hy / hx) + Gy \* (hx / hy) - M \* (pow(omega, 2) \* hi \* hx \* hy);

C = M \* (omega \* sigma \* hx \* hy);

CreateLocalA();

localbc = (M \*(hx \* hy)) \* localFc;

localbs = (M \*(hx \* hy)) \* localFs;

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

{

blocal[2 \* i] = localbs[i];

blocal[2 \* i + 1] = localbc[i];

}

AddLocalMatrix(j \* index.ixMax + i, (j+1)\*index.ixMax + i);

}

}

AddBorderInMatrix();

q1.resize(q.size());

q2.resize(q.size());

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();

di.clear();

ggu.clear();

ggl.clear();

ig.clear();

jg.clear();

FULL2Prof();

t1 = high\_resolution\_clock::now();

ldu(ggl, ggu, di, ig, b, q2);

t2 = high\_resolution\_clock::now();

tq2 = duration\_cast<microseconds>(t2 - t1).count();

di.clear();

ggu.clear();

ggl.clear();

ig.clear();

jg.clear();

Prof2Raz();

t1 = high\_resolution\_clock::now();

lastIter2 = BSG\_STAB(ig, jg, ggl, ggu, di, b, q, eps);

t2 = high\_resolution\_clock::now();

tq3 = duration\_cast<microseconds>(t2 - t1).count();

di.clear();

ggu.clear();

ggl.clear();

ig.clear();

jg.clear();

//gause(A, b, q);

#ifdef \_DEBUG

system("cls");

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

{

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

{

cout << setw(10) << A[i][j] << " ";

}

cout << "\t\t\t" << b[i] << endl;

}

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

{

cout << q[i] << " "<< q1[i] << " "<< q2[i] << endl;

}

#endif // \_DEBUG

//т.к. базис лагранджа то u = q

for (size\_t i = 0; i < q.size() / 2; i++)

{

us[i] = q[2 \*i];

uc[i] = q[2 \*i + 1];

}

}

//Высвобождение памяти //кототрого нет!!!

void Clear()

{

}

void GetResult(vector<double> &Us, vector<double> &Uc)

{

Us = us;

Uc = uc;

}

#pragma region SetFunctions

//Задать базис

void SetBasis(Basis b)

{

basis = b;

}

//Задать Сигму

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

void SetOmega(double Omega) { omega = Omega; }

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

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

//Добавление стороны

void AddBorder(Border b)

{

border.push\_back(b);

}

//Дабавление лямбды

//Задание правой части

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

{

fc = func;

fs = funs;

}

//SetX

void SetX(vector<double> X)

{

x = X;

}

//SetY

void SetY(vector<double> Y)

{

y = Y;

}

#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;

vector<double> y;

double lambda; //Лямбда

double sigma; //Сигма

double omega; //Омега

double hi;

function<double(double, double)> fc;

function<double(double, double)> fs;

//Результат

vector<double> us; //Искомая функция

vector<double> uc; //Искомая функция

//Переходные вычисления

int m;

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

vector<double> b; //Вектора правой части

Matrix Alocal;

vector<double> blocal;

Matrix P;

Matrix C;

Matrix M;

Matrix Gx; //hy/hx

Matrix Gy; //hx/hy

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;

void Prof2Raz()

{

ig.push\_back(0);

for (size\_t i = 0, j1 = 0; i < m; i++)

{

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

{

if (A[i][j] || A[j][i])

{

ggu.push\_back(A[j][i]);

ggl.push\_back(A[i][j]);

jg.push\_back(j);

j1++;

}

}

ig.push\_back(j1);

di.push\_back(A[i][i]);

}

}

void FULL2Prof()

{

di.clear();

ggu.clear();

ggl.clear();

ig.clear();

jg.clear();

ig.push\_back(0);

for (size\_t i = 0, j1 = 0; i < m; i++)

{

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

{

if (A[i][j] || A[j][i] || b)

{

ggu.push\_back(A[j][i]);

ggl.push\_back(A[i][j]);

j1++;

b = 1;

}

}

ig.push\_back(j1);

di.push\_back(A[i][i]);

}

}

//Обработка введеных данных

void Preprocessing()

{

index.ixMax = x.size();

index.iyMax = y.size();

m = 2 \* index.ixMax \* index.iyMax;

A.resize(m);

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

A[i].resize(m);

b.resize(m);

q.resize(m);

int order = basis.order + 1;

Alocal.resize(2 \* pow(order, 2));

blocal.resize(2 \* pow(order, 2));

P.resize(pow(order, 2));

C.resize(pow(order, 2));

CreateMG();

us.resize(m / 2);

uc.resize(m / 2);

}

void CreateMG()

{

Gy.x = vector<vector<double>>{ {2, 1, -2, -1}, {1, 2, -1, -2}, {-2, -1, 2, 1}, {-1, -2, 1, 2} };

Gy = Gy \*(this->lambda / 6.0);

Gy.n = 4;

Gx.x = vector<vector<double>>{ {2, -2, 1, -1}, {-2, 2, -1, 1}, {1, -1, 2, -2}, {-1, 1, -2, 2} };

Gx = Gx \* (this->lambda / 6.0);

Gx.n = 4;

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

M = M / 36.0;

M.n = 4;

}

void CreateLocalA()

{

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

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

Alocal.x[2 \* i + 1][2 \* j + 1] = Alocal.x[2 \* i][2 \* j] = P.x[i][j];

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

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

{

Alocal.x[2 \* i][2 \* j + 1] = -C.x[i][j];

Alocal.x[2 \* i + 1][2 \* j] = C.x[i][j];

}

}

//Провкрка введенных данных

void checkdata()

{

}

//Добавление локальных матриц в глобальную

void AddLocalMatrix(int x0, int x1)

{

for (size\_t i = 0, il = 2 \* x0 \* basis.order; i < 2 \* (basis.order + 1); i++, il++)

{

for (size\_t j = 0, jl = 2 \* x0 \* basis.order; j < 2 \* (basis.order + 1); j++, jl++)

A[il][jl] += Alocal.x[i][j];

}

for (size\_t i = 0, il = 2 \* x0 \* basis.order; i < 2 \* (basis.order + 1); i++, il++)

{

for (size\_t j = 0, j2 = 2 \* (basis.order + 1), jl = 2 \* x1 \* basis.order; j < 2 \* (basis.order + 1); j++, jl++, j2++)

A[il][jl] += Alocal.x[i][j2];

}

for (size\_t i = 0, i1 = 2 \* (basis.order + 1), il = 2 \* x1 \* basis.order; i < 2 \* (basis.order + 1); i++, il++, i1++)

{

for (size\_t j = 0, jl = 2 \* x0 \* basis.order; j < 2 \* (basis.order + 1); j++, jl++)

A[il][jl] += Alocal.x[i1][j];

}

for (size\_t i = 0, i1 = 2 \* (basis.order + 1), il = 2 \* x1 \* basis.order; i < 2 \* (basis.order + 1); i++, il++, i1++)

{

for (size\_t j = 0, j1 = 2 \* (basis.order + 1), jl = 2 \* x1 \* basis.order; j < 2 \* (basis.order + 1); j++, jl++, j1++)

A[il][jl] += Alocal.x[i1][j1];

}

for (size\_t i = 0, il = 2 \* x0 \* basis.order; i < 2 \* (basis.order + 1); i++, il++)

b[il] += blocal[i];

for (size\_t i = 0, i1 = 2 \* (basis.order + 1), il = 2 \* x1 \* basis.order; i < 2 \* (basis.order + 1); i++, i1++, il++)

b[il] += blocal[i1];

}

//Очистка A и b c предыдущей итерации

void ClearLastIter()

{

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

{

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

A[i][j] = 0;

b[i] = 0;

}

}

//Добавление краевых условаий в матрицу А и b

void AddBorderInMatrix()

{

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

{

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

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[0].Us(x[0], y[i]);

p++;

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[0].Uc(x[0], y[i]);

}

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

{

int p = 2 \* i;

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[1].Us(x[i], y[0]);

p++;

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[1].Uc(x[i], y[0]);

}

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

{

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

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[2].Us(x[index.ixMax - 1], y[i]);

p++;

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[2].Uc(x[index.ixMax - 1], y[i]);

}

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

{

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

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[3].Us(x[i], y[index.iyMax - 1]);

p++;

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

A[p][j] = 0;

A[p][p] = 1;

b[p] = border[3].Uc(x[i], y[index.iyMax - 1]);

}

}

//Норма

double Norm(vector<double> b)

{

double sum = 0;

for (auto &i : b)

sum += pow(i, 2);

return sqrt(sum);

}

};

*/\*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 {0, 1, 2, 3, 4, 5}

#define Y {0, 1, 2, 3, 4, 5}

//Значение констант

#define lambda 1000 //Лябда

#define sigma 1000 //Функция сигма от x t и dudx

#define omega 10

#define hi 1e-2

//U функция от x y, t

//#define Us pow(x,3) \* y

//#define Uc pow(y,3) \* x

//

//#define DivGrads 6 \* x \* y

//#define DivGradc 6 \* y \* x

#define Us pow(x,3)\*y

#define Uc pow(y,3)\*x

#define DivGrads 6 \* x \* y

#define DivGradc 6 \* y \* x

//#define Us sin(x)

//#define Uc cos(y)

//#define DivGrads -sin(x)

//#define DivGradc -cos(y)

#define Check

#define CheckLambda

#define CheckSigma

#define CheckOmega

#define CheckHI

#define CheckEPS

//#define Print

int main()

{

FEM m;

vector<double> x;// = X;

vector<double> y;// = Y;

for (size\_t i = 0; i < 20; i++)

{

x.push\_back(i);

y.push\_back(i);

}

#pragma region Функции

//Искомая функция

function<double(double, double)> us = [](double x, double y) { return Us; };

function<double(double, double)> uc = [](double x, double y) { return Uc; };

function<double(double, double)> divgradS = [](double x, double y) {return DivGrads; };

function<double(double, double)> divgradC = [](double x, double y) {return DivGradc; };

#pragma endregion

#pragma region Задаем базисы

FEM::Basis basis;

basis.type = basis.Lagrange;

basis.order = 1;

m.SetBasis(basis);

#pragma endregion

#pragma region Задаем константы

int NX = x.size();

int NY = y.size();

m.SetLambda(lambda);

m.SetSigma(sigma);

m.SetOmega(omega);

m.SetHi(hi);

m.eps = EPS;

m.maxIter = MAXITER;

#pragma endregion

#pragma region Задаем краевые условия

function<double(double, double)> f1s = [&](double z, double y) { return us(x[0], y); };

function<double(double, double)> f1c = [&](double z, double y) { return uc(x[0], y); };

function<double(double, double)> f2s = [&](double x, double z) { return us(x, y[0]); };

function<double(double, double)> f2c = [&](double x, double z) { return uc(x, y[0]); };

function<double(double, double)> f3s = [&](double z, double y) { return us(x[NX-1], y); };

function<double(double, double)> f3c = [&](double z, double y) { return uc(x[NX-1], y); };

function<double(double, double)> f4s = [&](double x, double z) { return us(x, y[NY-1]); };

function<double(double, double)> f4c = [&](double x, double z) { return uc(x, y[NY-1]); };

FEM::Border b1(f1s, f1c, FEM::Border::First),

b2(f2s, f2c, FEM::Border::First),

b3(f3s, f3c, FEM::Border::First),

b4(f4s, f4c, FEM::Border::First);

m.AddBorder(b1);

m.AddBorder(b2);

m.AddBorder(b3);

m.AddBorder(b4);

#pragma endregion

#pragma region Задаем форму

m.SetX(x);

m.SetY(y);

#pragma endregion

#pragma region Задаем правую часть уравнения

function<double(double, double)> Fs = [&](double x, double y) {return lambda \* -1 \* divgradS(x, y) - omega \* sigma \* uc(x, y) - pow(omega, 2) \* hi \* us(x, y); };

function<double(double, double)> Fc = [&](double x, double y) {return lambda \* -1 \* divgradC(x, y) + omega \* sigma \* us(x, y) - pow(omega, 2) \* hi \* uc(x, y); };

m.SetF(Fc, Fs);

#pragma endregion

#pragma region Вывод

#ifdef Print

int pid = 12;

m.Start();

vector<double> ResultU;

vector<double> ResultC;

m.GetResult(ResultU, ResultC);

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

cout << setw(pid) << " ";

cout << setw(pid) << "Us";

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

cout << setw(pid) << " ";

cout << setw(pid) << "Uc" << endl;

for (size\_t i = 0, i1 = 0, i2 = 0; i < y.size(); i++)

{

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

{

cout << setw(pid) << ResultU[i1] << " ";

}

cout << "\t\t";

for (size\_t j = 0; j < x.size(); j++, i2++)

{

cout << setw(pid) << ResultC[i2] << " ";

}

cout << endl;

}

cout << endl;

cout << "Погрешность" << endl;

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

cout << setw(pid) << " ";

cout << setw(pid) << "Us";

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

cout << setw(pid) << " ";

cout << setw(pid) << "Uc" << endl;

double sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 0; i < y.size(); i++)

{

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

{

double p = us(x[j], y[i]) - ResultU[i1];

cout << setw(pid) << p << " ";

sum += pow(p, 2);

}

cout << "\t\t";

for (size\_t j = 0; j < x.size(); j++, i2++)

{

double p = uc(x[j], y[i]) - ResultC[i2];

cout << setw(pid) << p << " ";

sum += pow(p, 2);

}

cout << endl;

}

cout << "Норма вектора погрешности: " << sqrt(sum) << endl;

#endif // Print

#ifdef Check

//ofstream out\_t("mini2.txt");

//out\_t << setw(12) << "EPS" << setw(12) << "Labmda" << setw(12) << "Sigma" << setw(12) << "Omega" << setw(12) << "HI" << setw(12) << " MSG: " << setw(12) << "TIMELOS" << setw(12) << " IterLOS"<< setw(12) << " LU:" << setw(12) << "TIMELU" << endl;

function<double(double, double)> Fs1;

function<double(double, double)> Fc1;

int pid = 12;

#ifdef CheckLambda

{

double eps = 1e-12, Sigma = 1e4, Omega = 1e2, HI = 1e-11;

vector<double> Lambda = { 1e2, 1e3, 1e4, 1e5, 8e5 };

double sum = 0;

double LOSError = 0;

double LUError = 0;

double BSGError = 0;

ofstream out\_t("lambda.txt");

out\_t << "EPS LOS: " << eps << endl;

out\_t << "Sigma: " << Sigma << endl;

out\_t << "Omega: " << Omega << endl;

out\_t << "HI: " << HI << endl;

out\_t << setw(pid) << "Lambda " << setw(pid) << "LOS Time" << setw(pid) << "LOS Iter" << setw(pid) <<

"LU Time" << setw(pid) << "BSG Time" << setw(pid) << "BSG Iter" << setw(pid) << "LOS error" << setw(pid) << "LU error" << setw(pid) << "BSG error"<< endl;

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

{

Fs1 = [&](double x, double y) {return Lambda[i] \* -1 \* divgradS(x, y) - Omega \* Sigma \* uc(x, y) - pow(Omega, 2) \* HI \* us(x, y); };

Fc1 = [&](double x, double y) {return Lambda[i] \* -1 \* divgradC(x, y) + Omega \* Sigma \* us(x, y) - pow(Omega, 2) \* HI \* uc(x, y); };

m.SetF(Fc1, Fs1);

m.SetLambda(Lambda[i]);

m.SetSigma(Sigma);

m.SetOmega(Omega);

m.SetHi(HI);

m.eps = eps;

m.Start();

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q1[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2+=2)

sum += pow(uc(x[j], y[i]) - m.q1[i2], 2);

}

LOSError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q2[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2+=2)

sum += pow(uc(x[j], y[i]) - m.q2[i2], 2);

}

LUError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q[i2], 2);

}

BSGError = sqrt(sum);

sum = 0;

out\_t << setw(pid) << Lambda[i] << setw(pid) << m.tq1 / 1000000.0 << setw(pid) << m.lastIter << setw(pid) << m.tq2 / 1000000.0 <<

setw(pid) << m.tq3 / 1000000.0 << setw(pid) << m.lastIter2 << setw(pid) << LOSError << setw(pid) << LUError << setw(pid) << BSGError << endl;

}

out\_t.close();

}

#endif

#ifdef CheckSigma

{

double eps = 1e-12, Lambda = 1e4, Omega = 1e2, HI = 1e-11;

vector<double> Sigma = { 0, 1, 1e2, 1e4, 1e6, 1e8 };

double sum = 0;

double LOSError = 0;

double LUError = 0;

double BSGError = 0;

ofstream out\_t("sigma.txt");

out\_t << "EPS LOS: " << eps << endl;

out\_t << "Lambda: " << Lambda << endl;

out\_t << "Omega: " << Omega << endl;

out\_t << "HI: " << HI << endl;

out\_t << setw(pid) << "Sigma " << setw(pid) << "LOS Time" << setw(pid) << "LOS Iter" << setw(pid) <<

"LU Time" << setw(pid) << "BSG Time" << setw(pid) << "BSG Iter" << setw(pid) << "LOS error" << setw(pid) << "LU error" << setw(pid) << "BSG error" << endl;

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

{

Fs1 = [&](double x, double y) {return Lambda \* -1 \* divgradS(x, y) - Omega \* Sigma[i] \* uc(x, y) - pow(Omega, 2) \* HI \* us(x, y); };

Fc1 = [&](double x, double y) {return Lambda \* -1 \* divgradC(x, y) + Omega \* Sigma[i] \* us(x, y) - pow(Omega, 2) \* HI \* uc(x, y); };

m.SetF(Fc1, Fs1);

m.SetLambda(Lambda);

m.SetSigma(Sigma[i]);

m.SetOmega(Omega);

m.SetHi(HI);

m.eps = eps;

m.Start();

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q1[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q1[i2], 2);

}

LOSError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q2[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q2[i2], 2);

}

LUError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q[i2], 2);

}

BSGError = sqrt(sum);

sum = 0;

out\_t << setw(pid) << Sigma[i] << setw(pid) << m.tq1 / 1000000.0 << setw(pid) << m.lastIter << setw(pid) << m.tq2 / 1000000.0 <<

setw(pid) << m.tq3 / 1000000.0 << setw(pid) << m.lastIter2 << setw(pid) << LOSError << setw(pid) << LUError << setw(pid) << BSGError << endl;

}

out\_t.close();

}

#endif

#ifdef CheckOmega

{

double eps = 1e-12, Lambda = 1e4, Sigma = 1e4, HI = 1e-11;

vector<double> Omega = { 1e-4, 1e-2, 1, 1e2, 1e4, 1e6, 1e8, 1e9 };

double sum = 0;

double LOSError = 0;

double LUError = 0;

double BSGError = 0;

ofstream out\_t("omega.txt");

out\_t << "EPS LOS: " << eps << endl;

out\_t << "Lambda: " << Lambda << endl;

out\_t << "Sigma: " << Sigma << endl;

out\_t << "HI: " << HI << endl;

out\_t << setw(pid) << "Omega " << setw(pid) << "LOS Time" << setw(pid) << "LOS Iter" << setw(pid) <<

"LU Time" << setw(pid) << "BSG Time" << setw(pid) << "BSG Iter" << setw(pid) << "LOS error" << setw(pid) << "LU error" << setw(pid) << "BSG error" << endl;

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

{

Fs1 = [&](double x, double y) {return Lambda \* -1 \* divgradS(x, y) - Omega[i] \* Sigma \* uc(x, y) - pow(Omega[i], 2) \* HI \* us(x, y); };

Fc1 = [&](double x, double y) {return Lambda \* -1 \* divgradC(x, y) + Omega[i] \* Sigma \* us(x, y) - pow(Omega[i], 2) \* HI \* uc(x, y); };

m.SetF(Fc1, Fs1);

m.SetLambda(Lambda);

m.SetSigma(Sigma);

m.SetOmega(Omega[i]);

m.SetHi(HI);

m.eps = eps;

m.Start();

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q1[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q1[i2], 2);

}

LOSError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q2[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q2[i2], 2);

}

LUError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q[i2], 2);

}

BSGError = sqrt(sum);

sum = 0;

out\_t << setw(pid) << Omega[i] << setw(pid) << m.tq1 / 1000000.0 << setw(pid) << m.lastIter << setw(pid) << m.tq2 / 1000000.0 <<

setw(pid) << m.tq3 / 1000000.0 << setw(pid) << m.lastIter2 << setw(pid) << LOSError << setw(pid) << LUError << setw(pid) << BSGError << endl;

}

out\_t.close();

}

#endif

#ifdef CheckHI

{

double eps = 1e-12, Lambda = 1e4, Sigma = 1e4, Omega = 1e2;

vector<double> HI = { 8.81e-12, 1e-12, 1e-11, 1e-10 };

double sum = 0;

double LOSError = 0;

double LUError = 0;

double BSGError = 0;

ofstream out\_t("HI.txt");

out\_t << "EPS LOS: " << eps << endl;

out\_t << "Lambda: " << Lambda << endl;

out\_t << "Sigma: " << Sigma << endl;

out\_t << "Omega: " << Omega << endl;

out\_t << setw(pid) << "HI " << setw(pid) << "LOS Time" << setw(pid) << "LOS Iter" << setw(pid) <<

"LU Time" << setw(pid) << "BSG Time" << setw(pid) << "BSG Iter" << setw(pid) << "LOS error" << setw(pid) << "LU error" << setw(pid) << "BSG error" << endl;

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

{

Fs1 = [&](double x, double y) {return Lambda \* -1 \* divgradS(x, y) - Omega \* Sigma \* uc(x, y) - pow(Omega, 2) \* HI[i] \* us(x, y); };

Fc1 = [&](double x, double y) {return Lambda \* -1 \* divgradC(x, y) + Omega \* Sigma \* us(x, y) - pow(Omega, 2) \* HI[i] \* uc(x, y); };

m.SetF(Fc1, Fs1);

m.SetLambda(Lambda);

m.SetSigma(Sigma);

m.SetOmega(Omega);

m.SetHi(HI[i]);

m.eps = eps;

m.Start();

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q1[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q1[i2], 2);

}

LOSError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q2[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q2[i2], 2);

}

LUError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q[i2], 2);

}

BSGError = sqrt(sum);

sum = 0;

out\_t << setw(pid) << HI[i] << setw(pid) << m.tq1 / 1000000.0 << setw(pid) << m.lastIter << setw(pid) << m.tq2 / 1000000.0 <<

setw(pid) << m.tq3 / 1000000.0 << setw(pid) << m.lastIter2 << setw(pid) << LOSError << setw(pid) << LUError << setw(pid) << BSGError << endl;

}

out\_t.close();

}

#endif

#ifdef CheckEPS

{

double Lambda = 1e4, Sigma = 1e4, Omega = 1e2, HI = 1e-11;

vector<double> eps = { 1e-10, 1e-11, 1e-12, 1e-13, 1e-14 };

double sum = 0;

double LOSError = 0;

double LUError = 0;

double BSGError = 0;

ofstream out\_t("eps.txt");

out\_t << "Lambda: " << Lambda << endl;

out\_t << "Sigma: " << Sigma << endl;

out\_t << "Omega: " << Omega << endl;

out\_t << "HI: " << HI << endl;

out\_t << setw(pid) << "eps " << setw(pid) << "LOS Time" << setw(pid) << "LOS Iter" << setw(pid) <<

"LU Time" << setw(pid) << "BSG Time" << setw(pid) << "BSG Iter" << setw(pid) << "LOS error" << setw(pid) << "LU error" << setw(pid) << "BSG error" << endl;

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

{

Fs1 = [&](double x, double y) {return Lambda \* -1 \* divgradS(x, y) - Omega \* Sigma \* uc(x, y) - pow(Omega, 2) \* HI \* us(x, y); };

Fc1 = [&](double x, double y) {return Lambda \* -1 \* divgradC(x, y) + Omega \* Sigma \* us(x, y) - pow(Omega, 2) \* HI \* uc(x, y); };

m.SetF(Fc1, Fs1);

m.SetLambda(Lambda);

m.SetSigma(Sigma);

m.SetOmega(Omega);

m.SetHi(HI);

m.eps = eps[i];

m.Start();

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q1[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q1[i2], 2);

}

LOSError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q2[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q2[i2], 2);

}

LUError = sqrt(sum);

sum = 0;

for (size\_t i = 0, i1 = 0, i2 = 1; i < y.size(); i++)

{

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

sum += pow(us(x[j], y[i]) - m.q[i1], 2);

for (size\_t j = 0; j < x.size(); j++, i2 += 2)

sum += pow(uc(x[j], y[i]) - m.q[i2], 2);

}

BSGError = sqrt(sum);

sum = 0;

out\_t << setw(pid) << eps[i] << setw(pid) << m.tq1 / 1000000.0 << setw(pid) << m.lastIter << setw(pid) << m.tq2 / 1000000.0 <<

setw(pid) << m.tq3 / 1000000.0 << setw(pid) << m.lastIter2 << setw(pid) << LOSError << setw(pid) << LUError << setw(pid) << BSGError << endl;

}

out\_t.close();

}

#endif

#endif // Check

#pragma endregion

system("pause");

return 0;

}