|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
| Практическое задание | | | |
| по дисциплине «Проектная деятельность» | | | |
| **Решение параболической краевой задачи МКЭ** | | | |
|  | | | |
|  | Группа | ПМ-83 |
| Студент | Леонович Дарьяна |
|  |  |
|  |  |
| Преподаватель | Персова М.Г., киселев Д.С. |
|  |  |
|  | | | |
| Новосибирск | | | |
| 2021 | | | |

1. **Постановка задачи**

Моделирование электромагнитного поля с помощью решения параболической начально-краевой задачи методом конечных элементов. Решать будем трехмерную векторную задачу в декартовой системе координат с использованием реберных базисных функций первого порядка на параллелепипедах и трехслойной неявной схемы для аппроксимации по времени.

1. **Получение уравнения для моделирования электромагнитного поля**

Электромагнитное поле в некоторой расчетной области может быть описано системой уравнений Максвелла (2.1)-(2.4). Подразумевается, что все коэффициенты постоянны и не зависят от времени и искомых величин.

 (2.1)

 (2.2)

 (2.3)

 (2.4)

где  – индукция магнитного поля,  – плотность сторонних токов,  – напряженность электрического поля,  – удельная проводимость среды,  – диэлектрическая проницаемость среды,  – магнитная проницаемость среды.

Один из подходов к моделированию электромагнитных полей – это решение векторного уравнения Гельмгольца, которое можно получить из первых двух уравнений системы Максвелла. Для этого возьмем производную по времени от уравнения (2.1) и подействуем оператором  на уравнение (2.2). Получим уравнения (2.5) и (2.6):

 (2.5)

 (2.6)

Подставляя уравнение (2.6) в (2.5) получим векторное уравнение Гельмгольца, описывающее поведение электромагнитного поля (2.7).

 (2.7)

 (2.8)

Чтобы избавиться от производной по времени в правой части, будем решать задачу через векторный потенциал магнитного поля. Для этого представим магнитную индукцию  в следующем виде

 (2.9)

где  – векторный потенциал магнитного поля.

Помимо  в уравнениях (2.1) и (1.2) присутствует напряженность электрического поля . Для того, чтобы определить  с помощью , подставим уравнение (2.9) в (2.2).

 (2.10)

Из уравнения (2.10) следует, что напряженность электрического поля есть производная по времени от векторного потенциала магнитного поля, взятая с обратным знаком:

 (2.11)

Подставляя уравнение (2.9) и (2.11) в (2.1), получим уравнение, описывающее поведение электромагнитного поля через векторный потенциал:

 (2.12)

 (2.13)

Так как эффект от компоненты со второй производной по времени считается несущественным в рассматриваемой задаче, считаем равным нулю. И получим параболическую задачу:

(2.14)

(2.15)

1. **Дискретизация по времени**

Выпишем задачу. Уравнение:

задано в области с краевым условием по границе S

и начальным условием:

Неявная схема аппроксимирует исходной уравнение по времени в следующем виде:

где j – номер текущего временного слоя.

Так как для аппроксимации по времени используем трехслойную схему, слои, к которым можем применить схему, начинаются со второго. Значение нулевого слоя задается из начального условия, значения на первом слое можно найти по двухслойной схеме.

В случае трехслойной схемы аппроксимируем по времени следующим образом:

где базисные функции имеют вид:

Тогда для аппроксимации производной по времени в точке t = tj вычислим эти производные для базисных функций:

Обозначим

Тогда аппроксимация производной по времени имеет вид:

Подставим в неявную схему:

Перенесем вправо известные значения на предыдущих слоях:

Так как значения на предыдущих слоях и величина шага по временной сетке известны, обозначим

Тогда в результате аппроксимации по времени на j-ом слое времени получим векторное уравнение:

1. **Вариационная постановка**

Запишем эквивалентную исходной задаче на j-ом временном слое вариационную формулировку в форме Галеркина.

Для этого обозначим невязку исходного уравнения:

Потребуем, чтобы невязка была ортогональна пространству пробных функций , т.е.

Преобразуем интеграл с использованием формулы Грина, распишем интеграл по границе с учетом краевых условий и, для исключения из суммы интеграла по S1, потребуем, чтобы , т.е. чтобы пробные функции были из пространства функций, имеющих суммируемые с квадратом производные, и равных нулю на границе S1. Решение задачи u будет принадлежать пространству . Перепишем получившееся выражение:

1. **Конечноэлементная дискретизация и переход к локальным матрицам**

Получим аппроксимацию уравнения Галеркина. Для этого возьмем конечноэлементные пространства , , которые аппроксимируют и соответственно. Для этого заменим  на аппроксимирующую *h* и на :

Пусть – базис *V rot* (пространство, аппроксимирующее *H rot*). Тогда и *h* может быть представлено в виде:

где – множество индексов i таких, что являются базисными функциями пространств , .

И уравнение Галеркина эквивалентно следующей СЛАУ для компонент вектора q с индексами :

Недостающие n-n0 уравнений для компонент вектора q с индексами могут быть получены из условия :

Так как мы решаем трехмерную задачу в декартовой системе координат с использованием реберных базисных функций первого порядка на параллелепипедах, ячейками дискретизации являются прямоугольники

Выпишем базисные функции. Для этого сперва построим одномерные линейные функции на :

И локальные базисные функции:

Компоненты локальных матриц жесткости и массы имеют вид:

При этом – осредненное значение

Таким образом аппроксимируется вектором G\*qj, аппроксимируется с помощью M\*qj.

Тогда вид уравнения на j-ом временном слое после конечноэлементной аппроксимации:

1. **Аналитические выражения для вычисления локальных матриц**

С учетом, что все подынтегральные функции представляются в виде произведения одномерных полиномов, интегралы по объему можно вычислить как произведение одномерных интегралов.

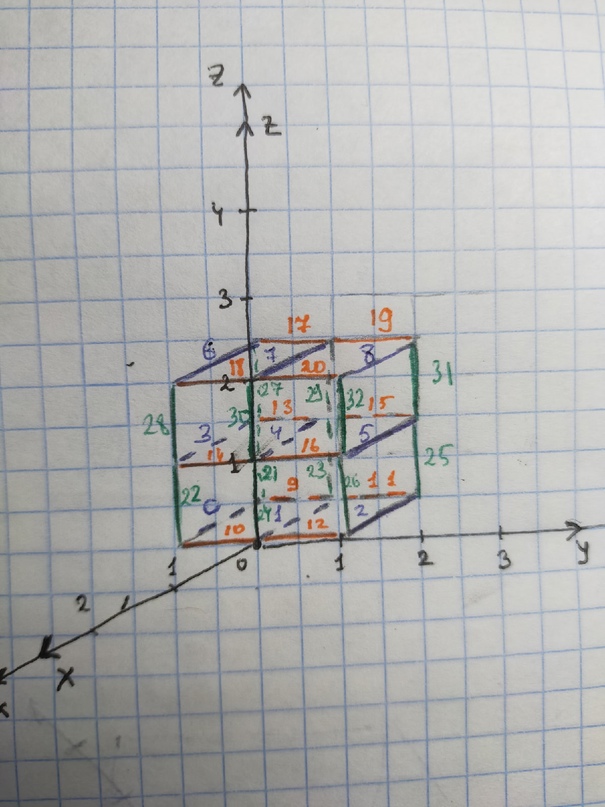
Запишем вид локальных матриц.

Матрица масс:

Матрица жесткости:

1. **Структуры данных для задания расчетной области и конечноэлементной сетки**

Рисунок расчетной области:



И в глобальной, и в локальной нумерации сперва нумеруются ребра, параллельные оси х (сначала увеличивается номер вдоль оси у, потом вдоль оси z), потом – параллельные у (сначала увеличивается номер вдоль оси х, потом вдоль оси z) и в конце ребра, параллельные оси z (сначала увеличивается номер вдоль оси х, потом вдоль оси у)

Сетка задается в следующем виде:

|  |  |  |
| --- | --- | --- |
| Имя файла | Описание | Пример |
| grid.txt | Используется для генерации сетки по пространству (для создания файлов info.txt, xyz.txt, S1.txt, elem.txt, material.txt). Содержит кол-во узлов по оси x, по оси у, по оси z; координату начала область (наименьший по x, y, z узел); координату следующего опорного узла по оси х, кол-во разбиений между опорными узлами, параметр релаксации (равен 1 для равномерных сеток); координату следующего опорного узла по оси у, кол-во разбиений между опорными узлами, параметр релаксации (равен 1 для равномерных сеток); координату следующего опорного узла по оси z, кол-во разбиений между опорными узлами, параметр релаксации (равен 1 для равномерных сеток); | 2 3 3  1 1 1  2 1 1  3 2 1  3 2 1 |
| time\_grid.txt | Используется для генерации сетки по времени (файла time.txt). Содержит кол-во узлов по времени, значение первой отметки времени, значение следующего опорного узла по времени, кол-во разбиений и коэффициент релаксации (равен 1 для равномерной сетки) | 5  0  4 4 1 |
| info.txt | Содержит четыре числа: кол-во узлов сетки, кол-во материалов в расчетной области, кол-во конечных элементов сетки, кол-во границ с краевыми условиями первого рода | 33 1 4 3 |
| xyz.txt | Содержит координаты всех ребер, . В программе создана структура edge, содержащая два элемента типа node, где хранятся координаты узлов ребра. Вектор all\_edges описывает все ребра сетки. Глобальный номер ребра соответствует индексу ребра в массиве. | 1 1 1 2 1 1  1 2 1 2 2 1  1 3 1 2 3 1  1 1 2 2 1 2  1 2 2 2 2 2  1 3 2 2 3 2  1 1 3 2 1 3  1 2 3 2 2 3  1 3 3 2 3 3  1 1 1 1 2 1  2 1 1 2 2 1  1 2 1 1 3 1  2 2 1 2 3 1  1 1 2 1 2 2  2 1 2 2 2 2  1 2 2 1 3 2  2 2 2 2 3 2  1 1 3 1 2 3  2 1 3 2 2 3  1 2 3 1 3 3  2 2 3 2 3 3  1 1 1 1 1 2  2 1 1 2 1 2  1 2 1 1 2 2  2 2 1 2 2 2  1 3 1 1 3 2  2 3 1 2 3 2  1 1 2 1 1 3  2 1 2 2 1 3  1 2 2 1 2 3  2 2 2 2 2 3  1 3 2 1 3 3  2 3 2 2 3 3 |
| S1.txt | Содержит записи вида: число ребер, где задано граничное условие, перечисление номеров ребер в глобальной нумерации.  Кол-во таких записей задается в info.txt. В программе хранится S1, вектор векторов с номерами ребер. i-ый вектор содержит номера ребер с i-ым краевым условием.  Отдельно задаются краевые для ребер, параллельных осям x, y и z | 8  6 0 7 1 8 2 3 5  12  17 9 18 10 19 11 20 12 13 14 15 16  12  21 25 22 26 27 31 28 32 23 24 29 30 |
| material.txt | Содержит описание материала: значение и значение для материала. В программе материалы хранятся в специальной структуре material, где хранится значения и | 1 1 |
| elem.txt | Содержит список глобальных номеров ребер в порядке локальной нумерации, индекс материала, индекс правой части F дифференциального уравнения (F задается в программе). Данные хранятся в специальной структуре element, вектор element описывает все элементы сетки. Глобальный номер элемента соответствует индексу элемента в массиве. | 0 1 3 4 9 10 13 14 21 22 23 24 0 0  1 2 4 5 11 12 15 16 23 24 25 26 0 0  3 4 6 7 13 14 17 18 27 28 29 30 0 0  4 5 7 8 15 16 19 20 29 30 31 32 0 0 |
| time.txt | Содержит количество узлов по сетке времени и сами значения этих узлов. В программе сетка времени хранится в векторе time | 5  0 1 2 3 4 |

1. **Структура основных модулей программы**

main.cpp – содержит все необходимые для построения СЛАУ функции, а так же глобальные переменные, описывающие сетку.

struct node // структура узла

{

double x;

double y;

double z;

};

struct edge // структура ребра

{

node \_1, \_2;

};

struct material // структура материала

{

double mu;

double sigma;

};

struct element // структура элемента

{

std::vector<int> edge\_loc;

int mater;

int f\_id;

};

void mult\_matrix\_vector(std::vector<int> &ia, std::vector<int> &ja, std::vector<double> &di, std::vector<double> &al, std::vector<double> &x, std::vector<double>& y); // умножение матрицы на вектор х, результат в у

void mult\_matrix\_double(std::vector<double>& di, std::vector<double>& al, double x); // умножение на месте матрицы на число

void mult\_vector\_double(std::vector<double>& a, double x); // умножение на месте вектора на число

void plus\_vector\_vector(std::vector<double>& a, std::vector<double>& b); // сложение векторов, в вектор а

void plus\_matrix\_matrix(std::vector<double>& diM, std::vector<double>& alM, std::vector<double>& diG, std::vector<double>& alG, std::vector<double>& di, std::vector<double>& al, double koef1); // A = koef1 \* M + G

class FEM

{

private:

std::vector<edge> all\_edges; // все ребра в порядке глобальной нумерации

std::vector<element> all\_elems; // все элементы в прядке глобальной нумерации

std::vector<material> all\_materials; // все материалы по индексам

std::vector<std::vector<int>> S1; // S1[i][j] на j-ом узле заданы i-ые краевые 1 рода

std::vector<std::vector<double>> q; // значения q на трех слоях: q0, q1 известны, q2 ищем

int i\_t = 0; // текущий временной слой

std::vector<double> time\_grid; // сетка по времени

// матрица A

std::vector<int> ia, ja; // портрет

std::vector<double> di, al, b; // симметричная

// М

std::vector<double> diM, alM;

// G

std::vector<double> diG, alG;

int N, Kel; // кол-во ребер и элементов

const double M = 1E+15; // большое число, используется для учета краевых первого рода

Matrix D, G1, G2, G3, GT3; // константные матрицы, используемые для сборки локальных

void Init\_const\_matrix(); // инициализация константных матриц

double func\_f(double x, double y, double z, int f\_id, int num); // значение f по индексу f\_id

double func\_S1(double x, double y, double z, int s1\_id); // значение краевого S1 по индексу s1\_id

double true\_func(double x, double y, double z, int num); // истинная функция (используется для определения погрешности решения)

int Input(); // чтение данных

int Loc\_matrix(int el\_id, std::vector<Matrix>& A\_loc, std::vector<double>& b\_loc); // сборка локальной матрицы A = M + G

int Loc\_matrix\_time(int el\_id, std::vector<Matrix>& M\_loc, std::vector<Matrix>& G\_loc); // сборка локальных матриц M и G

int Loc\_b\_time(int el\_id, std::vector<double>& b\_loc); // сборка локального вектора b

int GeneratePortrait(); // генерация портрета матрицы А

int AddLocal(int el\_id, std::vector<Matrix>& A\_loc); // внесение локальной матрицы в глобальную А

int AddLocalM(int el\_id, std::vector<Matrix>& M\_loc); // внесение локальной матрицы в глобальную M

int AddLocalG(int el\_id, std::vector<Matrix>& G\_loc); // внесение локальной матрицы в глобальную G

int AddLocal\_b(int el\_id, std::vector<double>& b\_loc); // внесение локального вектора правой части в глобальный

int SetS1(); // учет первых краевых условий

public:

int SolveTask(std::vector<double> &res); // решение стационарной задачи

int SolveTask\_time(); // решение нестационарной задачи

void ValueInPointXYZ(std::vector<double>& A, double x, double y, double z, int time\_id); // значение численно найденной функции в произвольной точке пространства в одном из трех последних временных узлов

};

Matrix.cpp, Matrix.h –используются для облегчения работы с матрицыми при сборке локальных матриц.

cgm.cpp, cgm.h содержат функции для решения СЛАУ методом МСГ с LLT-факторизацией.

Generate.h и Generate.cpp содержат функции для генерации сеток по времени (void Create\_time\_grid()) и пространству(void Make\_grid(std::string path)).

1. **Исследования на порядок аппроксимации по пространству**

Расчетная область и сетка:

* мю = сигма = 1; по всем границам краевые условия 1ого рода;
* начиная с точки (1,1,1) на каждой оси 10 узлов, расположенных равномерно, с шагом 0,5;
* сетка по времени из 7 временных узлов, от 0 до 3 с шагом 0,5.

Для исследований возьмем следующий набор точек:

(1.4 1.4 1.4), (1.4 2.4 1.4), (1.4 3.4 1.4), (2.4 1.4 1.4), (2.4 2.4 1.4), (2.4 3.4 1.4), (1.4 1.4 2.4), (1.4 2.4 2.4), (1.4 3.4 2.4), (2.4 1.4 2.4), (2.4 2.4 2.4), (2.4 3.4 2.4), (1.4 1.4 3.4), (1.4 2.4 3.4), (1.4 3.4 3.4), (2.4 1.4 3.4), (2.4 2.4 3.4), (2.4 3.4 3.4), (1.4 1.4 4.4), (1.4 2.4 4.4), (1.4 3.4 4.4), (2.4 1.4 4.4), (2.4 2.4 4.4), (2.4 3.4 4.4)

Для нахождения порядка аппроксимации по пространству будем увеличивать степень полинома по пространству и сравнивать среднеквадратичную погрешность на временных слоях, начиная со слоя t = 1, где начинается применение трехслойной неявной схемы. Чтобы исследовать только трехслойную схему, значения на предыдущих слоях зададим из входных данных. Чтобы не учитывать влияние аппроксимации значения функции на ребре, возьмем функции, являющиеся постоянными на ребрах (первая компонента не зависит от х, вторая – от у, третья – от z)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *среднеквадратичная погрешность* | | | | |
| *время:* | 1 | 1,5 | 2 | 2,5 | 3 |
| *A = {yt, zt, xt}* | 4,849E-11 | 9,032E-11 | 1,696E-10 | 2,426E-10 | 2,993E-10 |
| *A = {y2t, z2t, x2t}* | 6,469E-03 | 6,469E-03 | 6,469E-03 | 6,469E-03 | 6,469E-03 |

Так как базис по пространству используется трилинейный, увеличение степени полинома по пространственным координатам приводит к возникновению погрешности из-за аппроксимации по пространству. Так как на полиноме первой степени по пространству наблюдается только вычислительная погрешность, а на полиноме второй степени возникает уже значимая погрешность от аппроксимации пространства, порядок аппроксимации используемого базиса по пространству первый. Полученный порядок аппроксимации соответствует теоретически ожидаемому для линейного базиса.

1. **Исследования на порядок аппроксимации по времени**

Расчетная область и сетка:

* мю = сигма = 1; по всем границам краевые условия 1ого рода;
* начиная с точки (1,1,1) на каждой оси 10 узлов, расположенных равномерно, с шагом 0,5;
* сетка по времени из 7 временных узлов, от 0 до 6 с шагом 1.

Для исследований возьмем следующий набор точек:

(1.4 1.4 1.4), (1.4 2.4 1.4), (1.4 3.4 1.4), (2.4 1.4 1.4), (2.4 2.4 1.4), (2.4 3.4 1.4), (1.4 1.4 2.4), (1.4 2.4 2.4), (1.4 3.4 2.4), (2.4 1.4 2.4), (2.4 2.4 2.4), (2.4 3.4 2.4), (1.4 1.4 3.4), (1.4 2.4 3.4), (1.4 3.4 3.4), (2.4 1.4 3.4), (2.4 2.4 3.4), (2.4 3.4 3.4), (1.4 1.4 4.4), (1.4 2.4 4.4), (1.4 3.4 4.4), (2.4 1.4 4.4), (2.4 2.4 4.4), (2.4 3.4 4.4)

Для нахождения порядка аппроксимации по времени будем увеличивать степень полинома по времени и сравнивать среднеквадратичную погрешность на временных слоях, начиная со слоя t = 2, где начинается применение трехслойной неявной схемы. Чтобы исследовать только трехслойную схему, значения на предыдущих слоях зададим из входных данных. Чтобы не добавлять погрешность от аппроксимации по пространству, возьмем линейные по пространству функции.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *среднеквадратичная погрешность* | | | | |
| *время:* | 2 | 3 | 4 | 5 | 6 |
| *A = {yt, zt, xt}* | 1,568E-10 | 2,864E-10 | 3,678E-10 | 4,215E-10 | 4,588E-10 |
| *A = {yt2, zt2, xt2}* | 1,568E-10 | 2,430E-10 | 3,079E-10 | 3,570E-10 | 3,946E-10 |
| *A = {yt3, zt3, xt3}* | 9,235E-02 | 5,129E-02 | 2,810E-02 | 1,617E-02 | 9,840E-03 |

Так как базис по времени используется квадратичный, на полиноме первой и второй степени по времени наблюдается только вычислительная погрешность, а на полиноме третьей степени возникает уже значимая погрешность от аппроксимации по времени. Значит, порядок аппроксимации трехслойной неявной схемы второй. Полученный порядок аппроксимации соответствует теоретически ожидаемому для квадратичного базиса по времени.

1. **Исследования на порядок сходимости по пространству**

Расчетная область и сетка:

* мю = сигма = 1; по всем границам краевые условия 1ого рода;
* начиная с точки (1,1,1) на каждой оси 10 узлов, расположенных равномерно, с шагом 1 (с уменьшением шага последний узел по каждой оси расположен не ближе 4,5);
* сетка по времени из 7 временных узлов, от 0 до 3 с шагом 0.5.

Для исследований возьмем следующий набор точек:

(1.4 1.4 1.4), (1.4 2.4 1.4), (1.4 3.4 1.4), (2.4 1.4 1.4), (2.4 2.4 1.4), (2.4 3.4 1.4), (1.4 1.4 2.4), (1.4 2.4 2.4), (1.4 3.4 2.4), (2.4 1.4 2.4), (2.4 2.4 2.4), (2.4 3.4 2.4), (1.4 1.4 3.4), (1.4 2.4 3.4), (1.4 3.4 3.4), (2.4 1.4 3.4), (2.4 2.4 3.4), (2.4 3.4 3.4), (1.4 1.4 4.4), (1.4 2.4 4.4), (1.4 3.4 4.4), (2.4 1.4 4.4), (2.4 2.4 4.4), (2.4 3.4 4.4)

Сетку h/2 и h/4 получим, уменьшив шаг по переменным в 2 и 4 раза соответственно. Среднеквадратичную погрешность будем искать как:

Для определения порядка сходимости найдем значение отношения погрешностей

Возьмем *A = {cos(y), cos(z), cos(x)}* и будем уменьшать шаг по пространству для определения порядка сходимости базиса. Отношение погрешностей:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | среднее значение погрешности на сетке | ht к ht/2 | | ht/2 к ht/4 | | ht/4 к ht/8 | | ht/4 к ht/8 | |
| 2k | k | 2k | k | 2k | k | 2k | k |
| h | 1,443E-01 | 5,046 | 2,34 |  |  |  |  |  |  |
| h/2 | 2,858E-02 | 2,966 | 1,57 |  |  |  |  |
| h/4 | 9,638E-03 |  |  | 5,388 | 2,43 |  |  |
| h/8 | 1,789E-03 |  |  |  |  | 3,184 | 1,67 |
| h/16 | 5,618E-04 |  |  |  |  |  |  |

Так как отношение погрешностей при первом и втором дроблении не стабильно, для нахождения порядка сходимости раздробим сетку еще несколько раз. Хотя отношение погрешностей и не стало стабильным, оно периодично. Возможно, такая периодичность связана с выбором рассматриваемых точек и их местоположением в конечном элементе.

На основе полученных в ходе исследования результатов можно увидеть, что порядок сходимости для трилинейного базиса не ниже теоретически ожидаемого первого.

1. **Исследования на порядок сходимости по времени**

Расчетная область и сетка:

* мю = сигма = 1; по всем границам краевые условия 1ого рода;
* начиная с точки (1,1,1) на каждой оси 10 узлов, расположенных равномерно, с шагом 1;
* сетка по времени от 0 до 6 с шагом 0,25.

Для исследований возьмем следующий набор точек:

(1.4 1.4 1.4), (1.4 2.4 1.4), (1.4 3.4 1.4), (2.4 1.4 1.4), (2.4 2.4 1.4), (2.4 3.4 1.4), (1.4 1.4 2.4), (1.4 2.4 2.4), (1.4 3.4 2.4), (2.4 1.4 2.4), (2.4 2.4 2.4), (2.4 3.4 2.4), (1.4 1.4 3.4), (1.4 2.4 3.4), (1.4 3.4 3.4), (2.4 1.4 3.4), (2.4 2.4 3.4), (2.4 3.4 3.4), (1.4 1.4 4.4), (1.4 2.4 4.4), (1.4 3.4 4.4), (2.4 1.4 4.4), (2.4 2.4 4.4), (2.4 3.4 4.4)

По времени на всех сетках будем рассматривать решение для времен 1, 2, 3, 4, 5 и 6.

Сетку h/2 и h/4 получим, уменьшив шаг по времени в 2 и 4 раза соответственно. Среднеквадратичную погрешность будем искать как:

Для определения порядка сходимости найдем значение отношения погрешностей

Пусть *A = {yt3, zt3, xt3}.* Результаты работы программы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | среднее значение погрешности на сетке | ht к ht/2 | | ht/2 к ht/4 | |
| 2k | k | 2k | k |
| ht | 4,020E-03 | 3,840E+00 | 1,941010346 |  |  |
| ht/2 | 1,047E-03 | 3,933E+00 | 1,975473644 |
| ht/4 | 2,662E-04 |  |  |

Полученные значения подтверждают теоретическое предположение, что трехслойная неявная схема имеет второй порядок сходимости.

1. **Выводы**

Успешно реализовано решение векторной трехмерной начально-краевой параболической задачи в декартовой системе координат с использованием трехслойной неявной схемы для аппроксимации по времени.

На основе исследований экспериментальным путем удалось найти порядок аппроксимации и сходимости трехслойной неявной схемы и линейного на параллелепипедах базиса по пространству.

*Порядок аппроксимации трехслойной неявной схемы*

Для исследуемых функций погрешность аппроксимации по времени появляется на полиноме 3-ей степени по t. Следовательно, порядок аппроксимации трехслойной неявной схемы равен двум, что соответствует теоретическому предположению, так как базис по времени является квадратичным.

По пространству погрешность появляется на полиноме второй степени, что соответствует теоретическому предположению о том, что точно аппроксимироваться по пространству будут только линейные функции и базис имеет первый порядок аппроксимации.

*Порядок сходимости трехслойной неявной схемы*

Исследования на порядок аппроксимации по времени подтвердили теоретическое предположение о втором порядке сходимости, так как базис по времени квадратичный.

Полученный экспериментально порядок сходимости по пространству для линейного базиса не ниже теоретически ожидаемого первого.

1. **Текст программы**

**Source.cpp**

#include "Base.h"

#include "Generate.h"

int main()

{

Make\_grid("");

Create\_time\_grid();

FEM task;

std::vector<double> res;

task.SolveTask\_time();

//task.SolveTask(res);

return 0;

}

**Base.cpp**

#include "Base.h"

double FEM::func\_f(double x, double y, double z, int f\_id, int num) // значение f по индексу f\_id

{

double t = time\_grid[i\_t];

switch (f\_id)

{

case 0:

switch (num)

{

case 0: // F[0]

return 3 \* t \* y \* t

;

break;

case 1: // F[1]

return 3 \* t \* z \* t

;

break;

case 2: // F[2]

return 3 \* t \* x \* t

;

break;

default:

break;

}

default:

std::cout << "can't find f № " << f\_id << "\n";

break;

}

return 0;

}

double FEM::func\_S1(double x, double y, double z, int s1\_id) // значение краевого S1 по индексу f\_id

{

double t = time\_grid[i\_t];

switch (s1\_id)

{

case 0:

return y \* t \* t \* t

;

case 1:

return z \* t \* t \* t

;

case 2:

return x \* t \* t \* t

;

default:

std::cout << "can't find s1 № " << s1\_id << "\n";

break;

}

}

double FEM::true\_func(double x, double y, double z, int num)

{

double t = time\_grid[i\_t];

switch (num)

{

case 0:

return y \* t \* t \* t

;

case 1:

return z \* t \* t \* t

;

case 2:

return x \* t \* t \* t

;

default:

std::cout << "can't find true\_func № " << num << "\n";

break;

}

}

int FEM::Input() // чтение данных

{

int //N, // всего ребер

//Nx, Ny, Nz,

Nmat, // всего материалов

//Kel, // всего элементов

NS1; // всего 1ых краевых

std::ifstream in;

in.open("info.txt");

in >> N >> Nmat >> Kel >> NS1;

in.close();

in.open("xyz.txt");

all\_edges.resize(N);

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

{

in >> all\_edges[i].\_1.x >>

all\_edges[i].\_1.y >>

all\_edges[i].\_1.z >>

all\_edges[i].\_2.x >>

all\_edges[i].\_2.y >>

all\_edges[i].\_2.z;

}

in.close();

in.open("S1.txt");

S1.resize(NS1);

for (int i = 0; i < NS1; i++)

{

int size;

in >> size;

S1[i].resize(size);

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

{

in >> S1[i][j];

}

}

in.close();

in.open("material.txt");

all\_materials.resize(Nmat);

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

{

in >> all\_materials[i].mu >> all\_materials[i].sigma;

}

in.close();

in.open("elem.txt");

all\_elems.resize(Kel);

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

{

all\_elems[i].edge\_loc.resize(12);

in >> all\_elems[i].edge\_loc[0] >> all\_elems[i].edge\_loc[1]

>> all\_elems[i].edge\_loc[2] >> all\_elems[i].edge\_loc[3]

>> all\_elems[i].edge\_loc[4] >> all\_elems[i].edge\_loc[5]

>> all\_elems[i].edge\_loc[6] >> all\_elems[i].edge\_loc[7]

>> all\_elems[i].edge\_loc[8] >> all\_elems[i].edge\_loc[9]

>> all\_elems[i].edge\_loc[10] >> all\_elems[i].edge\_loc[11]

>> all\_elems[i].mater >> all\_elems[i].f\_id;

}

in.close();

q.resize(3);

q[0].resize(N);

q[1].resize(N);

q[2].resize(N);

in.open("time.txt");

int Nt;

in >> Nt;

time\_grid.resize(Nt);

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

{

in >> time\_grid[i];

}

// заполним 0 и 1 слои

for (int k = 0; k < 2; k++)

{

i\_t = k;

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

{

double x\_mid = (all\_edges[i].\_2.x + all\_edges[i].\_1.x) / 2,

y\_mid = (all\_edges[i].\_2.y + all\_edges[i].\_1.y) / 2,

z\_mid = (all\_edges[i].\_2.z + all\_edges[i].\_1.z) / 2;

int num;

if (all\_edges[i].\_2.x != all\_edges[i].\_1.x)

num = 0;

else if (all\_edges[i].\_2.y != all\_edges[i].\_1.y)

num = 1;

else

num = 2;

q[k][i] = true\_func(x\_mid, y\_mid, z\_mid, num);

}

}

in.close();

return 0;

}

void FEM::Init\_const\_matrix()

{

D.matrix[0][0] = 4;

D.matrix[0][1] = 2;

D.matrix[0][2] = 2;

D.matrix[0][3] = 1;

D.matrix[1][0] = 2;

D.matrix[1][1] = 4;

D.matrix[1][2] = 1;

D.matrix[1][3] = 2;

D.matrix[2][0] = 2;

D.matrix[2][1] = 1;

D.matrix[2][2] = 4;

D.matrix[2][3] = 2;

D.matrix[3][0] = 1;

D.matrix[3][1] = 2;

D.matrix[3][2] = 2;

D.matrix[3][3] = 4;

G1.matrix[0][0] = 2;

G1.matrix[0][1] = 1;

G1.matrix[0][2] = -2;

G1.matrix[0][3] = -1;

G1.matrix[1][0] = 1;

G1.matrix[1][1] = 2;

G1.matrix[1][2] = -1;

G1.matrix[1][3] = -2;

G1.matrix[2][0] = -2;

G1.matrix[2][1] = -1;

G1.matrix[2][2] = 2;

G1.matrix[2][3] = 1;

G1.matrix[3][0] = -1;

G1.matrix[3][1] = -2;

G1.matrix[3][2] = 1;

G1.matrix[3][3] = 2;

G2.matrix[0][0] = 2;

G2.matrix[0][1] = -2;

G2.matrix[0][2] = 1;

G2.matrix[0][3] = -1;

G2.matrix[1][0] = -2;

G2.matrix[1][1] = 2;

G2.matrix[1][2] = -1;

G2.matrix[1][3] = 1;

G2.matrix[2][0] = 1;

G2.matrix[2][1] = -1;

G2.matrix[2][2] = 2;

G2.matrix[2][3] = -2;

G2.matrix[3][0] = -1;

G2.matrix[3][1] = 1;

G2.matrix[3][2] = -2;

G2.matrix[3][3] = 2;

G3.matrix[0][0] = -2;

G3.matrix[0][1] = 2;

G3.matrix[0][2] = -1;

G3.matrix[0][3] = 1;

G3.matrix[1][0] = -1;

G3.matrix[1][1] = 1;

G3.matrix[1][2] = -2;

G3.matrix[1][3] = 2;

G3.matrix[2][0] = 2;

G3.matrix[2][1] = -2;

G3.matrix[2][2] = 1;

G3.matrix[2][3] = -1;

G3.matrix[3][0] = 1;

G3.matrix[3][1] = -1;

G3.matrix[3][2] = 2;

G3.matrix[3][3] = -2;

GT3.matrix[0][0] = -2;

GT3.matrix[1][0] = 2;

GT3.matrix[2][0] = -1;

GT3.matrix[3][0] = 1;

GT3.matrix[0][1] = -1;

GT3.matrix[1][1] = 1;

GT3.matrix[2][1] = -2;

GT3.matrix[3][1] = 2;

GT3.matrix[0][2] = 2;

GT3.matrix[1][2] = -2;

GT3.matrix[2][2] = 1;

GT3.matrix[3][2] = -1;

GT3.matrix[0][3] = 1;

GT3.matrix[1][3] = -1;

GT3.matrix[2][3] = 2;

GT3.matrix[3][3] = -2;

}

int FEM::Loc\_matrix(int el\_id, std::vector<Matrix>& A\_loc, std::vector<double>& b\_loc)

{

double x1 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_1.x,

x2 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_2.x,

y1 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_1.y,

y2 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_2.y,

z1 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_1.z,

z2 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_2.z,

hx = x2 - x1,

hy = y2 - y1,

hz = z2 - z1,

mu = all\_materials[all\_elems[el\_id].mater].mu,

gam = all\_materials[all\_elems[el\_id].mater].sigma,

koef\_M = hx\* hy\*hz / 36;

//локальная матрица масс для gam = 1

A\_loc[0] = D \* koef\_M;

A\_loc[4] = D \* koef\_M;

A\_loc[8] = D \* koef\_M;

//b = C\*F

std::vector<double> F(12);

double x\_mid = (x2 + x1) / 2,

y\_mid = (y2 + y1) / 2,

z\_mid = (z2 + z1) / 2;

F[0] = func\_f(x\_mid, y1, z1, all\_elems[el\_id].f\_id, 0);

F[1] = func\_f(x\_mid, y2, z1, all\_elems[el\_id].f\_id, 0);

F[2] = func\_f(x\_mid, y1, z2, all\_elems[el\_id].f\_id, 0);

F[3] = func\_f(x\_mid, y2, z2, all\_elems[el\_id].f\_id, 0);

F[4] = func\_f(x1, y\_mid, z1, all\_elems[el\_id].f\_id, 1);

F[5] = func\_f(x2, y\_mid, z1, all\_elems[el\_id].f\_id, 1);

F[6] = func\_f(x1, y\_mid, z2, all\_elems[el\_id].f\_id, 1);

F[7] = func\_f(x2, y\_mid, z2, all\_elems[el\_id].f\_id, 1);

F[8] = func\_f(x1, y1, z\_mid, all\_elems[el\_id].f\_id, 2);

F[9] = func\_f(x2, y1, z\_mid, all\_elems[el\_id].f\_id, 2);

F[10] = func\_f(x1, y2, z\_mid, all\_elems[el\_id].f\_id, 2);

F[11] = func\_f(x2, y2, z\_mid, all\_elems[el\_id].f\_id, 2);

for (int k = 0; k < 12; k += 4)

{

for (int i = k; i < k + 4; i++)

{

b\_loc[i] = 0;

for (int j = 0; j < 4; j++)

{

b\_loc[i] += A\_loc[k].matrix[i % 4][j] \* F[k + j];

}

}

}

// собираем локальную матрицу с учетом гаммы и матрицы жесткости

A\_loc[0] = A\_loc[0] \* gam + (G1 \* (hx \* hy / (6 \* hz)) + G2 \* (hx \* hz / (6 \* hy))) \* (1 / mu);

A\_loc[1] = G2 \* (-hz / 6 / mu);

A\_loc[2] = G3 \* (hy / 6 / mu);

A\_loc[3] = G2 \* (-hz / 6 / mu);

A\_loc[4] = A\_loc[4] \* gam + (G1 \* (hx \* hy / (6 \* hz)) + G2 \* (hy \* hz / (6 \* hx))) \* (1 / mu);

A\_loc[5] = G1 \* (-hx / 6 / mu);

A\_loc[6] = GT3 \* (hy / 6 / mu);

A\_loc[7] = G1 \* (-hx / 6 / mu);

A\_loc[8] = A\_loc[8] \* gam + (G1 \* (hx \* hz / (6 \* hy)) + G2 \* (hy \* hz / (6 \* hx))) \* (1 / mu);

return 0;

}

int FEM::Loc\_matrix\_time(int el\_id, std::vector<Matrix>& M\_loc, std::vector<Matrix>& G\_loc/\*, std::vector<double>& b\_loc\*/)

{

double x1 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_1.x,

x2 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_2.x,

y1 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_1.y,

y2 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_2.y,

z1 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_1.z,

z2 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_2.z,

hx = x2 - x1,

hy = y2 - y1,

hz = z2 - z1,

mu = all\_materials[all\_elems[el\_id].mater].mu,

gam = all\_materials[all\_elems[el\_id].mater].sigma, // это сигма

koef\_M = hx \* hy \* hz / 36;

// локальная матрица масс

M\_loc[0] = D \* koef\_M \* gam ;

M\_loc[4] = D \* koef\_M \* gam ;

M\_loc[8] = D \* koef\_M \* gam ;

// собираем локальную матрицу жесткости

G\_loc[0] = (G1 \* (hx \* hy / (6 \* hz)) + G2 \* (hx \* hz / (6 \* hy))) \* (1 / mu);

G\_loc[1] = G2 \* (-hz / 6 / mu);

G\_loc[2] = G3 \* (hy / 6 / mu);

G\_loc[3] = G2 \* (-hz / 6 / mu);

G\_loc[4] = (G1 \* (hx \* hy / (6 \* hz)) + G2 \* (hy \* hz / (6 \* hx))) \* (1 / mu);

G\_loc[5] = G1 \* (-hx / 6 / mu);

G\_loc[6] = GT3 \* (hy / 6 / mu);

G\_loc[7] = G1 \* (-hx / 6 / mu);

G\_loc[8] = (G1 \* (hx \* hz / (6 \* hy)) + G2 \* (hy \* hz / (6 \* hx))) \* (1 / mu);

return 0;

}

int FEM::Loc\_b\_time(int el\_id, std::vector<double>& b\_loc)

{

std::vector<Matrix> M\_loc(9);

double x1 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_1.x,

x2 = all\_edges[all\_elems[el\_id].edge\_loc[0]].\_2.x,

y1 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_1.y,

y2 = all\_edges[all\_elems[el\_id].edge\_loc[4]].\_2.y,

z1 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_1.z,

z2 = all\_edges[all\_elems[el\_id].edge\_loc[8]].\_2.z,

hx = x2 - x1,

hy = y2 - y1,

hz = z2 - z1,

koef\_M = hx \* hy \* hz / 36;

//локальная матрица масс для гамма = 1

M\_loc[0] = D \* koef\_M;

M\_loc[4] = D \* koef\_M;

M\_loc[8] = D \* koef\_M;

//b = C\*F

std::vector<double> F(12);

double x\_mid = (x2 + x1) / 2,

y\_mid = (y2 + y1) / 2,

z\_mid = (z2 + z1) / 2;

F[0] = func\_f(x\_mid, y1, z1, all\_elems[el\_id].f\_id, 0);

F[1] = func\_f(x\_mid, y2, z1, all\_elems[el\_id].f\_id, 0);

F[2] = func\_f(x\_mid, y1, z2, all\_elems[el\_id].f\_id, 0);

F[3] = func\_f(x\_mid, y2, z2, all\_elems[el\_id].f\_id, 0);

F[4] = func\_f(x1, y\_mid, z1, all\_elems[el\_id].f\_id, 1);

F[5] = func\_f(x2, y\_mid, z1, all\_elems[el\_id].f\_id, 1);

F[6] = func\_f(x1, y\_mid, z2, all\_elems[el\_id].f\_id, 1);

F[7] = func\_f(x2, y\_mid, z2, all\_elems[el\_id].f\_id, 1);

F[8] = func\_f(x1, y1, z\_mid, all\_elems[el\_id].f\_id, 2);

F[9] = func\_f(x2, y1, z\_mid, all\_elems[el\_id].f\_id, 2);

F[10] = func\_f(x1, y2, z\_mid, all\_elems[el\_id].f\_id, 2);

F[11] = func\_f(x2, y2, z\_mid, all\_elems[el\_id].f\_id, 2);

for (int k = 0; k < 12; k += 4)

{

for (int i = k; i < k + 4; i++)

{

b\_loc[i] = 0;

for (int j = 0; j < 4; j++)

{

b\_loc[i] += M\_loc[k].matrix[i % 4][j] \* F[k + j];

}

}

}

return 0;

}

int FEM::GeneratePortrait() // генерация портрета

{

di.resize(N);

diM.resize(N);

diG.resize(N);

b.resize(N);

ia.resize(N + 1);

ja.resize(144 \* Kel);

std::vector<int> temp\_list1(144 \* Kel),

temp\_list2(144 \* Kel);

std::vector<int> listbeg(N);

int listsize = 0;

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

{

listbeg[i] = 0;

}

for (int ielem = 0; ielem < Kel; ielem++)

{

for (int i = 0; i < 12; i++)

{

int k = all\_elems[ielem].edge\_loc[i];

for (int j = i + 1; j < 12; j++)

{

int ind1 = k;

int ind2 = all\_elems[ielem].edge\_loc[j];

if (ind2 < ind1)

{

ind1 = ind2;

ind2 = k;

}

int iaddr = listbeg[ind2];

if (iaddr == 0)

{

listsize++;

listbeg[ind2] = listsize;

temp\_list1[listsize] = ind1;

temp\_list2[listsize] = 0;

}

else

{

while (temp\_list1[iaddr] < ind1 && temp\_list2[iaddr] > 0)

{

iaddr = temp\_list2[iaddr];

}

if (temp\_list1[iaddr] > ind1)

{

listsize++;

temp\_list1[listsize] = temp\_list1[iaddr];

temp\_list2[listsize] = temp\_list2[iaddr];

temp\_list1[iaddr] = ind1;

temp\_list2[iaddr] = listsize;

}

else if (temp\_list1[iaddr] < ind1)

{

listsize++;

temp\_list2[iaddr] = listsize;

temp\_list1[listsize] = ind1;

temp\_list2[listsize] = 0;

}

}

}

}

}

ia[0] = 0;

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

{

ia[i + 1] = ia[i];

int iaddr = listbeg[i];

while (iaddr != 0)

{

ja[ia[i + 1]] = temp\_list1[iaddr];

ia[i + 1]++;

iaddr = temp\_list2[iaddr];

}

}

ja.resize(ia[N]);

al.resize(ia[N]);

alM.resize(ia[N]);

alG.resize(ia[N]);

return 0;

}

int FEM::AddLocal(int el\_id, std::vector<Matrix>& A\_loc)

// внесение локальных A в глобальную СЛАУ

{

std::vector<int> L = all\_elems[el\_id].edge\_loc;

int k = all\_elems[el\_id].edge\_loc.size(); // размерность локальной матрицы

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

{

di[L[i]] += A\_loc[0].matrix[i][i];

di[L[i + 4]] += A\_loc[4].matrix[i][i];

di[L[i + 8]] += A\_loc[8].matrix[i][i];

}

for (int i = 0; i < 12; i++)

{

int temp = ia[L[i]];

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

{

int loc\_pos = (i / 4) \* 3 + j / 4,

loc\_i = i % 4,

loc\_j = j % 4;

for (int k = temp; k < ia[L[i] + 1]; k++)

{

if (ja[k] == L[j])

{

al[k] += A\_loc[loc\_pos].matrix[loc\_i][loc\_j];

//std::cout << loc\_pos << " " << loc\_i << " " << loc\_j << "\n";

k++;

break;

}

}

}

}

return 0;

}

int FEM::AddLocalM(int el\_id, std::vector<Matrix>& M\_loc)

// внесение локальных A в глобальную СЛАУ

{

std::vector<int> L = all\_elems[el\_id].edge\_loc;

int k = all\_elems[el\_id].edge\_loc.size(); // размерность локальной матрицы

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

{

diM[L[i]] += M\_loc[0].matrix[i][i];

diM[L[i + 4]] += M\_loc[4].matrix[i][i];

diM[L[i + 8]] += M\_loc[8].matrix[i][i];

}

for (int i = 0; i < 12; i++)

{

int temp = ia[L[i]];

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

{

int loc\_pos = (i / 4) \* 3 + j / 4,

loc\_i = i % 4,

loc\_j = j % 4;

for (int k = temp; k < ia[L[i] + 1]; k++)

{

if (ja[k] == L[j])

{

alM[k] += M\_loc[loc\_pos].matrix[loc\_i][loc\_j];

k++;

break;

}

}

}

}

return 0;

}

int FEM::AddLocalG(int el\_id, std::vector<Matrix>& G\_loc)

// внесение локальных A в глобальную СЛАУ

{

std::vector<int> L = all\_elems[el\_id].edge\_loc;

int k = all\_elems[el\_id].edge\_loc.size(); // размерность локальной матрицы

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

{

diG[L[i]] += G\_loc[0].matrix[i][i];

diG[L[i + 4]] += G\_loc[4].matrix[i][i];

diG[L[i + 8]] += G\_loc[8].matrix[i][i];

}

for (int i = 0; i < 12; i++)

{

int temp = ia[L[i]];

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

{

int loc\_pos = (i / 4) \* 3 + j / 4,

loc\_i = i % 4,

loc\_j = j % 4;

for (int k = temp; k < ia[L[i] + 1]; k++)

{

if (ja[k] == L[j])

{

alG[k] += G\_loc[loc\_pos].matrix[loc\_i][loc\_j];

k++;

break;

}

}

}

}

return 0;

}

int FEM::AddLocal\_b(int el\_id, std::vector<double>& b\_loc)

// внесение локальных b в глобальную СЛАУ

{

std::vector<int> L = all\_elems[el\_id].edge\_loc;

int k = all\_elems[el\_id].edge\_loc.size(); // размерность локальной матрицы

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

{

b[L[i]] += b\_loc[i];

}

return 0;

}

int FEM::SetS1() // учет первых краевых

{

int NS1 = S1.size();

for (int i = 0; i < NS1; i++)

{

int s1\_id = i;

for (int j = 0; j < S1[i].size(); j++)

{

int node\_id = S1[i][j];

double

x\_mid = (all\_edges[node\_id].\_2.x + all\_edges[node\_id].\_1.x) / 2,

y\_mid = (all\_edges[node\_id].\_2.y + all\_edges[node\_id].\_1.y) / 2,

z\_mid = (all\_edges[node\_id].\_2.z + all\_edges[node\_id].\_1.z) / 2;

di[node\_id] = M;

b[node\_id] = M \* func\_S1(x\_mid, y\_mid, z\_mid, s1\_id);

}

}

return 0;

}

int FEM::SolveTask(std::vector<double> &res)

{

Init\_const\_matrix();

Input();

GeneratePortrait();

std::vector<Matrix> A\_loc(9);

std::vector<double> b\_loc(12);

for (int i = 0; i < all\_elems.size(); i++)

{

Loc\_matrix(i, A\_loc, b\_loc);

AddLocal(i, A\_loc);

AddLocal\_b(i, b\_loc);

}

SetS1();

cgm\_solver t(ia, ja, di, al, b);

t.llt\_preconditioning(q[2]);

res = q[2];

std::vector<double> U\_in\_point(3);

ValueInPointXYZ(U\_in\_point, 1.2, 1.3, 1.4, 2);

std::cout.precision(15);

std::cout.imbue(std::locale("Russian"));

std::cout << "u\*\n" << true\_func(1.2, 1.3, 1.4, 0) << "\n"

<< true\_func(1.2, 1.3, 1.4, 1) << "\n"

<< true\_func(1.2, 1.3, 1.4, 2) << "\n";

std::cout << "u\n" << U\_in\_point[0] << "\n"

<< U\_in\_point[1] << "\n"

<< U\_in\_point[2] << "\n";

return 0;

}

int FEM::SolveTask\_time()

{

std::ofstream out("result.txt");

out.precision(15);

out.imbue(std::locale("Russian"));

Init\_const\_matrix();

Input();

GeneratePortrait();

std::vector<Matrix> M\_loc(9), G\_loc(9);

std::vector<double> b\_loc(12);

//----------------------------------------

std::vector<node> test\_points;

std::ifstream input("test points.txt");

int n\_test;

input >> n\_test;

test\_points.resize(n\_test);

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

{

input >> test\_points[i].x >> test\_points[i].y >> test\_points[i].z;

}

//-----------------------------------------

std::cout << "start make M, G\n";

for (int i = 0; i < all\_elems.size(); i++)

{

std::cout << "on elem " << i << "\n";

Loc\_matrix\_time(i, M\_loc, G\_loc);

AddLocalM(i, M\_loc);

AddLocalG(i, G\_loc);

}

double koef1, koef2, koef3;

double dt01 = 0, dt02 = 0, dt12 = 0;

for (i\_t = 2; i\_t < time\_grid.size(); i\_t++)

{

if (dt01 != time\_grid[i\_t] - time\_grid[i\_t - 1] ||

dt02 != time\_grid[i\_t] - time\_grid[i\_t - 2] ||

dt12 != time\_grid[i\_t - 1] - time\_grid[i\_t - 2])

{

dt01 = time\_grid[i\_t] - time\_grid[i\_t - 1];

dt02 = time\_grid[i\_t] - time\_grid[i\_t - 2];

dt12 = time\_grid[i\_t - 1] - time\_grid[i\_t - 2];

koef1 = (dt01 + dt02) / dt01 / dt02;

koef2 = dt02 / dt01 / dt12;

koef3 = -dt01 / dt02 / dt12;

plus\_matrix\_matrix(diM, alM, diG, alG, di, al, koef1); // A = koef1 \* M + G

}

// (koef1 \* M + G) \* q\_(2) = b\_(j) + koef2 \* M \* q\_(1) + koef3 \* M \* q\_(0)

// пересобрать вектор правой части

std::fill(b.begin(), b.end(), 0);

for (int i = 0; i < all\_elems.size(); i++)

{

Loc\_b\_time(i, b\_loc);

AddLocal\_b(i, b\_loc);

}

std::vector<double> temp(N);

// koef2 \* M \* q\_(1)

mult\_matrix\_vector(ia, ja, diM, alM, q[1], temp);

mult\_vector\_double(temp, koef2);

plus\_vector\_vector(b, temp);

std::fill(temp.begin(), temp.end(), 0);

// koef3 \* M \* q\_(0)

mult\_matrix\_vector(ia, ja, diM, alM, q[0], temp);

mult\_vector\_double(temp, koef3);

plus\_vector\_vector(b, temp);

SetS1();

cgm\_solver t(ia, ja, di, al, b);

t.llt\_preconditioning(q[2]);

if ((int)time\_grid[i\_t] > 1 && (int)time\_grid[i\_t] != (int)time\_grid[i\_t - 1])

{

out << "time = " << "\t" << time\_grid[i\_t] << "\n"

<< "x" << "\t" << "y" << "\t" << "z" << "\t" << "u\*" << "\t" << "u" << "\t" << "|u\* - u|\n";

for (int i = 0; i < test\_points.size(); i++)

{

std::vector<double> U\_in\_point(3);

double tr\_u0 = true\_func(test\_points[i].x, test\_points[i].y, test\_points[i].z, 0),

tr\_u1 = true\_func(test\_points[i].x, test\_points[i].y, test\_points[i].z, 1),

tr\_u2 = true\_func(test\_points[i].x, test\_points[i].y, test\_points[i].z, 2);

ValueInPointXYZ(U\_in\_point, test\_points[i].x, test\_points[i].y, test\_points[i].z, 2);

out

<< test\_points[i].x << "\t" << test\_points[i].y << "\t" << test\_points[i].z << "\t"

<< tr\_u0 << "\t" << U\_in\_point[0] << "\t" << abs(U\_in\_point[0] - tr\_u0) << "\n\t\t\t"

<< tr\_u1 << "\t" << U\_in\_point[1] << "\t" << abs(U\_in\_point[1] - tr\_u1) << "\n\t\t\t"

<< tr\_u2 << "\t" << U\_in\_point[2] << "\t" << abs(U\_in\_point[2] - tr\_u2) << "\n";

}

}

q[0].swap(q[1]);

q[1].swap(q[2]);

}

return 0;

}

void FEM::ValueInPointXYZ(std::vector<double> &U\_in\_point, double x, double y, double z, int time\_id)

{

// find elem

double x1, x2, y1, y2, z1, z2;

int cur\_el = -1;

for (int i = 0; i < all\_elems.size() && cur\_el == -1; i++)

{

x1 = all\_edges[all\_elems[i].edge\_loc[0]].\_1.x;

x2 = all\_edges[all\_elems[i].edge\_loc[0]].\_2.x;

y1 = all\_edges[all\_elems[i].edge\_loc[0]].\_1.y;

y2 = all\_edges[all\_elems[i].edge\_loc[1]].\_1.y;

z1 = all\_edges[all\_elems[i].edge\_loc[0]].\_1.z;

z2 = all\_edges[all\_elems[i].edge\_loc[3]].\_1.z;

if (x1 < x && x < x2 &&

y1 < y && y < y2 &&

z1 < z && z < z2)

cur\_el = i;

}

if (cur\_el == -1)

{

std::cout << "can't find (" << x << " " << y << " " << z << ")\n";

return;

}

double hx = x2 - x1, hy = y2 - y1, hz = z2 - z1;

U\_in\_point[0] =

q[time\_id][all\_elems[cur\_el].edge\_loc[0]] \* (y2 - y) \* (z2 - z) / hy / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[1]] \* (y - y1) \* (z2 - z) / hy / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[2]] \* (y2 - y) \* (z - z1) / hy / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[3]] \* (y - y1) \* (z - z1) / hy / hz;

U\_in\_point[1] =

q[time\_id][all\_elems[cur\_el].edge\_loc[4]] \* (x2 - x) \* (z2 - z) / hx / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[5]] \* (x - x1) \* (z2 - z) / hx / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[6]] \* (x2 - x) \* (z - z1) / hx / hz +

q[time\_id][all\_elems[cur\_el].edge\_loc[7]] \* (x - x1) \* (z - z1) / hx / hz;

U\_in\_point[2] =

q[time\_id][all\_elems[cur\_el].edge\_loc[8]] \* (x2 - x) \* (y2 - y) / hx / hy +

q[time\_id][all\_elems[cur\_el].edge\_loc[9]] \* (x - x1) \* (y2 - y) / hx / hy +

q[time\_id][all\_elems[cur\_el].edge\_loc[10]] \* (x2 - x) \* (y - y1) / hx / hy +

q[time\_id][all\_elems[cur\_el].edge\_loc[11]] \* (x - x1) \* (y - y1) / hx / hy;

}

void mult\_matrix\_vector(std::vector<int>& ia, std::vector<int>& ja, std::vector<double>& di, std::vector<double>& al, std::vector<double>& x, std::vector<double>& y)

{

for (int i = 0; i < di.size(); i++)

{

y[i] = di[i] \* x[i];

for (int j = ia[i]; j < ia[i + 1]; j++)

{

int k = ja[j];

y[i] += al[j] \* x[k];

y[k] += al[j] \* x[i];

}

}

}

void mult\_matrix\_double(std::vector<double>& di, std::vector<double>& al, double x)

{

for (int i = 0; i < di.size(); i++)

{

di[i] \*= x;

}

for (int i = 0; i < al.size(); i++)

{

al[i] \*= x;

}

}

void mult\_vector\_double(std::vector<double>& a, double x)

{

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

{

a[i] \*= x;

}

}

void plus\_vector\_vector(std::vector<double>& a, std::vector<double>& b)

{

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

{

a[i] += b[i];

}

}

void plus\_matrix\_matrix(

std::vector<double>& diM, std::vector<double>& alM, std::vector<double>& diG, std::vector<double>& alG,

std::vector<double>& di, std::vector<double>& al, double koef1)

{

for (int i = 0; i < di.size(); i++)

{

di[i] = diM[i];

}

for (int i = 0; i < al.size(); i++)

{

al[i] = alM[i];

}

mult\_matrix\_double(di, al, koef1);

for (int i = 0; i < di.size(); i++)

{

di[i] += diG[i];

}

for (int i = 0; i < al.size(); i++)

{

al[i] += alG[i];

}

}

**Base.h**

#pragma once

#include <fstream>

#include <iostream>

#include "Matrix.h"

#include "cgm.h"

struct node // структура узла

{

double x;

double y;

double z;

};

struct edge // структура ребра

{

node \_1, \_2;

};

struct material // структура материала

{

double mu;

double sigma;

};

struct element // структура элемента

{

std::vector<int> edge\_loc;

int mater;

int f\_id;

};

void mult\_matrix\_vector(std::vector<int> &ia, std::vector<int> &ja, std::vector<double> &di, std::vector<double> &al, std::vector<double> &x, std::vector<double>& y); // умножение матрицы на вектор х, результат в у

void mult\_matrix\_double(std::vector<double>& di, std::vector<double>& al, double x); // умножение на месте матрицы на число

void mult\_vector\_double(std::vector<double>& a, double x); // умножение на месте вектора на число

void plus\_vector\_vector(std::vector<double>& a, std::vector<double>& b); // сложение векторов, в вектор а

void plus\_matrix\_matrix(std::vector<double>& diM, std::vector<double>& alM, std::vector<double>& diG, std::vector<double>& alG, std::vector<double>& di, std::vector<double>& al, double koef1); // A = koef1 \* M + G

class FEM

{

private:

std::vector<edge> all\_edges; // все ребра в порядке глобальной нумерации

std::vector<element> all\_elems; // все элементы в прядке глобальной нумерации

std::vector<material> all\_materials; // все материалы по индексам

std::vector<std::vector<int>> S1; // S1[i][j] на j-ом узле заданы i-ые краевые 1 рода

std::vector<std::vector<double>> q; // значения q на трех слоях: q0, q1 известны, q2 ищем

int i\_t = 0; // текущий временной слой

std::vector<double> time\_grid; // сетка по времени

// матрица A

std::vector<int> ia, ja; // портрет

std::vector<double> di, al, b; // симметричная

// М

std::vector<double> diM, alM;

// G

std::vector<double> diG, alG;

int N, Kel; // кол-во ребер и элементов

const double M = 1E+15; // большое число, используется для учета краевых первого рода

Matrix D, G1, G2, G3, GT3; // константные матрицы, используемые для сборки локальных

void Init\_const\_matrix(); // инициализация константных матриц

double func\_f(double x, double y, double z, int f\_id, int num); // значение f по индексу f\_id

double func\_S1(double x, double y, double z, int s1\_id); // значение краевого S1 по индексу s1\_id

double true\_func(double x, double y, double z, int num); // истинная функция (используется для определения погрешности решения)

int Input(); // чтение данных

int Loc\_matrix(int el\_id, std::vector<Matrix>& A\_loc, std::vector<double>& b\_loc); // сборка локальной матрицы A = M + G

int Loc\_matrix\_time(int el\_id, std::vector<Matrix>& M\_loc, std::vector<Matrix>& G\_loc); // сборка локальных матриц M и G

int Loc\_b\_time(int el\_id, std::vector<double>& b\_loc); // сборка локального вектора b

int GeneratePortrait(); // генерация портрета матрицы А

int AddLocal(int el\_id, std::vector<Matrix>& A\_loc); // внесение локальной матрицы в глобальную А

int AddLocalM(int el\_id, std::vector<Matrix>& M\_loc); // внесение локальной матрицы в глобальную M

int AddLocalG(int el\_id, std::vector<Matrix>& G\_loc); // внесение локальной матрицы в глобальную G

int AddLocal\_b(int el\_id, std::vector<double>& b\_loc); // внесение локального вектора правой части в глобальный

int SetS1(); // учет первых краевых условий

public:

int SolveTask(std::vector<double> &res); // решение стационарной задачи

int SolveTask\_time(); // решение нестационарной задачи

void ValueInPointXYZ(std::vector<double>& A, double x, double y, double z, int time\_id); // значение численно найденной функции в произвольной точке пространства в одном из трех последних временных узлов

};

**cgm.cpp**

#include "cgm.h"

// ind\_from\_zero - истина, если в файлах ig и jg содержат индексы с 0

cgm\_solver::cgm\_solver(const std::string& path, const bool& ind\_from\_zero)

{

// n, max\_iter и eps

std::ifstream in(path + "kuslau.txt");

in >> n >> max\_iter >> eps;

s.resize(n);

az.resize(n);

rr\_vec.resize(n);

z.resize(n);

r.resize(n);

in.close();

// ig

in.open(path + "ig.txt");

ig.resize(n + 1);

for (int i = 0; i < ig.size(); i++)

{

in >> ig[i];

if (!ind\_from\_zero)

--ig[i];

}

Ll.resize(ig.back());

Ld.resize(n);

in.close();

// jg

in.open(path + "jg.txt");

jg.resize(ig.back());

for (int i = 0; i < jg.size(); i++)

{

in >> jg[i];

if (!ind\_from\_zero)

--jg[i];

}

in.close();

// ggl

in.open(path + "ggl.txt");

ggl.resize(ig.back());

for (int i = 0; i < ggl.size(); i++)

in >> ggl[i];

in.close();

// di

in.open(path + "di.txt");

diag.resize(n);

for (int i = 0; i < diag.size(); i++)

in >> diag[i];

in.close();

// pr

in.open(path + "pr.txt");

pr.resize(n);

for (int i = 0; i < pr.size(); i++)

in >> pr[i];

pr\_norm = norm(pr);

}

cgm\_solver::cgm\_solver(std::vector<int> &\_ia, std::vector<int>& \_ja,

std::vector<double>& \_di, std::vector<double>& \_al, std::vector<double>& \_b)

{

n = \_di.size();

max\_iter = 10000;

eps = 1E-25;

ggu = \_al;

pr = \_b;

diag = \_di;

ig = \_ia;

jg = \_ja;

s.resize(n);

az.resize(n);

rr\_vec.resize(n);

z.resize(n);

r.resize(n);

Ll.resize(ig.back());

Ld.resize(n);

pr\_norm = norm(pr);

}

int cgm\_solver::get\_n()

{

return n;

}

int cgm\_solver::get\_total\_iter()

{

return total\_iter;

}

// умножение матрицы СЛАУ на вектор x, результат помещается в вектор y

void cgm\_solver::matrix\_dot\_vector(const std::vector<double>& x, std::vector<double>& y)

{

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

{

y[i] = diag[i] \* x[i];

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

{

y[i] += ggl[j] \* x[jg[j]];

y[jg[j]] += ggu[j] \* x[i];

}

}

}

// разложение Холецкого для матрицы СЛАУ

void cgm\_solver::llt()

{

int p = 0, m = 0;

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

{

Ld[i] = 0;

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

{

Ll[j] = ggu[j];

p = jg[j];

m = ig[i];

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

{

for (int q = m; q < j; q++)

if (jg[q] == jg[k])

{

Ll[j] -= Ll[q] \* Ll[k];

m = q + 1;

break;

}

}

Ll[j] /= Ld[jg[j]];

Ld[i] -= Ll[j] \* Ll[j];

}

Ld[i] = sqrt(diag[i] + Ld[i]);

}

}

// решение СЛАУ без предобуславливания

void cgm\_solver::no\_preconditioning(std::vector<double>& x0)

{

// r = A \* x0

matrix\_dot\_vector(x0, r);

// r = pr - A \* x0 = pr - r

vec\_diff(pr, r, r);

// z = r

z = r;

double r\_norm = norm(r), rr = r\_norm / pr\_norm;

while (rr >= eps && total\_iter < max\_iter)

{

// az = A \* z

matrix\_dot\_vector(z, az);

// скалярное произведение p и r с предыдщуей итерации необходи-мо сохранить,

// оно понадобится для вычисления bk после изменения p и r

double scal\_r\_r = scalar\_product(r, r), alpha\_k = scal\_r\_r / scalar\_product(az, z);

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

{

x0[i] += alpha\_k \* z[i];

r[i] -= alpha\_k \* az[i];

}

double beta\_k = scalar\_product(r, r) / scal\_r\_r;

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

z[i] = r[i] + beta\_k \* z[i];

r\_norm = norm(r);

rr = r\_norm / pr\_norm;

//std::cout << "iter #" << total\_iter << " | rr = " << rr << std::endl;

total\_iter++;

}

std::cout << "RELATIVE RESIDUAL = " << relative\_residual(x0) << std::endl;

}

// решение СЛАУ с диагональным предобуславливанием

void cgm\_solver::diag\_preconditioning(std::vector<double>& x0)

{

// r = A \* x0

matrix\_dot\_vector(x0, r);

// r = pr - A \* x0 = pr - r

vec\_diff(pr, r, r);

// z = M^(-1) \* r

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

z[i] = r[i] / diag[i];

double r\_norm = norm(r), rr = r\_norm / pr\_norm;

while (rr >= eps && total\_iter < max\_iter)

{

// az = A \* z

matrix\_dot\_vector(z, az);

double scal\_m\_inv\_r = scalar\_product(z, r), alpha\_k = scal\_m\_inv\_r / scalar\_product(az, z);

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

{

x0[i] += alpha\_k \* z[i];

r[i] -= alpha\_k \* az[i];

}

// s = M^(-1) \* r

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

s[i] = r[i] / diag[i];

double beta\_k = scalar\_product(s, r) / scal\_m\_inv\_r;

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

z[i] = s[i] + beta\_k \* z[i];

r\_norm = norm(r);

rr = r\_norm / pr\_norm;

//std::cout << "iter #" << total\_iter << " | rr = " << rr << std::endl;

total\_iter++;

}

std::cout << "RELATIVE RESIDUAL = " << relative\_residual(x0) << std::endl;

}

// решение СЛАУ с предобуславливанием неполным разложением Холецкого

void cgm\_solver::llt\_preconditioning(std::vector<double>& x0)

{

llt();

if (x0.size() != r.size())

x0.resize(r.size());

// r = A \* x0

matrix\_dot\_vector(x0, r);

// r = pr - A \* x0 = pr - r

vec\_diff(pr, r, r);

// z = M^(-1) \* r

solve\_auxiliary\_system(r, z);

double r\_norm = norm(r), rr = r\_norm / pr\_norm;

while (rr >= eps && total\_iter < max\_iter)

{

// s = M^(-1) \* r

solve\_auxiliary\_system(r, s);

// az = A \* z

matrix\_dot\_vector(z, az);

double scal\_m\_inv\_r = scalar\_product(s, r), alpha\_k = scal\_m\_inv\_r / scalar\_product(az, z);

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

{

x0[i] += alpha\_k \* z[i];

r[i] -= alpha\_k \* az[i];

}

// s = M^(-1) \* r

solve\_auxiliary\_system(r, s);

double beta\_k = scalar\_product(s, r) / scal\_m\_inv\_r;

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

z[i] = s[i] + beta\_k \* z[i];

r\_norm = norm(r);

rr = r\_norm / pr\_norm;

std::cout << "iter #" << total\_iter << " | rr = " << rr << std::endl;

total\_iter++;

}

std::cout << "RELATIVE RESIDUAL = " << relative\_residual(x0) << std::endl;

}

// решение вспомогательной СЛАУ (используется в методе с предобуславли-ванием разложением Холецкого)

void cgm\_solver::solve\_auxiliary\_system(const std::vector<double>& f, std::vector<double>& x)

{

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

{

double sum = 0.;

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

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

x[i] = (f[i] - sum) / Ld[i];

}

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

{

x[i] /= Ld[i];

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

x[jg[j]] -= Ll[j] \* x[i];

}

}

double cgm\_solver::relative\_residual(const std::vector<double>& x)

{

matrix\_dot\_vector(x, rr\_vec);

vec\_diff(pr, rr\_vec, rr\_vec);

return norm(rr\_vec) / pr\_norm;

}

void cgm\_solver::vec\_diff(const std::vector<double>& x, const std::vector<double>& y, std::vector<double>& res)

{

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

res[i] = x[i] - y[i];

}

double cgm\_solver::scalar\_product(const std::vector<double>& x, const std::vector<double>& y)

{

double sp = 0.;

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

sp += x[i] \* y[i];

return sp;

}

double cgm\_solver::norm(const std::vector<double>& x)

{

return sqrt(scalar\_product(x, x));

}

void cgm\_solver::reset()

{

total\_iter = 0;

s.assign(s.size(), 0.);

az.assign(az.size(), 0.);

rr\_vec.assign(rr\_vec.size(), 0.);

z.assign(z.size(), 0.);

r.assign(r.size(), 0.);

Ld.assign(Ld.size(), 0.);

Ll.assign(Ll.size(), 0.);

}

**cgm.h**

#pragma once

#pragma once

#include <iostream>

#include <fstream>

#include <vector>

#include <math.h>

class cgm\_solver

{

private:

int n = 0, total\_iter = 0, max\_iter = 0;

double eps = 0., pr\_norm = 0.;

std::vector<int> ig, jg;

std::vector<double> ggu, & ggl = ggu, diag, pr;

std::vector<double> r, z, rr\_vec, az, s, Ll, Ld;

void llt();

void matrix\_dot\_vector(const std::vector<double>& x, std::vector<double>& y);

void matrix\_dot\_vector\_llt(const std::vector<double>& x, std::vector<double>& y);

void vec\_diff(const std::vector<double>& x, const std::vector<double>& y, std::vector<double>& res);

double scalar\_product(const std::vector<double>& x, const std::vector<double>& y);

double norm(const std::vector<double>& x);

void solve\_auxiliary\_system(const std::vector<double>& f, std::vector<double>& x);

double relative\_residual(const std::vector<double>& x);

public:

cgm\_solver(const std::string& path, const bool& ind\_from\_zero = true);

cgm\_solver(std::vector<int>& ia, std::vector<int> &ja,

std::vector<double> &di, std::vector<double> &al, std::vector<double> &b);

int get\_n();

int get\_total\_iter();

void no\_preconditioning(std::vector<double>& x0);

void diag\_preconditioning(std::vector<double>& x0);

void llt\_preconditioning(std::vector<double>& x0);

void reset();

};

**Generate.cpp**

#include "Generate.h"

void Make\_grid(std::string path)

{

std::ofstream out;

out.precision(15);

std::vector<double> all\_X, all\_Y, all\_Z;

std::ifstream in(path + "grid.txt");

double X, Y, Z, kx, ky, kz;

int Nx, Ny, Nz;

int count\_x, count\_y, count\_z;

in >> count\_x >> count\_y >> count\_z;

all\_X.resize(count\_x);

all\_Y.resize(count\_y);

all\_Z.resize(count\_z);

in >> all\_X[0] >> all\_Y[0] >> all\_Z[0];

for (int curr\_count\_x = 0; curr\_count\_x < count\_x - 1; )

{

in >> X >> Nx >> kx;

double hx;

if (kx == 1)

{

hx = (X - all\_X[curr\_count\_x]) / Nx;

for (int p = 1; p < Nx; p++)

{

all\_X[curr\_count\_x + p] = all\_X[curr\_count\_x] + hx \* p;

}

curr\_count\_x += Nx;

}

else

{

hx = (X - all\_X[curr\_count\_x]) \* (kx - 1) / (pow(kx, Nx) - 1);

for (int p = 0; p < Nx - 1; curr\_count\_x++, p++)

{

all\_X[curr\_count\_x + 1] = all\_X[curr\_count\_x] + hx \* pow(kx, p);

}

curr\_count\_x++;

}

all\_X[curr\_count\_x] = X;

}

for (int curr\_count\_y = 0; curr\_count\_y < count\_y - 1; )

{

in >> Y >> Ny >> ky;

double hy;

if (ky == 1)

{

hy = (Y - all\_Y[curr\_count\_y]) / Ny;

for (int p = 1; p < Ny; p++)

{

all\_Y[curr\_count\_y + p] = all\_Y[curr\_count\_y] + hy \* p;

}

curr\_count\_y += Ny;

}

else

{

hy = (Y - all\_Y[curr\_count\_y]) \* (ky - 1) / (pow(ky, Ny) - 1);

for (int p = 0; p < Ny - 1; curr\_count\_y++, p++)

{

all\_Y[curr\_count\_y + 1] = all\_Y[curr\_count\_y] + hy \* pow(ky, p);

}

curr\_count\_y++;

}

all\_Y[curr\_count\_y] = Y;

}

for (int curr\_count\_z = 0; curr\_count\_z < count\_z - 1; )

{

in >> Z >> Nz >> kz;

double hy;

if (kz == 1)

{

hy = (Z - all\_Z[curr\_count\_z]) / Nz;

for (int p = 1; p < Nz; p++)

{

all\_Z[curr\_count\_z + p] = all\_Z[curr\_count\_z] + hy \* p;

}

curr\_count\_z += Nz;

}

else

{

hy = (Z - all\_Z[curr\_count\_z]) \* (kz - 1) / (pow(kz, Nz) - 1);

for (int p = 0; p < Nz - 1; curr\_count\_z++, p++)

{

all\_Z[curr\_count\_z + 1] = all\_Z[curr\_count\_z] + hy \* pow(kz, p);

}

curr\_count\_z++;

}

all\_Z[curr\_count\_z] = Z;

}

in.close();

out.open("xyz.txt");

// ребра параллельные х

for (int i = 0; i < count\_z; i++)

{

for (int j = 0; j < count\_x - 1 ; j++)

{

for (int k = 0; k < count\_y; k++)

{

out << all\_X[j] << "\t" << all\_Y[k] << "\t" << all\_Z[i] << "\t"

<< all\_X[j + 1] << "\t" << all\_Y[k] << "\t" << all\_Z[i] << "\n";

}

}

}

// ребра параллельные y

for (int i = 0; i < count\_z; i++)

{

for (int j = 0; j < count\_y - 1; j++)

{

for (int k = 0; k < count\_x; k++)

{

out << all\_X[k] << "\t" << all\_Y[j] << "\t" << all\_Z[i] << "\t"

<< all\_X[k] << "\t" << all\_Y[j+1] << "\t" << all\_Z[i] << "\n";

}

}

}

// ребра параллельные z

for (int i = 0; i < count\_z - 1; i++)

{

for (int k = 0; k < count\_y; k++)

{

for (int j = 0; j < count\_x; j++)

{

out << all\_X[j] << "\t" << all\_Y[k] << "\t" << all\_Z[i] << "\t"

<< all\_X[j] << "\t" << all\_Y[k] << "\t" << all\_Z[i+1] << "\n";

}

}

}

out.close();

// input area

// mu, sigma same in all area

int shift\_y = count\_z \* count\_y \* (count\_x - 1),

shift\_z = shift\_y + count\_z \* count\_x \* (count\_y - 1);

out.open("elem.txt");

for (int i = 0; i < count\_z - 1; i++)

{

for (int j = 0; j < count\_x - 1; j++)

{

for (int k = 0; k < count\_y - 1; k++)

{

out << i \* count\_y \* (count\_x - 1) + j \* count\_y + k << " "

<< i \* count\_y \* (count\_x - 1) + j \* count\_y + k + 1 << " "

<< (i + 1) \* count\_y \* (count\_x - 1) + j \* count\_y + k << " "

<< (i + 1) \* count\_y \* (count\_x - 1) + j \* count\_y + k + 1 << " "

<< shift\_y + i \* count\_x \* (count\_y - 1) + j + k \* count\_x << " "

<< shift\_y + i \* count\_x \* (count\_y - 1) + j + 1 + k \* count\_x << " "

<< shift\_y + (i + 1) \* count\_x \* (count\_y - 1) + j + k \* count\_x << " "

<< shift\_y + (i + 1) \* count\_x \* (count\_y - 1) + j + 1 + k \* count\_x << " "

<< shift\_z + i \* count\_y \* count\_x + k \* count\_x + j << " "

<< shift\_z + i \* count\_y \* count\_x + k \* count\_x + 1 + j << " "

<< shift\_z + i \* count\_y \* count\_x + (k + 1) \* count\_x + j << " "

<< shift\_z + i \* count\_y \* count\_x + (k + 1) \* count\_x + 1 + j

<< " 0 0 \n";

}

}

}

out.close();

// bounder

out.open("S1.txt");

// параллельно х

out << count\_y\*(count\_x - 1)\*2 + (count\_z - 2) \* (count\_x - 1) \* 2 << "\n";

// верх - низ

for (int k = 0; k < count\_y; k++)

{

for (int j = 0; j < count\_x - 1; j++)

{

out << (count\_z - 1) \* count\_y \* (count\_x - 1) + j \* count\_y + k << " " << j \* count\_y + k << " ";

}

}

// лево - право

for (int i = 1; i < count\_z - 1; i++)

{

for (int j = 0; j < count\_x - 1; j++)

{

out << i \* count\_y \* (count\_x - 1) + j \* count\_y << " " << i \* count\_y \* (count\_x - 1) + (j + 1) \* count\_y - 1 << " ";

}

}

// параллельно у

out << "\n" << 2\*count\_x \* (count\_y - 1) + 2\*(count\_z - 2) \* (count\_y - 1) << "\n";

// верх - низ

for (int k = 0; k < count\_y - 1; k++)

{

for (int j = 0; j < count\_x; j++)

{

out << shift\_y + (count\_z - 1) \* count\_x \* (count\_y - 1) + k \* count\_x + j << " "

<< shift\_y + k \* count\_x + j << " ";

}

}

// перед - зад

for (int i = 1; i < count\_z - 1; i++)

{

for (int k = 0; k < count\_y - 1; k++)

{

out << shift\_y + i \* count\_x \* (count\_y - 1) + k \* count\_x << " "

<< shift\_y + i \* count\_x \* (count\_y - 1) + (k + 1) \* count\_x - 1 << " ";

}

}

// параллельно z

out << "\n" << 2\*(count\_z - 1) \* count\_x + 2\*(count\_z - 1) \* (count\_y - 2) << "\n";

// лево - право

for (int i = 0; i < count\_z - 1; i++)

{

for (int j = 0; j < count\_x; j++)

{

out << shift\_z + i \* count\_y \* count\_x + j << " "

<< shift\_z + i \* count\_y \* count\_x + count\_x \* (count\_y - 1) + j << " ";

}

}

// перед - зад

for (int i = 0; i < count\_z - 1; i++)

{

for (int k = 1; k < count\_y - 1; k++)

{

out << shift\_z + i \* count\_y \* count\_x + k \* count\_x << " "

<< shift\_z + i \* count\_y \* count\_x + (k + 1) \* count\_x - 1 << " ";

}

}

out.close();

out.open("info.txt");

out << (count\_x - 1) \* count\_y \* count\_z + count\_x \* (count\_y - 1) \* count\_z + count\_x \* count\_y \* (count\_z - 1) << " 1 "

<< (count\_x - 1)\* (count\_y - 1)\* (count\_z - 1) << " 3";

out.close();

out.open("material.txt");

out << "1 1";

out.close();

}

void Create\_time\_grid()

{

std::ofstream out;

out.precision(15);

std::ifstream in;

// time grid

in.open("time\_grid.txt");

std::vector<double> time\_grid;

double T, kt;

int Nt;

int count\_t;

in >> count\_t;

time\_grid.resize(count\_t);

in >> time\_grid[0];

for (int curr\_count\_t = 0; curr\_count\_t < count\_t - 1; )

{

in >> T >> Nt >> kt;

double ht;

if (kt == 1)

{

ht = (T - time\_grid[curr\_count\_t]) / Nt;

for (int p = 1; p < Nt; p++)

{

time\_grid[curr\_count\_t + p] = time\_grid[curr\_count\_t] + ht \* p;

}

curr\_count\_t += Nt;

}

else

{

ht = (T - time\_grid[curr\_count\_t]) \* (kt - 1) / (pow(kt, Nt) - 1);

double pow\_kt = 1;

for (int p = 0; p < Nt - 1; curr\_count\_t++, p++)

{

time\_grid[curr\_count\_t + 1] = time\_grid[curr\_count\_t] + ht \* pow\_kt;

pow\_kt \*= kt;

}

curr\_count\_t++;

}

time\_grid[curr\_count\_t] = T;

}

in.close();

out.open("time.txt");

out << time\_grid.size() << "\n";

for (int i = 0; i < count\_t; i++)

{

out << time\_grid[i] << " ";

}

out.close();

}

**Generate.h**

#pragma once

#include <vector>

#include <fstream>

void Make\_grid(std::string path);

void Create\_time\_grid();

**Matrix.cpp**

#include "Matrix.h"

Matrix::Matrix(int \_N)

{

matrix.resize(\_N);

for (int i = 0; i < \_N; i++)

matrix[i].resize(\_N);

N = \_N;

}

Matrix::Matrix()

{

matrix.resize(4);

for (int i = 0; i < 4; i++)

matrix[i].resize(4);

N = 4;

}

Matrix Matrix::operator+(Matrix B)

{

if (B.N != N)

return -1;

Matrix C(N);

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

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

C.matrix[i][j] = matrix[i][j] + B.matrix[i][j];

return C;

}

Matrix Matrix::operator\*(double a)

{

Matrix C(N);

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

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

C.matrix[i][j]=a\*matrix[i][j];

return C;

}

**Matrix.h**

#pragma once

#include <vector>

class Matrix

{

public:

int N;

std::vector<std::vector<double>> matrix;

Matrix operator \* (double a);

Matrix operator + (Matrix B);

Matrix(int \_N);

Matrix(); // matrix 4x4

}