Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Курсовая работа

по уравнениям математической физики

Факультет: ПМИ

Группа: ПМ-01

Студент: Ряховский М.И.

Варианты: 81, 9

Преподаватель: Персова М.Г.

Новосибирск

2013

Содержание

[1. Постановка задачи 3](#_Toc356408652)

[2. Теоретическая часть 3](#_Toc356408653)

[2.1 Дискретизация по времени 3](#_Toc356408654)

[2.2 Вариационная постановка 4](#_Toc356408655)

[2.3 Конечноэлементная дискретизация 5](#_Toc356408656)

[2.4 Локальные матрицы и вектора конечных элементов 5](#_Toc356408657)

[2.5 Локальные матрицы и вектора граней, на которых заданы краевые условия второго и третьего рода 6](#_Toc356408658)

[3. Описание разработанной программы 7](#_Toc356408659)

[3.1 Структуры данных, используемые для задания расчётной области и конечноэлементной сетки 7](#_Toc356408660)

[3.2 Структура основных модулей программы 8](#_Toc356408661)

[3.2.1. base\_classes 8](#_Toc356408662)

[3.2.2. additional\_classes 9](#_Toc356408663)

[3.2.3. areas 9](#_Toc356408664)

[3.2.4. CGM 10](#_Toc356408665)

[3.2.5. launch 10](#_Toc356408666)

[4. Тестирование программы 10](#_Toc356408667)

[4.1. Тест 1 10](#_Toc356408668)

[5. Исследования 11](#_Toc356408669)

[5.1 Определение порядка аппроксимации по времени 11](#_Toc356408670)

[5.2 Задача с точечным источником 12](#_Toc356408671)

[6. Выводы 14](#_Toc356408672)

[7. Код программы 14](#_Toc356408673)

[Файл «base\_classes.h» 14](#_Toc356408674)

[Файл «additional\_classes.h» 14](#_Toc356408675)

[Файл «areas.h» 15](#_Toc356408676)

[Файл «CGM.h» 15](#_Toc356408677)

[Файл «Gauss.h» 15](#_Toc356408678)

[Файл «base\_clasess.cpp» 16](#_Toc356408679)

[Файл «additional\_classes.cpp» 20](#_Toc356408680)

[Файл «areas.cpp» 21](#_Toc356408681)

[Файл «CGM.cpp» 22](#_Toc356408682)

[Файл «Gauss.cpp» 23](#_Toc356408683)

[Файл «launch.cpp» 23](#_Toc356408684)

# Постановка задачи

Решить методом конечных элементов трёхмерную гиперболическую задачу в декартовых координатах. Вид уравнения:

, 

 (коэффициент диффузии) – кусочно-постоянная функция;

 - некоторая известная функция трёх переменных;

 - некоторая известная функция трёх переменных;

 - некоторая известная функция трёх переменных;

 - правая часть задана в виде производной некоторой скалярной функции с известными значениями в узлах.

Краевые условия всех типов:







Вид разностной схемы по времени: неявная, четырёхслойная.

Вид конечных элементов: тетраэдры.

Вид базисных функций: линейные.

Формат хранения матрицы СЛАУ: разреженный строчно-столбцевой.

Метод решения СЛАУ: метод сопряженных градиентов или локально-оптимальная схема с неполной факторизацией.

# Теоретическая часть

## Дискретизация по времени

Введём сетку по времени, разбив необходимый отрезок точками . Введём обозначение для шага по времени: . Рассмотрим отрезок  и представим на нём нашу функцию в виде полиномов Лагранжа:



Тогда производные  и  можно представить в виде:





Для нахождения коэффициентов неявной схемы рассмотрим значение производных в точке . Обозначим коэффициенты для аппроксимации первой производной, как для аппроксимации второй производной, как :

















После подставленные разностной схемы в уравнение, оно преобразуется в:



Для краткости выкладок обозначим правую часть за, а 

## Вариационная постановка

Выполним вариационную постановку методом Бубнова-Галёркина. Введём гильбертово пространство . В нём скалярное произведение определяется следующим образом:



И норма, ассоциированная со скалярным произведением: .

В общем виде постановка Бубнова-Галёркина для операторного уравнения записывается в следующем виде:



Где . Для уравнения постановка примет вид:



Применив формулу Грина к этому выражению получим:



## Конечноэлементная дискретизация

Разобьём область  на непересекающиеся подобласти – конечные элементы: . В соответствии с заданием  - это тетраэдры. А граничные поверхности  и представим в виде объединения соответствующих граней тетраэдров: .Тогда формулу можно переписать в виде:

В области выберем линейный финитный базис , обладающий следующими свойствами:

– Базисная функция ассоциирована с узлом конечноэлементной сетки , причём: 

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

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

## Локальные матрицы и вектора конечных элементов

В качестве локальных базисных функций на элементе  выберем L-координаты соответствующего тетраэдра, получим четыре базисных функции на каждом конечном элементе: . И для конечного элемента элементы локальной матрицы жёсткости примут вид:



Поскольку функции  линейны, то их мы можем записать в виде:

, тогда соотношение для нахождения коэффициентов можно записать в матричном виде:, где

 - единичная матрица

, ,  - координаты узлов элемента

В этих обозначениях элементы локальной матрицы жёсткости будут вычисляться по следующему правилу:



Используя соотношение для интегралов от L-координат:



Получаем итоговый результат:



Элементы локальной матрицы жесткости будут вычисляться по следующей формуле:



Используя соотношение получим матрицу:

, 

Правая часть , интерполируем функцию  по базисным функциям, получим: , тогда , а остальные слагаемые есть матрица массы умноженная на вектор старого решения:



## Локальные матрицы и вектора граней, на которых заданы краевые условия второго и третьего рода

Грани конечных элементов – это треугольники, аналогично с конечными элементами введём на них линейный базис, используя L-координаты треугольника. Проведя аналогичные выкладки и используя аналог соотношения для треугольников, получим следующие локальные матрицы и вектора:

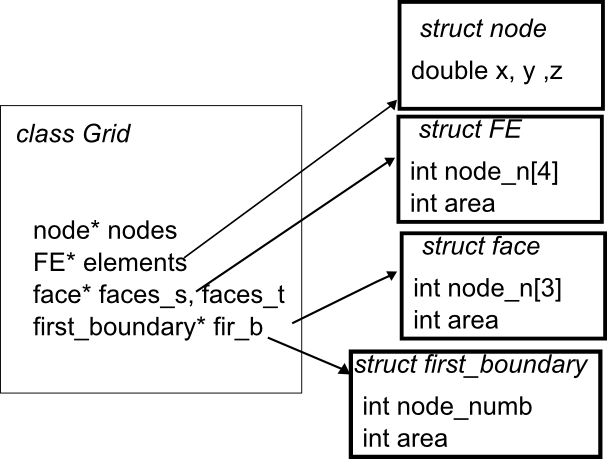
, 



# Описание разработанной программы

## Структуры данных, используемые для задания расчётной области и конечноэлементной сетки

Внутри программы конечно элементная сетка хранится в соответствии со следующей схемой:



Структура node определяет узел и хранит координаты узла.

Структура FE определяет конечный элемент и хранит координаты узлов конечного элемента и номер подобласти, которой принадлежит конечный элемент.

Структура face аналогично определяет ребро конечного элемента.

Структура first\_boundary определяет узлы, в которых заданы первые краевые условия и хранит номер узла и номер значения краевого условия.

Данные о значениях  и , функций ,  и хранятся внутри программы и соотносятся с элементами и гранями по номеру подобласти.

Следующая таблица задаёт соответствие между файлами и данными содержащимися в них:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Данные | Описание способа задания данных | Пример |
| nodes.txt | узлы сетки | сначала задаётся число узлов, затем в каждой новой строке перечисляются их координаты | 6  0 0 1  0 1 1  1 0 1  0 0 0  0 1 0  1 0 0 |
| el.txt | конечные элементы | сначала задаётся количество кэ, затем по строкам перечисляются узлы элементов | 3  1 3 4 5  0 1 2 5  0 1 3 5 |
| el\_areas.txt | подобласти кэ | в каждой строке сначала номер кэ, затем номер подобласти | 0 0  1 0  2 0 |
| faces\_s.txt | грани, где заданы вторые краевые условия | сначала задаётся количество граней, затем с каждой новой строки перечисляются узлы грани | 4  0 1 2  3 4 5  1 5 4  1 2 5 |
| fases\_t.txt | грани, где заданы третьи краевые условия | аналогично с предыдущим | 4  0 1 3  3 1 4  0 2 5  0 3 5 |
| faces\_s\_ar.txt | подобласти граней со вторым краевыми условиями | в каждой строке сначала номер грани, затем номер подобласти | 0 0  1 0  2 1  3 1 |
| faces\_t\_ar.txt | подобласти граней с третьими краевыми условиями | аналогично с предыдущим | 0 0  1 0  2 1  3 1 |
| first\_bound.txt | подобласти узлов, где заданы первые краевые условия | сначала количество узлов, затем в каждой строке номер узла, затем номер подобласти | 3  2 1  3 0  4 1 |
| time data.txt | изменение времени | сначала начальное время, затем минимальный шаг и коэффициент разрядки | 0 1 1.1 |
| time0.txt, time1.txt, time2.txt | начальные условия | в каждой строке файла хранится значения веса в один из начальных моментов времени. Чем меньше номер файла тем больше время. | 1  2  3  2  1 |

## Структура основных модулей программы

Основные модули программы:

* *base\_classes* – описание и реализация основного класса;
* *addtional\_classes* – описание и реализация дополнительных классов для генерации портрета СЛАУ;
* *areas* – описание тестов;
* *CGM* – описание и реализация класса решателя СЛАУ (метод сопряженных градиентов);
* *launch* – запуск программы, вызов основных подпрограмм;
* *grid\_gen* – генератор сеток.

### base\_classes

* *Описание модуля:* содержит описание и реализацию класса Grid, основного класса программы. Функции класса генерируют локальныt и глобальные матрицы СЛАУ (при наличии портрета). Так же класс хранит решение задачи – полученные веса базисных функций.
* *Основные методы класса:*
  + Функции ввода, которые осуществляют ввод по указанным выше правилам;
  + gen\_SLAE\_port – функция, которая обращается к классу-генератору портрета СЛАУ;
  + gen\_SLAE – функция, генерирующая конечноэлементную СЛАУ, для нового шага по времени;
  + solve\_SLAE – функция, которая обращается к классу-решателю СЛАУ;
  + gen\_local\_el – функция, генерирующая локальную матрицу и вектор СЛАУ, ей на вход подаётся адрес матрицы и ветора, а так же номер элемента;
  + gen\_local\_thi – функция, генерирующая локальную матрицу и вектор правой части для граней с третьими краевыми условиями, ей на вход падётся адрес матрицы и вектора и номер грани;
  + gen\_local\_sec – функция, генерирующая локальный векор правой части для граней со вторыми краевыми условяии, на вход ей подаётся адрес вектора и номер грани;
  + u\_in\_el – расчёт функции-решения в произвольной точке (расчёт производится для функции на текущем временом слое).

### additional\_classes

* *Описание модуля:* содержит классы, реализующие генерацию портрета СЛАУ и взаимодействие этого класса с классом Grid.
* *Класс SLAE\_port\_list:* реализует упорядоченный список, который содержит функции добавления элемента в список, номер списка и подсчёт элементов. Каждый список ассоциирован со своим узлом и со строкой матрицы и хранит элементы массива gj для этой строки.
* *Класс SLAE\_port\_gen:* класс предназначен для генерации портрета СЛАУ и хранит массив из элементов класса SLAE\_port\_list.

*Основные методы класса:*

* + init – инициализация массив lists, составленного из элементов класса SLAE\_port\_list. На вход подаётся количество узлов;
  + add\_el – добавление элементов в списки lists. На вход подаётся конечный элемент;
  + gen – генерация портрета СЛАУ, на вход подаются адреса массивов gi и gj, функция заполняет эти массивы и возвращает число элементов в массиве gj.

### areas

* *Описание модуля:* содержит множество namespace’ов, в которых определенны параметры тестов. Все namespace имеют одинаковую структуру. Так же содержит описание вспомогательного типа func:

typedef double(\*func)(double, double, double);

* *Структура namespace:*
  + string path – имя папки, где лежат входные файлы для теста;
  + int lambda\_n, u\_betta\_n, tetta\_n, fir\_n – количество областей конечных элементов, граней с третьими краевыми, граней со вторыми краевыми и узлов с первыми краевыми условиями;
  + double \*lambda\_values – массив значений ;
  + double \*betta\_values – массив значений 
  + func \*fir\_values – массив указателей на функции для первых краевых условий;
  + func \*gamma\_values – массив указателей на функции, определяющих функцию в различных подобластях;
  + func \*u\_betta\_values – массив указателй на функции, определяющих функцию  в разлиных подобластях;
  + func \*tetta\_values – массив указателй на функции, определяющих функцию  в разлиных подобластях, сами функции указнные в предудищх пунктах определются внутри namespace’а;
  + void init\_area() – функция, инициализирующая namespace, создающая указанные массивы и задающая им значения.

### CGM

* *Описание модуля:* содержит класс CGM, который выполняет решение СЛАУ с симметричной матрицей в разреженном формате с помощью метода сопряженных градиентов с неполной  факторизацией.
* *Основные методы класса:*
  + init – инициализация основных массивов. Массивы gi, gj, di, gg передаются в эту функцию по адресу, так же передаётся правая часть;
  + make\_LLT\_decomposition – построение неполного  разложения;
  + mul\_matrix – перемножение разреженной симметричной матрицы на вектор. Входные данные – вектор, который умножаем, выходные – результат произведения;
  + solve\_LLT – решение СЛАУ вида;
  + dot\_prod – расчёт евклидова скалярного произведия (как сумма произведений соответсвующих координат). Входные данные – два вектора, результат – вещественное число;
  + solve – функция, реализующая МСГ с неполной  факторизацией. На вход подаётся адрес массива, куда записывается решение СЛАУ.

### launch

* *Описание модуля*: содержит подпрограмму main, в которой запускаются основные функции класса Grid, так же модуль содержит вывод результата в файл result.txt

# Тестирование программы

## Тест 1

* *Цель теста:* аппроксимацию по времени
* *Параметры теста:*

, , , ,,

, , 

|  |  |  |  |
| --- | --- | --- | --- |
| Номер узла | x | y | z |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0.1 | 0.1 | 0.1 |

|  |  |
| --- | --- |
| Номер элемента | узлы |
| 0 | 0 1 2 4 |
| 1 | 0 1 3 4 |
| 2 | 0 2 3 4 |
| 3 | 1 2 3 4 |

Краевые условия: первого рода

|  |  |
| --- | --- |
|  | Относительная погрешность |
| 3 | 3.46E-33 |
| 4 | 0.00E+00 |
| 5 | 2.59E-33 |
| 6 | 3.46E-33 |
| 7 | 5.49E-33 |

# Исследования

## Определение порядка аппроксимации по времени

* *Параметры теста:*

,  ,, , , 

, ,,, 

|  |  |  |  |
| --- | --- | --- | --- |
| Номер узла | x | y | z |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0.1 | 0.1 | 0.1 |

|  |  |
| --- | --- |
| Номер элемента | узлы |
| 0 | 0 1 2 4 |
| 1 | 0 1 3 4 |
| 2 | 0 2 3 4 |
| 3 | 1 2 3 4 |

Краевые условия: первого рода

При 

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг по времени |  |  |  |
| Относительная погрешность | 5.60E-03 | 1.25E-03 | 1.21E-04 |

Оценка порядка аппроксимации: 

## Задача с точечным источником

* Уравнение:



* Начальные условия однородные. Начальное время . Краевые условия: однородные, первого рода.
* Сетка: получена из параллелепипидальной: разбиение параллелепипеда с шагом  и коэффициентом разрядки .

Узлов: 216, элементов: 750.

* Источник: , узел сетки номер 86

**Тест 1**

, , 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер узла | Время, нс | | | |
| - | 30 | 40 | 50 | 120 |
| 79 | -0.000000000000006 | -0.000000000000022 | -0.000000000000049 | -0.000000000000580 |
| 80 | -0.000000000000007 | -0.000000000000023 | -0.000000000000051 | -0.000000000000605 |
| 81 | 0.000000000000000 | 0.000000000000002 | 0.000000000000003 | 0.000000000000040 |
| 82 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000001 |
| 83 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 84 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 85 | -0.000000000000003 | -0.000000000000011 | -0.000000000000025 | -0.000000000000299 |
| 86 | 0.000000000000071 | 0.000000000000248 | 0.000000000000548 | 0.000000000006508 |
| 87 | -0.000000000000003 | -0.000000000000012 | -0.000000000000027 | -0.000000000000320 |
| 88 | 0.000000000000000 | 0.000000000000000 | 0.000000000000001 | 0.000000000000013 |
| 89 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 90 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 91 | 0.000000000000000 | 0.000000000000002 | 0.000000000000004 | 0.000000000000043 |
| 92 | -0.000000000000007 | -0.000000000000024 | -0.000000000000053 | -0.000000000000632 |
| 93 | -0.000000000000007 | -0.000000000000024 | -0.000000000000053 | -0.000000000000626 |

**Тест 2**

, , 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер узла | Время, нс | | | |
| - | 30 | 40 | 50 | 120 |
| 79 | -0.000000000000063 | -0.000000000000221 | -0.000000000000489 | -0.000000000005802 |
| 80 | -0.000000000000066 | -0.000000000000230 | -0.000000000000510 | -0.000000000006050 |
| 81 | 0.000000000000004 | 0.000000000000015 | 0.000000000000034 | 0.000000000000401 |
| 82 | 0.000000000000000 | 0.000000000000001 | 0.000000000000001 | 0.000000000000014 |
| 83 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 84 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 85 | -0.000000000000032 | -0.000000000000114 | -0.000000000000252 | -0.000000000002986 |
| 86 | 0.000000000000707 | 0.000000000002476 | 0.000000000005482 | 0.000000000065077 |
| 87 | -0.000000000000035 | -0.000000000000122 | -0.000000000000269 | -0.000000000003197 |
| 88 | 0.000000000000001 | 0.000000000000005 | 0.000000000000011 | 0.000000000000128 |
| 89 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 90 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 91 | 0.000000000000005 | 0.000000000000017 | 0.000000000000037 | 0.000000000000434 |
| 92 | -0.000000000000069 | -0.000000000000241 | -0.000000000000533 | -0.000000000006325 |
| 93 | -0.000000000000068 | -0.000000000000238 | -0.000000000000527 | -0.000000000006256 |

**Тест 3**

, , 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер узла | Время, нс | | | |
| - | 30 | 40 | 50 | 120 |
| 79 | -0.000000000000001 | -0.000000000000002 | -0.000000000000005 | -0.000000000000058 |
| 80 | -0.000000000000001 | -0.000000000000002 | -0.000000000000005 | -0.000000000000060 |
| 81 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000004 |
| 82 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 83 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 84 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 85 | 0.000000000000000 | -0.000000000000001 | -0.000000000000003 | -0.000000000000030 |
| 86 | 0.000000000000007 | 0.000000000000025 | 0.000000000000055 | 0.000000000000651 |
| 87 | 0.000000000000000 | -0.000000000000001 | -0.000000000000003 | -0.000000000000032 |
| 88 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000001 |
| 89 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 90 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 91 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000004 |
| 92 | -0.000000000000001 | -0.000000000000002 | -0.000000000000005 | -0.000000000000063 |
| 93 | -0.000000000000001 | -0.000000000000002 | -0.000000000000005 | -0.000000000000063 |

**Тест 4**

, , 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер узла | Время, нс | | | |
| - | 6 | 8 | 10 | 12 |
| 79 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 | -0.000000000000004 |
| 80 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 | -0.000000000000004 |
| 81 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 82 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 83 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 84 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 85 | 0.000000000000000 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 |
| 86 | 0.000000000000003 | 0.000000000000010 | 0.000000000000022 | 0.000000000000039 |
| 87 | 0.000000000000000 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 |
| 88 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 89 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 90 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 91 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 | 0.000000000000000 |
| 92 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 | -0.000000000000004 |
| 93 | 0.000000000000000 | -0.000000000000001 | -0.000000000000002 | -0.000000000000004 |

# Выводы

Для построенной разностной схемы фактический и теоретический порядок аппроксимации совпадают. В зависимости от поведения функции решения и отрезка времени схема может давать как лучшую, так и худшую аппроксимацию.

Скорость распространения колебаний в задаче с точечным источником зависит от скорости волны - чем больше скорость, тем быстрее распространение и от интенсивности источника – чем сильнее источник, тем быстрее распространяется волна.

# Код программы

## Файл «base\_classes.h»

#pragma once

#include <stdio.h>

#include "additional\_classes.h"

#include "areas.h"

#include "CGM.h"

#include "Gauss.h"

using namespace area\_3;

struct first\_boundary{

int node\_numb;

int area;

};

//Класс для решения гиперболиечского уравнения

//Дискретизация по времени: неявная 4-х слойная схема

//Базисные функции: линейные на тетраэдрах

class Grid{

public:

Grid();

//Ввод

void input\_nodes(FILE\* nodes\_f);

void input\_elements(FILE\* el\_f);

void input\_faces\_s(FILE\* face\_f\_sec);

void input\_faces\_t(FILE\* faces\_f\_thi);

void input\_fir\_nodes(FILE \*fir\_f);

void input\_time\_data(FILE\* time\_d); //ввод информации о времени: начальный шаг и коэффциент разрядки

void input\_start(FILE\* st0, FILE\* st1, FILE\* st2); //Ввод начальных условий на i-2, i-1 и i-ой итерациях

void input\_el\_areas(FILE \*el\_a\_f);

void input\_f\_s\_areas(FILE \*fs\_a\_f);

void input\_f\_t\_areas(FILE \*ft\_a\_f);

double u\_in\_el(double x, double y, double z); //получение значения функции с текущего временого слоя в точке

void gen\_SLAE\_port(); //запуск генерации портрета СЛАУ

void gen\_SLAE(); //запуск генерации СЛАУ

void solve\_SLAE(); //запуск решения СЛАУ

void init\_areas(FILE\* elem\_areas, FILE\* fir\_areas, FILE\* sec\_areas, FILE\* thi\_areas);

void output\_weight(FILE\* outp\_f); //вывод текущего решения

void output\_diff(FILE\* outp\_f); //вывод текущей погрешности

private:

node\* nodes; //массив узлов

FE\* elements; //массив элементов

face \*faces\_s, \*faces\_t; //массивы для вторых и третьих краевых

first\_boundary\* fir\_b; //массив для первых краевых

double \*weight\_0; //веса с i-ой итерации

double \*weight\_1; //веса с (i-1)-ой итерации

double \*weight\_2; //веса с (i-2)-ой итерации

//Соответсвующие весам времена

double t0;

double t1;

double t2;

double t\_new; //время на i+1 итерации

double k\_time; //коэффциент разрядки по времени

int nodes\_number, elements\_number, faces\_number\_sec, faces\_number\_thi, first\_number; //количество узлов, элементов и каждого из краевых условий

//Массивы для матрицы

int \*gi, \*gj;

double \*gg, \*di, \*right\_part;

int SLAE\_el\_numbers;

SLAE\_port\_gen SLAE\_gen; //класс для генерации портрета СЛАУ

//Получение локальных матриц указанного элемента или грани

void gen\_local\_el(double \*\*A\_loc, double \*b\_loc, int el\_n);

void gen\_local\_thi(double \*\*A\_loc, double \*b\_loc, int f\_n);

void gen\_local\_sec(double \*b\_loc, int f\_n);

//Получение значения коэффицента на элементе

double get\_lambda\_value(int el\_n);

double get\_gamma\_value(int el\_n);

double get\_sigma\_value(int el\_n, double t);

double get\_epsilon\_value(int el\_n, double t);

double get\_rp\_value(int node\_n);

//Получение значения коэффициента на грани

double get\_betta\_value(int f\_n);

double get\_u\_betta\_value(int f\_n, int node\_n);

double get\_tetta\_value(int f\_n, int node\_n);

//Получение значения первого краевого в узле

double get\_func\_value\_fir(int node\_n);

//Получение площади грани

double get\_face\_mes(int boun, int f\_n);

//Нахождение номера элемента СЛАУ в указаном предстовлении

int find\_el\_pos(int i, int j);

//вспомогетельные матрицы для формирования локальных

double \*\*D, \*\*alpha;

};

## Файл «additional\_classes.h»

#pragma once

#include <math.h>

//узел

struct node{

double x, y, z;

int number;

};

//Тетраэдр

struct FE{

int node\_n[4];

int number;

int area;

};

//Грань

struct face{

int node\_n[3];

int number;

int area;

};

//элемент списка для геренации портрета СЛАУ

struct gen\_l\_el{

int value;

gen\_l\_el\* next;

};

class SLAE\_port\_list{

public:

SLAE\_port\_list();

~SLAE\_port\_list();

void add\_el(FE &el\_a);

void init();

void set\_num(int s\_num);

int size();

int size\_before(int n);

int take\_and\_next();

void cash\_off();

int get\_m\_size();

private:

void exclude\_last\_el();

void add(int val);

gen\_l\_el \*begin, \*end, \*cash;

int l\_size;

int m\_size;

int number\_of\_line;

};

class SLAE\_port\_gen{

public:

SLAE\_port\_gen();

~SLAE\_port\_gen();

void init(int nodes\_n);

void add\_el(FE& el\_a);

void gen(int \*gi, int \*&gj, int &m);

private:

int n;

SLAE\_port\_list\* lists;

};

## Файл «areas.h»

#pragma once

#include <math.h>

#include <string>

typedef double(\*func)(double, double, double, double);

using namespace std;

namespace area\_1{

extern string path; //путь папке с данными

extern int lambda\_n, betta\_n, u\_betta\_n, tetta\_n, fir\_n; //количество тех или иных данных

//Сами данные

extern double \*lambda\_values;

extern double \*betta\_values;

extern func \*gamma\_values;

extern func \*sigma\_values;

extern func \*epsilon\_values;

extern func \*u\_betta\_values;

extern func \*tetta\_values;

extern func \*fir\_values;

double sol\_func(double x, double y, double z, double t); //решение

double equation\_rp(double x, double y, double z, double t); //правая часть уравнения(точнее функция V(x,y,z,t), f = dV/dx)

void init\_area(); //инициализировать данные

};

namespace area\_2{

extern string path; //путь папке с данными

extern int lambda\_n, betta\_n, u\_betta\_n, tetta\_n, fir\_n; //количество тех или иных данных

//Сами данные

extern double \*lambda\_values;

extern double \*betta\_values;

extern func \*gamma\_values;

extern func \*sigma\_values;

extern func \*epsilon\_values;

extern func \*u\_betta\_values;

extern func \*tetta\_values;

extern func \*fir\_values;

double sol\_func(double x, double y, double z, double t); //решение

double equation\_rp(double x, double y, double z, double t); //правая часть уравнения(точнее функция V(x,y,z,t), f = dV/dx)

void init\_area(); //инициализировать данные

};

namespace area\_3{

extern string path; //путь папке с данными

extern int lambda\_n, betta\_n, u\_betta\_n, tetta\_n, fir\_n; //количество тех или иных данных

//Сами данные

extern double \*lambda\_values;

extern double \*betta\_values;

extern func \*gamma\_values;

extern func \*sigma\_values;

extern func \*epsilon\_values;

extern func \*u\_betta\_values;

extern func \*tetta\_values;

extern func \*fir\_values;

double sol\_func(double x, double y, double z, double t); //решение

double equation\_rp(double x, double y, double z, double t); //правая часть уравнения(точнее функция V(x,y,z,t), f = dV/dx)

void init\_area(); //инициализировать данные

};

## Файл «CGM.h»

#pragma once

#include <math.h>

class CGM{

public:

void init(int \*gi\_s, int \*gj\_s, double \*di\_s, double \*gg\_s, double \*rp\_s, int n\_s);

void solve(double \*&solution);

private:

void make\_LLT\_decomposition();

void mul\_matrix(double \*f, double \*&x);

void solve\_L(double \*f, double \*&x);

void solve\_LT(double \*f, double \*&x);

void solve\_LLT(double \*f, double \*&x);

double dot\_prod(double \*a, double \*b);

int n;

int \*gi, \*gj;

double \*di, \*gg, \*rp, \*r, \*x0, \*z, \*p, \*s;

double \*L\_di, \*L\_gg;

};

## Файл «Gauss.h»

#pragma once

#include <math.h>

void prepared(int n);

void solve\_Gauss(double \*\*A, double \*x, int n); //решение СЛАУ, метод Гаусса

void trianglematrix1(double \*\*A, double \*x, int n); //Приведение матрицы к верхнему треугольному виду

void transf1(double \*\*A, double \*x, int i,int n); // перестановка строк

## Файл «base\_clasess.cpp»

#include "base\_classes.h"

Grid::Grid(){

nodes\_number = elements\_number = faces\_number\_sec = faces\_number\_thi = first\_number = 0;

}

void Grid::input\_nodes(FILE \*nodes\_f){

fscanf(nodes\_f,"%d",&nodes\_number);

nodes = new node [nodes\_number];

for(int i = 0; i < nodes\_number; i++){

fscanf(nodes\_f,"%lf %lf %lf", &nodes[i].x, &nodes[i].y, &nodes[i].z);

nodes[i].number = i;

}

weight\_0 = new double [nodes\_number];

weight\_1 = new double [nodes\_number];

weight\_2 = new double [nodes\_number];

}

void Grid::input\_elements(FILE \*el\_f){

fscanf(el\_f,"%d",&elements\_number);

elements = new FE [elements\_number];

for(int i = 0; i < elements\_number; i++){

fscanf(el\_f, "%d %d %d %d", &elements[i].node\_n[0], &elements[i].node\_n[1], &elements[i].node\_n[2], &elements[i].node\_n[3]);

elements[i].number = i;

}

}

void Grid::input\_faces\_s(FILE\* face\_f\_sec){

fscanf(face\_f\_sec, "%d", &faces\_number\_sec);

faces\_s = new face [faces\_number\_sec];

for(int i = 0; i < faces\_number\_sec; i++){

fscanf(face\_f\_sec, "%d %d %d",&faces\_s[i].node\_n[0],&faces\_s[i].node\_n[1], &faces\_s[i].node\_n[2]);

faces\_s[i].number = i;

}

}

void Grid::input\_faces\_t(FILE\* face\_f\_thi){

fscanf(face\_f\_thi, "%d", &faces\_number\_thi);

faces\_t = new face [faces\_number\_thi];

for(int i = 0; i < faces\_number\_thi; i++){

fscanf(face\_f\_thi, "%d %d %d",&faces\_t[i].node\_n[0],&faces\_t[i].node\_n[1], &faces\_t[i].node\_n[2]);

faces\_t[i].number = i;

}

}

void Grid::input\_fir\_nodes(FILE \*fir\_f){

int node\_n;

fscanf(fir\_f, "%d", &first\_number);

fir\_b = new first\_boundary [first\_number];

for(int i = 0; i < first\_number; i++)

fscanf(fir\_f, "%d %d", &fir\_b[i].node\_numb, &fir\_b[i].area);

}

void Grid::input\_el\_areas(FILE \*el\_a\_f){

int el\_n, ar\_n;

for(int i = 0; i < elements\_number; i++){

fscanf(el\_a\_f, "%d %d",&el\_n, &ar\_n);

elements[el\_n].area = ar\_n;

}

}

void Grid::input\_f\_s\_areas(FILE\* fs\_a\_f){

int f\_n, ar\_n;

for(int i = 0; i < faces\_number\_sec; i++){

fscanf(fs\_a\_f, "%d %d",&f\_n, &ar\_n);

faces\_s[f\_n].area = ar\_n;

}

}

void Grid::input\_f\_t\_areas(FILE\* ft\_a\_f){

int f\_n, ar\_n;

for(int i = 0; i < faces\_number\_sec; i++){

fscanf(ft\_a\_f, "%d %d",&f\_n, &ar\_n);

faces\_t[f\_n].area = ar\_n;

}

}

void Grid::input\_start(FILE \*st0, FILE \*st1, FILE \*st2){

for(int i = 0; i < nodes\_number; i++)

fscanf(st0, "%lf", &weight\_0[i]);

for(int i = 0; i < nodes\_number; i++)

fscanf(st1, "%lf", &weight\_1[i]);

for(int i = 0; i < nodes\_number; i++)

fscanf(st2, "%lf", &weight\_2[i]);

}

void Grid::input\_time\_data(FILE \*time\_d){

double t\_start;

double tau0;

fscanf(time\_d, "%lf %lf %lf", &t\_start, &tau0, &k\_time);

t2 = t\_start;

t1 = t2 + tau0;

tau0 \*= k\_time;

t0 = t1 + tau0;

}

void Grid::gen\_SLAE\_port(){

init\_area();

SLAE\_gen.init(nodes\_number);

gi = new int [nodes\_number+1];

for(int i = 0; i < elements\_number; i++)

SLAE\_gen.add\_el(elements[i]);

SLAE\_gen.gen(gi, gj, SLAE\_el\_numbers);

gg = new double [SLAE\_el\_numbers];

di = new double [nodes\_number];

right\_part = new double [nodes\_number];

SLAE\_gen.~SLAE\_port\_gen();

}

void Grid::gen\_SLAE(){

t\_new = t0 + k\_time\*(t0 - t1); //устанавливаем новое время

for(int i = 0; i < SLAE\_el\_numbers; i++)

gg[i] = 0;

for(int i = 0; i < nodes\_number; i++)

right\_part[i] = di[i] = 0;

double \*\*A\_loc, \*b\_loc;

int pos, cur\_row;

A\_loc = new double\* [4];

D = new double\* [4];

alpha = new double\* [4];

b\_loc = new double [4];

prepared(4);

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

A\_loc[i] = new double [4];

D[i] = new double [4];

alpha[i] = new double [4];

}

// Сборка основной матрицы

for(int k = 0; k < elements\_number; k++){

gen\_local\_el(A\_loc, b\_loc, k);

int aaa\_1 = 0;

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

cur\_row = elements[k].node\_n[i];

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

//if(cur\_row > elements[k].node\_n[j]){

pos = find\_el\_pos(cur\_row, elements[k].node\_n[j]);

gg[pos] += A\_loc[i][j];

aaa\_1++;

//}

}

di[cur\_row] += A\_loc[i][i];

right\_part[cur\_row] += b\_loc[i];

}

}

// Учёт третьих краевых условий

for(int k = 0; k < faces\_number\_thi; k++){

gen\_local\_thi(A\_loc, b\_loc, k);

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

cur\_row = faces\_t[k].node\_n[i];

for(int j = 0 ; j < 3 ; j++){

if(cur\_row > faces\_t[k].node\_n[j]){

pos = find\_el\_pos(cur\_row, faces\_t[k].node\_n[j]);

gg[pos] += A\_loc[i][j];

}

}

di[cur\_row] += A\_loc[i][i];

right\_part[cur\_row] += b\_loc[i];

}

}

//Учёт вторых краевых условий

for(int k = 0; k < faces\_number\_sec; k++){

gen\_local\_sec(b\_loc, k);

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

right\_part[faces\_s[k].node\_n[i]] += b\_loc[i];

}

//============== Спец. тест =============

//Учёт точненого источника

right\_part[86] += 20.0;

//Учёт первых краевых условий

for(int k = 0; k < first\_number; k++){

cur\_row = fir\_b[k].node\_numb;

double val = get\_func\_value\_fir(k);

di[cur\_row] = 1;

right\_part[cur\_row] = val;

int i\_s = gi[cur\_row], i\_e = gi[cur\_row+1];

for(int i = i\_s; i < i\_e; i++){

right\_part[gj[i]] -= gg[i]\*val;

gg[i] = 0;

}

for(int p = cur\_row + 1; p < nodes\_number; p++){

int i\_s = gi[p], i\_e = gi[p+1];

for(int i = i\_s; i < i\_e; i++){

if(gj[i] == cur\_row){

right\_part[p] -= gg[i]\*val;

gg[i] = 0;

}

}

}

}

}

void Grid::solve\_SLAE(){

CGM solver; //решатель СЛАУ

//Перебрасываем значения с предыдущего слоя на новый

double \*c = weight\_0;

weight\_0 = weight\_2;

weight\_2 = weight\_1;

weight\_1 = c;

t2 = t1;

t1 = t0;

t0 = t\_new;

//Иниицализируем СЛАУ и решаем

solver.init(gi, gj, di, gg, right\_part, nodes\_number);

solver.solve(weight\_0);

}

void Grid::gen\_local\_el(double \*\*A\_loc, double \*b\_loc, int el\_n){

//Формирование L-координат

double mes;

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

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

if(i==j)

alpha[i][j] = 1;

else

alpha[i][j] = 0;

A\_loc[i][j] = 0;

}

b\_loc[i] = 0;

}

//Матрица D

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

D[0][i] = 1;

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

D[1][i] = nodes[elements[el\_n].node\_n[i]].x;

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

D[2][i] = nodes[elements[el\_n].node\_n[i]].y;

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

D[3][i] = nodes[elements[el\_n].node\_n[i]].z;

mes = (D[1][1]-D[1][0])\*((D[2][2]-D[2][0])\*(D[3][3]-D[3][0]) - (D[2][3]-D[2][0])\*(D[3][2]-D[3][0]));

mes += (D[2][1]-D[2][0])\*((D[3][2]-D[3][0])\*(D[1][3]-D[1][0]) - (D[3][3]-D[3][0])\*(D[1][2]-D[1][0]));

mes += (D[3][1]-D[3][0])\*((D[1][2]-D[1][0])\*(D[2][3]-D[2][0]) - (D[1][3]-D[1][0])\*(D[2][2]-D[2][0]));

mes = fabs(mes)/6.0;

//Сами координаты

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

solve\_Gauss(D, alpha[i], 4);

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

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

D[i][j] = alpha[j][i];

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

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

alpha[i][j] = D[i][j];

//Формирование локальной матрицы

//Матрица жескости

double G[4][4];

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

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

double sum = 0;

for(int k = 1; k < 4; k++)

sum += alpha[i][k]\*alpha[j][k];

G[i][j] = sum\*mes;

}

}

//Матрица массы

double M[4][4];

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

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

if(i==j) M[i][j] = 2;

else M[i][j] = 1;

M[i][j] \*= mes/20.0;

}

}

//Локальные значения коэффициентов

double loc\_lambda = get\_lambda\_value(el\_n);

double loc\_gamma = get\_gamma\_value(el\_n);

double loc\_sigma = get\_sigma\_value(el\_n, t\_new);

double loc\_epsilon = get\_epsilon\_value(el\_n, t\_new);

double tau[3] = {t\_new - t0, t0 - t1, t1 - t2}; //шаги по времени

double tetta[4]; //коэффициенты при аппроксимации первой производной

double miu[4]; //коэффициенты при аппроксимации второй поизводной

double coef\_denoms[4]; //знаменатели коэффциентов

coef\_denoms[3] = (tau[0] + tau[1] + tau[2]) \* (tau[1] + tau[2]) \* tau[2];

coef\_denoms[2] = tau[2] \* tau[1] \* (tau[0] + tau[1]);

coef\_denoms[1] = tau[0] \* tau[1] \* (tau[1] + tau[2]);

coef\_denoms[0] = tau[0] \* (tau[0] + tau[1]) \* (tau[0] + tau[1] + tau[2]);

tetta[3] = -tau[0]\*(tau[0] + tau[1])/coef\_denoms[3];

tetta[2] = tau[0]\*(tau[0] + tau[1] + tau[2]) / coef\_denoms[2];

tetta[1] = -(tau[0] + tau[1])\*(tau[0] + tau[1] + tau[2]) / coef\_denoms[1];

tetta[0] = 1.0/tau[0] + 1.0/(tau[0] + tau[1]) + 1.0/(tau[0] + tau[1] + tau[2]);

miu[3] = -2.0\*(2.0\*tau[0] + tau[1]) / coef\_denoms[3];

miu[2] = 2.0\*(2.0\*tau[0] + tau[1] + tau[2]) / coef\_denoms[2];

miu[1] = -2.0\*(2.0\*tau[0] + 2.0\*tau[1] + tau[2]) / coef\_denoms[1];

miu[0] = 2.0\*(3.0\*tau[0] + 2.0\*tau[1] + tau[2]) / coef\_denoms[0];

//Сама локальная матрица

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

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

A\_loc[i][j] += loc\_lambda\*G[i][j] + (loc\_gamma + loc\_sigma\*tetta[0] + loc\_epsilon\*miu[0])\*M[i][j];

}

}

//Формирование локального вектора

double f\_loc = 0;

//Часть от исходного уравнения

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

f\_loc += alpha[i][1]\*get\_rp\_value(elements[el\_n].node\_n[i]);

//Часть от дискретизации по времени

double loc\_q0[4], loc\_q1[4], loc\_q2[4];

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

loc\_q0[i] = loc\_q1[i] = loc\_q2[i] = 0;

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

loc\_q0[i] += M[i][j]\*weight\_0[elements[el\_n].node\_n[j]];

loc\_q1[i] += M[i][j]\*weight\_1[elements[el\_n].node\_n[j]];

loc\_q2[i] += M[i][j]\*weight\_2[elements[el\_n].node\_n[j]];

}

loc\_q0[i] \*= (loc\_sigma\*tetta[1] + loc\_epsilon\*miu[1]);

loc\_q1[i] \*= (loc\_sigma\*tetta[2] + loc\_epsilon\*miu[2]);

loc\_q2[i] \*= (loc\_sigma\*tetta[3] + loc\_epsilon\*miu[3]);

}

//Суммируем их

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

b\_loc[i] += f\_loc\*mes/4.0 - loc\_q0[i] - loc\_q1[i] - loc\_q2[i];

//Debug

double t\_rp[4], t\_x[4];

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

t\_x[i] = sol\_func(nodes[elements[el\_n].node\_n[i]].x, nodes[elements[el\_n].node\_n[i]].y, nodes[elements[el\_n].node\_n[i]].z, t\_new);

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

t\_rp[i] = 0;

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

t\_rp[i] += A\_loc[i][j]\*t\_x[j];

}

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

t\_x[i] = t\_rp[i] - b\_loc[i];

//Debug

}

void Grid::gen\_local\_thi(double \*\*A\_loc, double \*b\_loc, int f\_n){

double mes = get\_face\_mes(3, f\_n);

double betta = get\_betta\_value(f\_n);

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

for(int j = 0; j < 3; j++){

A\_loc[i][j] = betta\*mes/12;

}

A\_loc[i][i] += betta\*mes/12;

}

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

b\_loc[i] = 0;

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

b\_loc[i] += A\_loc[i][j]\*get\_u\_betta\_value(f\_n, faces\_t[f\_n].node\_n[j]);

}

}

void Grid::gen\_local\_sec(double \*b\_loc, int f\_n){

double mes = get\_face\_mes(2, f\_n);

double tetta[3];

double sum\_tetta = 0;

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

tetta[i] = get\_tetta\_value(f\_n, faces\_s[f\_n].node\_n[i]);

sum\_tetta += tetta[i];

}

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

b\_loc[i] = mes\*(tetta[i] + sum\_tetta)/12;

}

}

double Grid::get\_face\_mes(int boun, int f\_n){

double mes;

double p[3];

node loc\_node[3];

if(boun == 3)

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

loc\_node[i] = nodes[faces\_t[f\_n].node\_n[i]];

else

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

loc\_node[i] = nodes[faces\_s[f\_n].node\_n[i]];

p[0] = (loc\_node[1].y - loc\_node[0].y) \* (loc\_node[2].z - loc\_node[0].z) - (loc\_node[2].y - loc\_node[0].y) \* (loc\_node[1].z - loc\_node[0].z);

p[1] = (loc\_node[2].x - loc\_node[0].x) \* (loc\_node[1].z - loc\_node[0].z) - (loc\_node[1].x - loc\_node[0].x) \* (loc\_node[2].z - loc\_node[0].z);

p[2] = (loc\_node[1].x - loc\_node[0].x) \* (loc\_node[2].y - loc\_node[0].y) - (loc\_node[2].x - loc\_node[0].x) \* (loc\_node[1].y - loc\_node[0].y);

mes = 0;

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

mes += p[i]\*p[i];

mes = 0.5\*sqrt(mes);

return mes;

}

int Grid::find\_el\_pos(int i, int j){

int k\_s = gi[i], k\_e = gi[i+1];

int cur;

bool find = false;

for(int k = k\_s; k < k\_e && !find; k++){

if(gj[k] == j){

cur = k;

find = true;

}

}

return cur;

}

double Grid::get\_lambda\_value(int el\_n){

return lambda\_values[elements[el\_n].area];

}

double Grid::get\_gamma\_value(int el\_n){

double gamma = 0;

int area = elements[el\_n].area;

node loc\_nodes[4];

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

loc\_nodes[i] = nodes[elements[el\_n].node\_n[i]];

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

gamma += gamma\_values[area](loc\_nodes[i].x, loc\_nodes[i].y, loc\_nodes[i].z, t\_new);

gamma /= 4;

return gamma;

}

double Grid::get\_sigma\_value(int el\_n, double t){

double sigma = 0;

int area = elements[el\_n].area;

node loc\_nodes[4];

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

loc\_nodes[i] = nodes[elements[el\_n].node\_n[i]];

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

sigma += sigma\_values[area](loc\_nodes[i].x, loc\_nodes[i].y, loc\_nodes[i].z, t\_new);

sigma /= 4;

return sigma;

}

double Grid::get\_epsilon\_value(int el\_n, double t){

double epsilon = 0;

int area = elements[el\_n].area;

node loc\_nodes[4];

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

loc\_nodes[i] = nodes[elements[el\_n].node\_n[i]];

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

epsilon += epsilon\_values[area](loc\_nodes[i].x, loc\_nodes[i].y, loc\_nodes[i].z, t\_new);

epsilon /= 4;

return epsilon;

}

double Grid::get\_betta\_value(int f\_n){

return betta\_values[faces\_t[f\_n].area];

}

double Grid::get\_u\_betta\_value(int f\_n, int node\_n){

return u\_betta\_values[faces\_t[f\_n].area](nodes[node\_n].x, nodes[node\_n].y, nodes[node\_n].z, t\_new);

}

double Grid::get\_tetta\_value(int f\_n, int node\_n){

return tetta\_values[faces\_s[f\_n].area](nodes[node\_n].x, nodes[node\_n].y, nodes[node\_n].z, t\_new);

}

double Grid::get\_rp\_value(int node\_n){

return equation\_rp(nodes[node\_n].x, nodes[node\_n].y, nodes[node\_n].z, t\_new);

}

double Grid::get\_func\_value\_fir(int node\_n){

int gl\_node = fir\_b[node\_n].node\_numb;

return fir\_values[fir\_b[node\_n].area](nodes[gl\_node].x, nodes[gl\_node].y, nodes[gl\_node].z, t\_new);

}

void Grid::output\_weight(FILE \*outp\_f){

fprintf(outp\_f, "Time =\t%.10lf\n", t0);

for(int i = 0; i < nodes\_number; i++)

fprintf(outp\_f,"%d\t%.15lf\n",i , weight\_0[i]);

}

void Grid::output\_diff(FILE\* outp\_f){

double diff = 0;

double denom = 0;

for(int i = 0; i < nodes\_number; i++){

double u\_val = sol\_func(nodes[i].x, nodes[i].y, nodes[i].z, t0);

diff += (weight\_0[i] - u\_val) \* (weight\_0[i] - u\_val);

denom += u\_val \* u\_val;

}

diff /= denom;

fprintf(outp\_f, "%.5lf\t%.3e\n", t0, diff);

}

double Grid::u\_in\_el(double x, double y, double z){

for(int k = 0; k < elements\_number; k++){

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

D[0][i] = 1;

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

D[1][i] = nodes[elements[k].node\_n[i]].x;

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

D[2][i] = nodes[elements[k].node\_n[i]].y;

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

D[3][i] = nodes[elements[k].node\_n[i]].z;

double V[4] = {0, 0, 0, 0}, mes = 0;

V[0] = (D[1][1]-x)\*((D[2][2]-y)\*(D[3][3]-z) - (D[2][3]-y)\*(D[3][2]-z));

V[0] += (D[2][1]-y)\*((D[3][2]-z)\*(D[1][3]-x) - (D[3][3]-z)\*(D[1][2]-x));

V[0] += (D[3][1]-z)\*((D[1][2]-x)\*(D[2][3]-y) - (D[1][3]-x)\*(D[2][2]-y));

V[1] = (x-D[1][0])\*((D[2][2]-D[2][0])\*(D[3][3]-D[3][0]) - (D[2][3]-D[2][0])\*(D[3][2]-D[3][0]));

V[1] += (y-D[2][0])\*((D[3][2]-D[3][0])\*(D[1][3]-D[1][0]) - (D[3][3]-D[3][0])\*(D[1][2]-D[1][0]));

V[1] += (z-D[3][0])\*((D[1][2]-D[1][0])\*(D[2][3]-D[2][0]) - (D[1][3]-D[1][0])\*(D[2][2]-D[2][0]));

V[2] = (D[1][1]-D[1][0])\*((y-D[2][0])\*(D[3][3]-D[3][0]) - (D[2][3]-D[2][0])\*(z-D[3][0]));

V[2] += (D[2][1]-D[2][0])\*((z-D[3][0])\*(D[1][3]-D[1][0]) - (D[3][3]-D[3][0])\*(x-D[1][0]));

V[2] += (D[3][1]-D[3][0])\*((x-D[1][0])\*(D[2][3]-D[2][0]) - (D[1][3]-D[1][0])\*(y-D[2][0]));

V[3] = (D[1][1]-D[1][0])\*((D[2][2]-D[2][0])\*(z-D[3][0]) - (y-D[2][0])\*(D[3][2]-D[3][0]));

V[3] += (D[2][1]-D[2][0])\*((D[3][2]-D[3][0])\*(x-D[1][0]) - (z-D[3][0])\*(D[1][2]-D[1][0]));

V[3] += (D[3][1]-D[3][0])\*((D[1][2]-D[1][0])\*(y-D[2][0]) - (x-D[1][0])\*(D[2][2]-D[2][0]));

mes = (D[1][1]-D[1][0])\*((D[2][2]-D[2][0])\*(D[3][3]-D[3][0]) - (D[2][3]-D[2][0])\*(D[3][2]-D[3][0]));

mes += (D[2][1]-D[2][0])\*((D[3][2]-D[3][0])\*(D[1][3]-D[1][0]) - (D[3][3]-D[3][0])\*(D[1][2]-D[1][0]));

mes += (D[3][1]-D[3][0])\*((D[1][2]-D[1][0])\*(D[2][3]-D[2][0]) - (D[1][3]-D[1][0])\*(D[2][2]-D[2][0]));

if(V[0] + V[1] + V[2] + V[3] <= mes){

double val = 0;

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

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

if(i==j)

alpha[i][j] = 1;

else

alpha[i][j] = 0;

}

}

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

solve\_Gauss(D, alpha[i], 4);

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

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

D[i][j] = alpha[j][i];

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

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

alpha[i][j] = D[i][j];

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

val += weight\_0[elements[k].node\_n[i]]\*(alpha[i][0] + x\*alpha[i][1] + y\*alpha[i][2] + z\*alpha[i][3]);

return val;

}

}

return 0;

}

## Файл «additional\_classes.cpp»

#include "additional\_classes.h"

SLAE\_port\_list::SLAE\_port\_list(){

init();

}

void SLAE\_port\_list::init(){

begin = end = cash = 0;

l\_size = 0;

}

void SLAE\_port\_list::set\_num(int s\_num){

number\_of\_line = s\_num;

}

SLAE\_port\_list::~SLAE\_port\_list(){

cash = 0;

while(begin != 0)

exclude\_last\_el();

}

void SLAE\_port\_list::add\_el(FE &el\_a){

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

add(el\_a.node\_n[i]);

}

void SLAE\_port\_list::cash\_off(){

cash = begin;

}

int SLAE\_port\_list::take\_and\_next(){

int cash\_v = cash->value;

cash = cash->next;

return cash\_v;

}

int SLAE\_port\_list::get\_m\_size(){

return m\_size;

}

void SLAE\_port\_list::add(int val){

if(val <= number\_of\_line){

gen\_l\_el \*add\_el;

if(begin == 0){

begin = new gen\_l\_el;

begin->value = val;

begin->next = 0;

end = begin;

cash = begin;

}

else{

if(val < begin->value){

add\_el = new gen\_l\_el;

add\_el->value = val;

add\_el->next = begin;

begin = add\_el;

cash = begin;

}

else{

if(val > end->value){

add\_el = new gen\_l\_el;

add\_el->value = val;

add\_el->next = 0;

end->next = add\_el;

end = end->next;

}

else{

cash = begin;

while(cash->next != 0 && val > cash->next->value) cash = cash->next;

if(cash->next != 0 && cash->next->value != val && cash->value != val){

add\_el = new gen\_l\_el;

add\_el->value = val;

add\_el->next = cash->next;

cash->next = add\_el;

}

}

}

}

l\_size++;

}

}

void SLAE\_port\_list::exclude\_last\_el(){

if(begin != end){

cash = begin;

while(cash->next != end) cash = cash->next;

delete cash->next;

cash->next = 0;

end = cash;

}

else{

delete begin;

begin = end = cash = 0;

}

}

int SLAE\_port\_list::size\_before(int n){

int tmp\_s = 0;

cash = begin;

if(begin != 0){

if(begin->value < n) tmp\_s++;

while(cash->next != 0 && cash->next->value < n){

tmp\_s++;

cash = cash->next;

}

}

m\_size = tmp\_s;

return tmp\_s;

}

SLAE\_port\_gen::SLAE\_port\_gen(){

}

SLAE\_port\_gen::~SLAE\_port\_gen(){

delete []lists;

lists = 0;

}

void SLAE\_port\_gen::init(int nodes\_n){

n = nodes\_n;

lists = new SLAE\_port\_list [n];

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

lists[i].set\_num(i);

}

void SLAE\_port\_gen::add\_el(FE &el\_a){

lists[el\_a.node\_n[0]].add\_el(el\_a);

lists[el\_a.node\_n[1]].add\_el(el\_a);

lists[el\_a.node\_n[2]].add\_el(el\_a);

lists[el\_a.node\_n[3]].add\_el(el\_a);

}

void SLAE\_port\_gen::gen(int \*gi, int \*&gj, int &m){

m = 0;

gi[0] = 0;

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

gi[i] = m;

m += lists[i].size\_before(i);

}

gi[n] = m;

gj = new int [m];

int shift, m1 = 0, iters\_m;

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

shift = m1;

iters\_m = lists[i].get\_m\_size();

lists[i].cash\_off();

for(int j = 0; j < iters\_m; j++)

gj[shift+j] = lists[i].take\_and\_next();

m1 += iters\_m;

}

}

## Файл «areas.cpp»

#include "areas.h"

namespace area\_1{

string path = "time\_test\_1/";

int lambda\_n = 1;

int betta\_n = 0;

int u\_betta\_n = 0;

int tetta\_n = 1;

int fir\_n = 1;

double \*lambda\_values;

double \*betta\_values;

func \*gamma\_values;

func \*sigma\_values;

func \*epsilon\_values;

func \*u\_betta\_values;

func \*tetta\_values;

func \*fir\_values;

double gamma(double x, double y, double z, double t){

return 0;

}

double sigma(double x, double y, double z, double t){

return 0;

}

double epsilon(double x, double y, double z, double t){

return 0;

}

double firval0(double x, double y, double z, double t){

return y\*t;

}

double tetta0(double x, double y, double z, double t){

return 0.0;

}

void init\_area(){

lambda\_values = new double [lambda\_n];

gamma\_values = new func [lambda\_n];

sigma\_values = new func [lambda\_n];

epsilon\_values = new func [lambda\_n];

betta\_values = new double [betta\_n];

u\_betta\_values = new func [u\_betta\_n];

tetta\_values = new func [tetta\_n];

fir\_values = new func [fir\_n];

lambda\_values[0] = 1.0;

gamma\_values[0] = gamma;

sigma\_values[0] = sigma;

epsilon\_values[0] = epsilon;

tetta\_values[0] = tetta0;

fir\_values[0] = firval0;

}

double sol\_func(double x, double y, double z, double t){

return y\*t;

}

double equation\_rp(double x, double y, double z, double t){

return 0;

}

}

namespace area\_2{

string path = "time\_test\_2/";

int lambda\_n = 1;

int betta\_n = 0;

int u\_betta\_n = 0;

int tetta\_n = 0;

int fir\_n = 1;

double \*lambda\_values;

double \*betta\_values;

func \*gamma\_values;

func \*sigma\_values;

func \*epsilon\_values;

func \*u\_betta\_values;

func \*tetta\_values;

func \*fir\_values;

double gamma(double x, double y, double z, double t){

return 0.0;

}

double sigma(double x, double y, double z, double t){

return 10;

}

double epsilon(double x, double y, double z, double t){

return 0.1;

}

double firval0(double x, double y, double z, double t){

return exp(4\*t)+y;

}

void init\_area(){

lambda\_values = new double [lambda\_n];

gamma\_values = new func [lambda\_n];

sigma\_values = new func [lambda\_n];

epsilon\_values = new func [lambda\_n];

betta\_values = new double [betta\_n];

u\_betta\_values = new func [u\_betta\_n];

tetta\_values = new func [tetta\_n];

fir\_values = new func [fir\_n];

lambda\_values[0] = 2.0;

gamma\_values[0] = gamma;

sigma\_values[0] = sigma;

epsilon\_values[0] = epsilon;

fir\_values[0] = firval0;

}

double sol\_func(double x, double y, double z, double t){

return exp(4\*t)+y;

}

double equation\_rp(double x, double y, double z, double t){

return 41.6\*x\*exp(4\*t);

}

}

namespace area\_3{

string path = "time\_test\_3/";

int lambda\_n = 1;

int betta\_n = 0;

int u\_betta\_n = 0;

int tetta\_n = 0;

int fir\_n = 1;

double \*lambda\_values;

double \*betta\_values;

func \*gamma\_values;

func \*sigma\_values;

func \*epsilon\_values;

func \*u\_betta\_values;

func \*tetta\_values;

func \*fir\_values;

double p = 10.0;

double G = 2.0;

double gamma(double x, double y, double z, double t){

return 0.0;

}

double sigma(double x, double y, double z, double t){

return 0.0;

}

double epsilon(double x, double y, double z, double t){

return p/G;

}

double firval0(double x, double y, double z, double t){

return 0;

}

void init\_area(){

lambda\_values = new double [lambda\_n];

gamma\_values = new func [lambda\_n];

sigma\_values = new func [lambda\_n];

epsilon\_values = new func [lambda\_n];

betta\_values = new double [betta\_n];

u\_betta\_values = new func [u\_betta\_n];

tetta\_values = new func [tetta\_n];

fir\_values = new func [fir\_n];

lambda\_values[0] = 1.0;

gamma\_values[0] = gamma;

sigma\_values[0] = sigma;

epsilon\_values[0] = epsilon;

fir\_values[0] = firval0;

}

double sol\_func(double x, double y, double z, double t){

/\*double x0 = 0.600006000020000, y0 = 0.60000600002000, z0 = 0.600006000020000;

double r = sqrt((x-x0)\*(x-x0) + (y-y0)\*(y-y0) + (z-z0)\*(z-z0));

double pi = 4\*atan(1.0);

double v = sqrt(G/p);

return sin(t - r/v)/(4.0\*pi\*r);\*/

return 0.0;

}

double equation\_rp(double x, double y, double z, double t){

return 0;

}

}

## Файл «CGM.cpp»

#include "CGM.h"

void CGM::init(int \*gi\_s, int \*gj\_s, double \*di\_s, double \*gg\_s, double \*rp\_s, int n\_s){

gi = gi\_s;

gj = gj\_s;

di = di\_s;

gg = gg\_s;

rp = rp\_s;

n = n\_s;

int m = gi[n];

r = new double [n];

x0 = new double [n];

z = new double [n];

p = new double [n];

s = new double [n];

L\_di = new double [n];

L\_gg = new double [m];

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

L\_di[i] = di[i];

x0[i] = 0;

}

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

L\_gg[i] = gg[i];

}

void CGM::make\_LLT\_decomposition(){

double sum\_d, sum\_l;

for(int k = 0; k < n ; k++){

sum\_d = 0;

int i\_s = gi[k], i\_e = gi[k+1];

for(int i = i\_s; i < i\_e ; i++){

sum\_l = 0;

int j\_s = gi[gj[i]], j\_e = gi[gj[i]+1];

for(int m = i\_s; m < i; m++){

for(int j = j\_s; j < j\_e; j++){

if(gj[m] == gj[j])

{

sum\_l += L\_gg[m]\*L\_gg[j];

j\_s++;

}

}

}

L\_gg[i] = (L\_gg[i] - sum\_l)/L\_di[gj[i]];

sum\_d += L\_gg[i]\*L\_gg[i];

}

L\_di[k] = sqrt(L\_di[k] - sum\_d);

}

}

double CGM::dot\_prod(double \*a, double \*b){

double d\_p = 0;

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

d\_p += a[i]\*b[i];

return d\_p;

}

void CGM::mul\_matrix(double \*f, double \*&x){

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

double v\_el = f[i];

x[i] = di[i]\*v\_el;

for(int k = gi[i], k1 = gi[i+1]; k < k1; k++){

int j = gj[k];

x[i] += gg[k]\*f[j];

x[j] += gg[k]\*v\_el;

}

}

}

void CGM::solve\_L(double \*f, double \*&x){

for(int k = 1, k1 = 0; k <= n; k++, k1++){

double sum = 0;

for(int i = gi[k1]; i < gi[k]; i++)

sum += L\_gg[i]\*x[gj[i]];

x[k1] = (f[k1] - sum)/L\_di[k1];

}

}

void CGM::solve\_LT(double \*f, double \*&x){

for(int k = n, k1 = n-1; k > 0; k--, k1--){

x[k1] = f[k1]/L\_di[k1];

double v\_el = x[k1];

for(int i = gi[k1]; i < gi[k]; i++)

f[gj[i]] -= L\_gg[i]\*v\_el;

}

}

void CGM::solve\_LLT(double \*f, double \*&x){

solve\_L(f, x);

solve\_LT(x,x);

}

void CGM::solve(double \*&solution){

// Параметры решателя

int max\_iter = 1000;

double eps = 1E-20;

mul\_matrix(x0, r);

make\_LLT\_decomposition();

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

r[i] = rp[i] - r[i];

solve\_LLT(r, z);

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

p[i] = z[i];

double alpha, betta, prod\_1, prod\_2;

double discr, rp\_norm;

rp\_norm = sqrt(dot\_prod(rp,rp));

prod\_1 = dot\_prod(p, r);

bool end = false;

for(int iter = 0; iter < max\_iter && !end; iter++){

discr = sqrt(dot\_prod(r,r));

if(eps < discr/rp\_norm){

mul\_matrix(z, s);

alpha = prod\_1 / dot\_prod(s, z);

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

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

r[i] -= alpha \* s[i];

}

solve\_LLT(r, p);

prod\_2 = dot\_prod(p, r);

betta = prod\_2 / prod\_1;

prod\_1 = prod\_2;

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

z[i] = p[i] + betta\*z[i];

}

else

end = true;

}

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

solution[i] = x0[i];

}

## Файл «Gauss.cpp»

#include "Gauss.h"

double \*\*B;

void prepared(int n){

B = new double\* [n];

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

B[i] = new double [n];

}

void solve\_Gauss(double \*\*A, double \*x, int n){

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

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

B[i][j] = A[i][j];

trianglematrix1(B,x,n);

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

double sum = 0;

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

sum += B[i][j]\*x[j];

x[i] -= sum;

}

}

void trianglematrix1(double \*\*A, double \*x, int n){

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

transf1(A,x,i,n);

double A\_d = A[i][i];

for(int p = i; p < n; p++){

A[i][p] /= A\_d;

}

x[i] /= A\_d;

for(int j = i + 1; j < n; j++){

double A\_j = A[j][i];

if(A\_j != 0){

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

A[j][k] -= A[i][k]\*A\_j;

x[j] -= x[i]\*A\_j;

}

}

}

}

void transf1(double \*\*A, double \*x, int i, int n){

int line = i;

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

if(fabs(A[j][i]) > fabs(A[line][i]))

line = j;

if (line != i){

double c;

for(int j = i; j < n; j++){

c = A[i][j];

A[i][j] = A[line][j];

A[line][j] = c;

}

c = x[i];

x[i] = x[line];

x[line] = c;

}

}

## Файл «launch.cpp»

#include "base\_classes.h"

#include "grid\_gen\_tet.h"

#include <sstream>

using namespace std;

void gen\_grid(); //Генерируем сетку итд

void gener\_starts(); //генерация начальных условий

int main(){

//gen\_grid();

gener\_starts();

Grid g1;

string f\_m;

FILE\* f;

f\_m = path + "el.txt";

f = fopen(f\_m.c\_str(), "r");

g1.input\_elements(f);

fclose(f);

f\_m = path + "nodes.txt";

f = fopen(f\_m.c\_str(), "r");

g1.input\_nodes(f);

fclose(f);

f\_m = path + "faces\_s.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_faces\_s(f);

fclose(f);

f\_m = path + "faces\_s\_ar.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_f\_s\_areas(f);

fclose(f);

f\_m = path + "faces\_t.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_faces\_t(f);

fclose(f);

f\_m = path + "faces\_t\_ar.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_f\_t\_areas(f);

fclose(f);

f\_m = path + "first\_bound.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_fir\_nodes(f);

fclose(f);

f\_m = path + "el\_areas.txt";

f = fopen(f\_m.c\_str(),"r");

g1.input\_el\_areas(f);

fclose(f);

//Инифиализация всего, что связанно со временем

f\_m = path + "time data.txt";

f = fopen(f\_m.c\_str(), "r");

g1.input\_time\_data(f);

fclose(f);

FILE \*f1, \*f2;

f\_m = path + "time0.txt";

f = fopen(f\_m.c\_str(), "r");

f\_m = path + "time1.txt";

f1 = fopen(f\_m.c\_str(), "r");

f\_m = path + "time2.txt";

f2 = fopen(f\_m.c\_str(), "r");

g1.input\_start(f, f1, f2);

fclose(f);

fclose(f1);

fclose(f2);

g1.gen\_SLAE\_port();

f\_m = path + "diffs.txt";

f1 = fopen(f\_m.c\_str(), "w");

f\_m = path + "result.txt";

f = fopen(f\_m.c\_str(),"w");

for(int i = 0; i < 10; i++){

stringstream a;

a << i;

string tmp\_str;

a >> tmp\_str;

f\_m = path + "result" + tmp\_str + ".txt";

f = fopen(f\_m.c\_str(),"w");

g1.gen\_SLAE();

g1.solve\_SLAE();

g1.output\_weight(f);

g1.output\_diff(f1);

fprintf(f, "\n\n");

}

fclose(f);

fclose(f1);

/\*

double po[12][3] = {{0,0.0625,0.125}, {0.0625,0,0.125}, {0.0625, 0.0625, 0.125}, {0, 0, 0.1875}, {0.0625, 0 ,0.1875}, {0, 0.0625, 0.1875}, {0.1875, 0, 0}, {0.1875, 0 ,0.0625}, {0.125, 0 , 0.0625}, {0.1875, 0.0625, 0}, {0.125, 0.0625, 0.0625}, {0.125, 0.0625, 0} };

f\_m = path + "res\_inpoint.txt";

f = fopen(f\_m.c\_str(),"w");

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

fprintf(f,"%.15lf\n",g1.u\_in\_el(po[i][0],po[i][1],po[i][2]));

fclose(f);

\*/

return 0;

}

void gen\_grid(){

//Сначало строим кубическую сетку

double hx = 0.2;

double hy = 0.2;

double hz = 0.2;

double kx = 1.00001;

double ky = 1.00001;

double kz = 1.00001;

grid\_gen\_cube::generate\_unreg\_grid\_FEM(gen\_point(0, 0, 0), gen\_point(1.0, 1.0, 1.0), hz, hy, hz, kx, ky, kz, "nodes.txt", "cube\_elements.txt", "cube\_faces.txt");

//Далее проебразуем её в тетраэдальную сетку

grid\_gen\_tet::transform\_cube\_to\_tetrahedron("cube\_elements.txt", "cube\_faces.txt", "el.txt", "faces.txt");

grid\_gen\_tet::generate\_face\_nodes("faces.txt", "faces\_nodes.txt");

//grid\_gen\_tet::ins\_grid("to gen/el.txt", "to gen/face1.txt", "to gen/nodes.txt", "el.txt", "1face.txt", "nodes.txt");

//grid\_gen\_tet::generate\_face\_nodes("1face.txt", "faces\_nodes.txt");

FILE\* inp\_f = fopen("faces\_nodes.txt", "r");

FILE\* out\_f = fopen("first\_bound.txt", "w");

int fir\_b\_n;

fscanf(inp\_f, "%d", &fir\_b\_n);

fprintf(out\_f, "%d\n", fir\_b\_n);

int tmp\_int;

for(int i = 0; i < fir\_b\_n; i++){

fscanf(inp\_f,"%d", &tmp\_int);

fprintf(out\_f, "%d %d\n", tmp\_int, 0);

}

fclose(inp\_f);

fclose(out\_f);

int els\_n;

inp\_f = fopen("el.txt", "r");

fscanf(inp\_f, "%d", &els\_n);

fclose(inp\_f);

out\_f = fopen("el\_areas.txt", "w");

for(int i = 0; i < els\_n; i++)

fprintf(out\_f, "%d %d\n", i, 0);

fclose(out\_f);

}

void gener\_starts(){

string f\_m;

int node\_n;

double x, y, z;

//Считываемм данные о времени

f\_m = path + "time data.txt";

FILE\* fd = fopen(f\_m.c\_str(), "r");

double t0, tau0, k;

fscanf(fd, "%lf %lf %lf",&t0, &tau0, &k);

fclose(fd);

double t1, t2;

t1 = t0 + tau0;

t2 = t1 + k\*tau0;

//Открывае файлы для вывода

FILE \*f0, \*f1, \*f2;

f\_m = path + "time0.txt";

f0 = fopen(f\_m.c\_str(), "w");

f\_m = path + "time1.txt";

f1 = fopen(f\_m.c\_str(), "w");

f\_m = path + "time2.txt";

f2 = fopen(f\_m.c\_str(), "w");

//Считываем данные об узлах

f\_m = path + "nodes.txt";

FILE\* fn = fopen(f\_m.c\_str(), "r");

fscanf(fn, "%d", &node\_n);

//Генерируем

for(int i = 0; i < node\_n; i++){

fscanf(fn, "%lf %lf %lf", &x, &y, &z);

fprintf(f0, "%.15lf\n", sol\_func(x,y,z,t2));

fprintf(f1, "%.15lf\n", sol\_func(x,y,z,t1));

fprintf(f2, "%.15lf\n", sol\_func(x,y,z,t0));

}

fclose(fn);

fclose(f0);

fclose(f1);

fclose(f2);

}