Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ПМт

УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Практическое задание № 3

# Решение гармонических задач

Факультет: ПМИ Преподаватели:

Задорожный А. Г.,

Патрушев И.И.

Группа: ПМ-81

Студенты: Ефремов А.А.,

Ртищева К. С.

Бригада: 1

Вариант: 9

Новосибирск

2021

1. **Цель работы**

Разработать программу решения гармонической задачи методом конечных элементов. Провести сравнение прямого и итерационного методов решения получаемой в результате конечноэлементной аппроксимации СЛАУ

1. **Задание**

Решить трехмерную гармоническую задачу в декартовых координатах, базисные функции – трилинейные.

1. **Анализ задачи**

Рассмотрим задачу для уравнения:

Функции удовлетворяют системе

И должны удовлетворять краевым условиям .

Умножим каждое уравнение системы на пробную функцию и применим формулу Грина, получим систему двух вариационных уравнений:

Построим конечноэлементную аппроксимацию:

Заменим каждую из искомых функций на функции

– базисные функции

В результате получаем систему из 2n уравнений с 2n неизвестными и . Чтобы определить матрицу и вектор правой части, полученной конечноэлементной СЛАУ, пронумеруем уравнения и неизвестные этой системы поочередно.

Обозначим

где

Тогда матрица конечноэлементной СЛАУ будет состоять из блоков вида:

Для трехмерной задачи в декартовых координатах получаем

где – трилинейные базисные функции.

Тогда, учитывая представление базисных функций:

– локальные матрицы жесткости, – локальные матрицы массы соответствующих одномерных линейных элементов

Получаем

**Учет первых краевых условий:**

в матрице СЛАУ в строке на место диагонального элемента ставится единица, все остальные элементы этой строки матрицы обнуляются, а компоненте вектора правой части присваивается значение .

1. **Проверка работоспособности программы**

|  |  |  |
| --- | --- | --- |
| 0  2  2  2 | ***Файл boundaries.txt***  6  1 0 1 0 1 0 0  1 0 1 0 1 1 1  1 0 0 0 1 0 1  1 1 1 0 1 0 1  1 0 1 0 0 0 1  1 0 1 1 1 0 1 | ***Файл grid.txt***  ----X----  2  0 2  2  ----Y----  2  0 2  2  ----Z----  2  0 2  2  -Regions-  1  0 1 0 1 0 1 |

**x y z prec calc diff n loc**

0.000000e+00 0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 0 border

0.000000e+00 0.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 1 border

1.000000e+00 0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 2 border

1.000000e+00 0.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 3 border

2.000000e+00 0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 4 border

2.000000e+00 0.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 5 border

0.000000e+00 1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 6 border

0.000000e+00 1.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 7 border

1.000000e+00 1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 8 border

1.000000e+00 1.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 9 border

2.000000e+00 1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 10 border

2.000000e+00 1.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 11 border

0.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 12 border

0.000000e+00 2.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 13 border

1.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 14 border

1.000000e+00 2.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 15 border

2.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 16 border

2.000000e+00 2.000000e+00 0.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 17 border

0.000000e+00 0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 18 border

0.000000e+00 0.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 19 border

1.000000e+00 0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 20 border

1.000000e+00 0.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 21 border

2.000000e+00 0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 22 border

2.000000e+00 0.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 23 border

0.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 24 border

0.000000e+00 1.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 25 border

1.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 26 inner

1.000000e+00 1.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 27 inner

2.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 28 border

2.000000e+00 1.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 29 border

0.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 30 border

0.000000e+00 2.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 31 border

1.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 32 border

1.000000e+00 2.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 33 border

2.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 34 border

2.000000e+00 2.000000e+00 1.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 35 border

0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 36 border

0.000000e+00 0.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 37 border

1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 38 border

1.000000e+00 0.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 39 border

2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 40 border

2.000000e+00 0.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 41 border

0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 42 border

0.000000e+00 1.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 43 border

1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 44 border

1.000000e+00 1.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 0.000000e+00 45 border

2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 46 border

2.000000e+00 1.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 47 border

0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 48 border

0.000000e+00 2.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 49 border

1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 50 border

1.000000e+00 2.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 1.776357e-15 51 border

2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-15 52 border

2.000000e+00 2.000000e+00 2.000000e+00 5.000000e+00 5.000000e+00 8.881784e-16 53 border

||u-u\*||/||u\*|| = 2.648036e-16

||u-u\*|| = 6.879800e-15

**x y z prec calc diff n loc**

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.078267e-15 1.078267e-15 0 border

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -1.801862e-16 1.801862e-16 1 border

1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 -5.293063e-16 5.293063e-16 2 border

1.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.443290e-15 3 border

2.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.078267e-15 1.078267e-15 4 border

2.000000e+00 0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 5 border

0.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.110223e-15 6 border

0.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.443290e-15 7 border

1.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 9.992007e-16 8 border

1.000000e+00 1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 9 border

2.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.110223e-15 10 border

2.000000e+00 1.000000e+00 0.000000e+00 3.000000e+00 3.000000e+00 8.881784e-16 11 border

0.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 12 border

0.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 13 border

1.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 14 border

1.000000e+00 2.000000e+00 0.000000e+00 3.000000e+00 3.000000e+00 8.881784e-16 15 border

2.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 16 border

2.000000e+00 2.000000e+00 0.000000e+00 4.000000e+00 4.000000e+00 0.000000e+00 17 border

0.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.110223e-15 18 border

0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 -4.434891e-16 4.434891e-16 19 border

1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 9.992007e-16 20 border

1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 21 border

2.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.110223e-15 22 border

2.000000e+00 0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 23 border

0.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 8.881784e-16 24 border

0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 25 border

1.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 26 inner

1.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.554312e-15 27 inner

2.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 8.881784e-16 28 border

2.000000e+00 1.000000e+00 1.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 29 border

0.000000e+00 2.000000e+00 1.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 30 border

0.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 31 border

1.000000e+00 2.000000e+00 1.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 32 border

1.000000e+00 2.000000e+00 1.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 33 border

2.000000e+00 2.000000e+00 1.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 34 border

2.000000e+00 2.000000e+00 1.000000e+00 4.000000e+00 4.000000e+00 2.664535e-15 35 border

0.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 36 border

0.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 -1.801862e-16 1.801862e-16 37 border

1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 38 border

1.000000e+00 0.000000e+00 2.000000e+00 1.000000e+00 1.000000e+00 1.443290e-15 39 border

2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.332268e-15 40 border

2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 41 border

0.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 42 border

0.000000e+00 1.000000e+00 2.000000e+00 1.000000e+00 1.000000e+00 1.443290e-15 43 border

1.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 44 border

1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 45 border

2.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 4.440892e-16 46 border

2.000000e+00 1.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 8.881784e-16 47 border

0.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 1.776357e-15 48 border

0.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 49 border

1.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 0.000000e+00 50 border

1.000000e+00 2.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 8.881784e-16 51 border

2.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 1.776357e-15 52 border

2.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 0.000000e+00 53 border

||u-u\*||/||u\*|| = 4.289137e-16

||u-u\*|| = 5.146964e-15

**x y z prec calc diff n loc**

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 0 border

0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 1 border

1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 2 border

1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 3 border

2.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 4 border

2.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 5 border

0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 6 border

0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 7 border

1.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 -2.341408e-16 2.341408e-16 8 border

1.000000e+00 1.000000e+00 0.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 9 border

2.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 10 border

2.000000e+00 1.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 11 border

0.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 12 border

0.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 13 border

1.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 14 border

1.000000e+00 2.000000e+00 0.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 15 border

2.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 16 border

2.000000e+00 2.000000e+00 0.000000e+00 4.000000e+00 4.000000e+00 8.881784e-16 17 border

0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 18 border

0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 19 border

1.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 -2.341408e-16 2.341408e-16 20 border

1.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 -7.313969e-17 7.313969e-17 21 border

2.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 22 border

2.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 23 border

0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.110223e-16 24 border

0.000000e+00 1.000000e+00 1.000000e+00 0.000000e+00 -7.313969e-17 7.313969e-17 25 border

1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.110223e-15 26 inner

1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 4.440892e-16 27 inner

2.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.110223e-16 28 border

2.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 29 border

0.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 30 border

0.000000e+00 2.000000e+00 1.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 31 border

1.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 32 border

1.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 4.440892e-16 33 border

2.000000e+00 2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 34 border

2.000000e+00 2.000000e+00 1.000000e+00 4.000000e+00 4.000000e+00 8.881784e-16 35 border

0.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 36 border

0.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 37 border

1.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 1.237749e-15 1.237749e-15 38 border

1.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 39 border

2.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 7.000781e-16 7.000781e-16 40 border

2.000000e+00 0.000000e+00 2.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 41 border

0.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 42 border

0.000000e+00 1.000000e+00 2.000000e+00 0.000000e+00 1.011203e-15 1.011203e-15 43 border

1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.220446e-16 44 border

1.000000e+00 1.000000e+00 2.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 45 border

2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 1.776357e-15 46 border

2.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 47 border

0.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 1.776357e-15 48 border

0.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 3.722623e-16 3.722623e-16 49 border

1.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 0.000000e+00 50 border

1.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 2.000000e+00 0.000000e+00 51 border

2.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 1.776357e-15 52 border

2.000000e+00 2.000000e+00 2.000000e+00 4.000000e+00 4.000000e+00 8.881784e-16 53 border

||u-u\*||/||u\*|| = 3.839572e-16

||u-u\*|| = 3.325167e-15

1. **Исследования**



**МСГ 1000 узлов**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Кол-во итераций** | **Время решения, сек** |  |  |
| 1e-04 | 1e+02 | 0e+00 | 8.81e-12 | 258 | 7.700000e-02 | 2.356138e-12 | 6.354032e-15 |
| 1e-11 | 255 | 7.700000e-02 | 2.509431e-12 | 6.767432e-15 |
| 1e-10 | 255 | 7.600000e-02 | 2.250035e-12 | 6.067893e-15 |
| 1e+04 | 8.81e-12 | 258 | 7.800000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e-11 | 258 | 9.600000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e-10 | 258 | 7.700000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e+08 | 8.81e-12 | 675 | 2.060000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e-11 | 675 | 2.020000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e-10 | 675 | 2.480000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e+04 | 0e+00 | 8.81e-12 | 463 | 1.380000e-01 | 2.078898e-10 | 5.606371e-13 |
| 1e-11 | 462 | 1.790000e-01 | 2.704708e-10 | 7.294056e-13 |
| 1e-10 | 441 | 1.320000e-01 | 2.445248e-10 | 6.594344e-13 |
| 1e+04 | 8.81e-12 | 454 | 1.360000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e-11 | 454 | 1.360000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e-10 | 454 | 1.370000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e+08 | 8.81e-12 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 1e-11 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 1e-10 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 8e+05 | 0e+00 | 8.81e-12 | 683 | 2.600000e-01 | 1.866418e-08 | 5.033355e-11 |
| 1e-11 | 686 | 2.040000e-01 | 1.712741e-08 | 4.618918e-11 |
| 1e-10 | 687 | 2.060000e-01 | 1.892983e-08 | 5.104995e-11 |
| 1e+04 | 8.81e-12 | 696 | 2.090000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e-11 | 696 | 2.410000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e-10 | 696 | 2.080000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e+08 | 8.81e-12 | 615 | 1.840000e-01 | 1.271652e-03 | 3.429390e-06 |
| 1e-11 | 615 | 1.840000e-01 | 1.271652e-03 | 3.429390e-06 |
| 1e-10 | 615 | 1.850000e-01 | 1.271652e-03 | 3.429390e-06 |
| 1e+03 | 1e+02 | 0e+00 | 8.81e-12 | 255 | 7.600000e-02 | 2.670551e-12 | 7.201941e-15 |
| 1e-11 | 254 | 7.700000e-02 | 2.049821e-12 | 5.527956e-15 |
| 1e-10 | 254 | 7.600000e-02 | 2.763689e-12 | 7.453115e-15 |
| 1e+04 | 8.81e-12 | 908 | 2.720000e-01 | 1.234291e+00 | 3.328634e-03 |
| 1e-11 | 909 | 2.720000e-01 | 1.239676e+00 | 3.343158e-03 |
| 1e-10 | 899 | 3.450000e-01 | 1.364506e+00 | 3.679800e-03 |
| 1e+08 | 8.81e-12 | 2215 | 6.630000e-01 | 1.187553e+08 | 3.202593e+05 |
| 1e-11 | 2213 | 8.490000e-01 | 1.243775e+08 | 3.354211e+05 |
| 1e-10 | 2235 | 6.670000e-01 | 1.440707e+08 | 3.885297e+05 |
| 1e+04 | 0e+00 | 8.81e-12 | 463 | 1.790000e-01 | 2.460335e-10 | 6.635030e-13 |
| 1e-11 | 468 | 1.410000e-01 | 2.793820e-10 | 7.534373e-13 |
| 1e-10 | 464 | 1.390000e-01 | 4.513420e-10 | 1.217179e-12 |
| 1e+04 | 8.81e-12 | 951 | 2.840000e-01 | 1.234061e+00 | 3.328015e-03 |
| 1e-11 | 957 | 2.900000e-01 | 1.252724e+00 | 3.378346e-03 |
| 1e-10 | 945 | 2.830000e-01 | 1.351849e+00 | 3.645667e-03 |
| 1e+08 | 8.81e-12 | 2218 | 6.660000e-01 | 1.283107e+08 | 3.460282e+05 |
| 1e-11 | 2225 | 6.650000e-01 | 1.290851e+08 | 3.481167e+05 |
| 1e-10 | 2238 | 8.720000e-01 | 1.516932e+08 | 4.090862e+05 |
| 8e+05 | 0e+00 | 8.81e-12 | 689 | 2.060000e-01 | 1.778680e-08 | 4.796744e-11 |
| 1e-11 | 689 | 2.640000e-01 | 3.394333e-08 | 9.153834e-11 |
| 1e-10 | 685 | 2.050000e-01 | 2.545746e-08 | 6.865365e-11 |
| 1e+04 | 8.81e-12 | 585 | 1.750000e-01 | 1.136910e+00 | 3.066018e-03 |
| 1e-11 | 587 | 1.760000e-01 | 1.043425e+00 | 2.813907e-03 |
| 1e-10 | 586 | 1.750000e-01 | 1.098670e+00 | 2.962893e-03 |
| 1e+08 | 8.81e-12 | 2230 | 6.670000e-01 | 1.227869e+08 | 3.311316e+05 |
| 1e-11 | 2223 | 8.410000e-01 | 1.189097e+08 | 3.206756e+05 |
| 1e-10 | 2233 | 6.790000e-01 | 1.600718e+08 | 4.316816e+05 |
| 1e+09 | 1e+02 | 0e+00 | 8.81e-12 | 902 | 2.690000e-01 | 7.431699e-01 | 2.004180e-03 |
| 1e-11 | 917 | 2.740000e-01 | 1.056385e+00 | 2.848858e-03 |
| 1e-10 | 1094 | 3.260000e-01 | 1.042179e+02 | 2.810548e-01 |
| 1e+04 | 8.81e-12 | 3791 | 1.132000e+00 | 1.196128e+12 | 3.225718e+09 |
| 1e-11 | 3794 | 1.327000e+00 | 1.326249e+12 | 3.576627e+09 |
| 1e-10 | 3809 | 1.137000e+00 | 1.350150e+12 | 3.641085e+09 |
| 1e+08 | 8.81e-12 | 30000 | 1.162000e+01 | 2.541073e+21 | 6.852765e+18 |
| 1e-11 | 30000 | 8.956000e+00 | 2.654550e+21 | 7.158789e+18 |
| 1e-10 | 30000 | 8.957000e+00 | 3.202865e+21 | 8.637484e+18 |
| 1e+04 | 0e+00 | 8.81e-12 | 956 | 2.850000e-01 | 8.048267e-01 | 2.170456e-03 |
| 1e-11 | 973 | 2.940000e-01 | 9.489772e-01 | 2.559201e-03 |
| 1e-10 | 1110 | 3.320000e-01 | 1.075141e+02 | 2.899439e-01 |
| 1e+04 | 8.81e-12 | 3882 | 1.523000e+00 | 1.180740e+12 | 3.184220e+09 |
| 1e-11 | 3847 | 1.150000e+00 | 1.169669e+12 | 3.154363e+09 |
| 1e-10 | 3844 | 1.165000e+00 | 1.322633e+12 | 3.566875e+09 |
| 1e+08 | 8.81e-12 | 30000 | 8.970000e+00 | 3.140670e+21 | 8.469758e+18 |
| 1e-11 | 30000 | 1.135200e+01 | 2.996304e+21 | 8.080431e+18 |
| 1e-10 | 30000 | 8.954000e+00 | 4.098286e+21 | 1.105225e+19 |
| 8e+05 | 0e+00 | 8.81e-12 | 28564 | 8.573000e+00 | 6.265629e+03 | 1.689714e+01 |
| 1e-11 | 30000 | 8.954000e+00 | 4.535235e+04 | 1.223062e+02 |
| 1e-10 | 1555 | 4.730000e-01 | 1.132154e+02 | 3.053191e-01 |
| 1e+04 | 8.81e-12 | 3847 | 1.149000e+00 | 1.194472e+12 | 3.221252e+09 |
| 1e-11 | 3846 | 1.475000e+00 | 1.370794e+12 | 3.696757e+09 |
| 1e-10 | 3841 | 1.146000e+00 | 1.317878e+12 | 3.554051e+09 |
| 1e+08 | 8.81e-12 | 30000 | 8.976000e+00 | 3.090694e+21 | 8.334982e+18 |
| 1e-11 | 30000 | 9.005000e+00 | 2.988677e+21 | 8.059862e+18 |
| 1e-10 | 30000 | 9.010000e+00 | 3.319746e+21 | 8.952688e+18 |

Можно выделить такие значения параметров, при которых были получены наилучшее результаты:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Кол-во итераций** | **Время решения, сек** |  |  |
| 1e-04 | 1e+02 | 0e+00 | 8.81e-12 | 258 | 7.700000e-02 | 2.356138e-12 | 6.354032e-15 |
| 1e-11 | 255 | 7.700000e-02 | 2.509431e-12 | 6.767432e-15 |
| 1e-10 | 255 | 7.600000e-02 | 2.250035e-12 | 6.067893e-15 |
| 1e+04 | 8.81e-12 | 258 | 7.800000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e-11 | 258 | 9.600000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e-10 | 258 | 7.700000e-02 | 8.431911e-12 | 2.273917e-14 |
| 1e+08 | 8.81e-12 | 675 | 2.060000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e-11 | 675 | 2.020000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e-10 | 675 | 2.480000e-01 | 1.222727e-06 | 3.297450e-09 |
| 1e+04 | 0e+00 | 8.81e-12 | 463 | 1.380000e-01 | 2.078898e-10 | 5.606371e-13 |
| 1e-11 | 462 | 1.790000e-01 | 2.704708e-10 | 7.294056e-13 |
| 1e-10 | 441 | 1.320000e-01 | 2.445248e-10 | 6.594344e-13 |
| 1e+04 | 8.81e-12 | 454 | 1.360000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e-11 | 454 | 1.360000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e-10 | 454 | 1.370000e-01 | 1.344673e-09 | 3.626312e-12 |
| 1e+08 | 8.81e-12 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 1e-11 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 1e-10 | 300 | 9.000000e-02 | 5.112716e-06 | 1.378797e-08 |
| 8e+05 | 0e+00 | 8.81e-12 | 683 | 2.600000e-01 | 1.866418e-08 | 5.033355e-11 |
| 1e-11 | 686 | 2.040000e-01 | 1.712741e-08 | 4.618918e-11 |
| 1e-10 | 687 | 2.060000e-01 | 1.892983e-08 | 5.104995e-11 |
| 1e+04 | 8.81e-12 | 696 | 2.090000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e-11 | 696 | 2.410000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e-10 | 696 | 2.080000e-01 | 9.362788e-08 | 2.524956e-10 |
| 1e+03 | 1e+02 | 0e+00 | 8.81e-12 | 255 | 7.600000e-02 | 2.670551e-12 | 7.201941e-15 |
| 1e-11 | 254 | 7.700000e-02 | 2.049821e-12 | 5.527956e-15 |
| 1e-10 | 254 | 7.600000e-02 | 2.763689e-12 | 7.453115e-15 |
| 1e+04 | 8.81e-12 | 463 | 1.790000e-01 | 2.460335e-10 | 6.635030e-13 |
| 1e-11 | 468 | 1.410000e-01 | 2.793820e-10 | 7.534373e-13 |
| 1e-10 | 464 | 1.390000e-01 | 4.513420e-10 | 1.217179e-12 |
| 8e+05 | 8.81e-12 | 689 | 2.060000e-01 | 1.778680e-08 | 4.796744e-11 |
| 1e-11 | 689 | 2.640000e-01 | 3.394333e-08 | 9.153834e-11 |
| 1e-10 | 685 | 2.050000e-01 | 2.545746e-08 | 6.865365e-11 |

**МСГ 27000 узлов**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Кол-во итераций** | **Время решения, сек** |  |  |
| 1e-04 | 1e+02 | 0e+00 | 8.81e-12 | 1800 | 3.840100e+01 | 4.024038e-12 | 2.104365e-15 |
| 1e-11 | 1798 | 2.164400e+01 | 4.022089e-12 | 2.103346e-15 |
| 1e-10 | 1807 | 2.149100e+01 | 4.915044e-12 | 2.570316e-15 |
| 1e+04 | 8.81e-12 | 1807 | 3.628000e+01 | 4.456076e-12 | 2.330299e-15 |
| 1e-11 | 1807 | 2.087500e+01 | 4.456076e-12 | 2.330299e-15 |
| 1e-10 | 1798 | 2.059500e+01 | 4.183928e-12 | 2.187979e-15 |
| 1e+08 | 8.81e-12 | 447 | 4.917000e+00 | 3.700395e-09 | 1.935116e-12 |
| 1e-11 | 447 | 4.824000e+00 | 3.700395e-09 | 1.935116e-12 |
| 1e-10 | 446 | 5.134000e+00 | 3.700364e-09 | 1.935100e-12 |
| 1e+04 | 0e+00 | 8.81e-12 | 4631 | 5.526800e+01 | 1.840318e-10 | 9.623918e-14 |
| 1e-11 | 4583 | 5.028600e+01 | 2.234097e-10 | 1.168318e-13 |
| 1e-10 | 4592 | 5.372500e+01 | 2.113995e-10 | 1.105511e-13 |
| 1e+04 | 8.81e-12 | 4619 | 5.079600e+01 | 3.411185e-10 | 1.783875e-13 |
| 1e-11 | 4619 | 5.215300e+01 | 3.411185e-10 | 1.783875e-13 |
| 1e-10 | 4608 | 5.157500e+01 | 3.412264e-10 | 1.784439e-13 |
| 1e+08 | 8.81e-12 | 3299 | 5.852300e+01 | 1.073642e-06 | 5.614596e-10 |
| 1e-11 | 3299 | 3.706300e+01 | 1.073642e-06 | 5.614596e-10 |
| 1e-10 | 3299 | 3.499900e+01 | 1.073642e-06 | 5.614596e-10 |
| 8e+05 | 0e+00 | 8.81e-12 | 7768 | 6.579900e+01 | 1.792482e-08 | 9.373759e-12 |
| 1e-11 | 7795 | 6.345100e+01 | 2.042845e-08 | 1.068303e-11 |
| 1e-10 | 7878 | 6.442800e+01 | 1.600665e-08 | 8.370656e-12 |
| 1e+04 | 8.81e-12 | 7848 | 6.574400e+01 | 2.291232e-08 | 1.198197e-11 |
| 1e-11 | 7848 | 6.900500e+01 | 2.291232e-08 | 1.198197e-11 |
| 1e-10 | 7848 | 6.374600e+01 | 2.291232e-08 | 1.198197e-11 |
| 1e+03 | 1e+02 | 0e+00 | 8.81e-12 | 1795 | 1.425900e+01 | 5.168741e-12 | 2.702986e-15 |
| 1e-11 | 1811 | 2.215600e+01 | 7.148262e-12 | 3.738174e-15 |
| 1e-10 | 1810 | 3.700200e+01 | 1.078908e-11 | 5.642133e-15 |
| 1e+04 | 8.81e-12 | 4588 | 4.875100e+01 | 2.006124e-10 | 1.049100e-13 |
| 1e-11 | 4608 | 5.197400e+01 | 2.092784e-10 | 1.094418e-13 |
| 1e-10 | 4637 | 5.681600e+01 | 2.031389e-10 | 1.062312e-13 |
| 8e+05 | 8.81e-12 | 7822 | 8.745300e+01 | 1.530967e-08 | 8.006169e-12 |
| 1e-11 | 7790 | 8.903900e+01 | 1.478231e-08 | 7.730390e-12 |
| 1e-10 | 7885 | 8.913300e+01 | 1.703368e-08 | 8.907737e-12 |

1. **Выводы**

В результате исследований можно сказать, что для сеток с небольшим и большим количеством узлов выполняется следующее:

1. При фиксировании значений и увеличении значения параметра можно увидеть, что при метод расходится. При количество итераций, точность и время решения примерно одинаковое.
2. При фиксировании значений , с увеличением значения параметра наблюдается возрастание погрешности, метод сходится за большее время и количество итераций.
3. При фиксировании значений и увеличении значения параметра можно увидеть, что при точность решения возрастает, а затем при начинает падать. Начинает возрастать погрешность, время решения и количество итераций.
4. При фиксировании значений и увеличении значения параметра изменения не наблюдаются.
5. **Текст программы**

**Файл “Vector.h”**

#pragma once

#include <vector>

#include <iomanip>

#include <fstream>

using namespace std;

// Умножение вектора на число

vector<double> operator \* (const double& val, vector<double> vec)

{

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

vec[i] \*= val;

return vec;

}

// Сложение векторов

vector<double> operator + (vector<double> vec1, const vector<double>& vec2)

{

for (size\_t i = 0; i < vec1.size(); i++)

vec1[i] += vec2[i];

return vec1;

}

// Вычитание векторов

vector<double> operator - (vector<double> vec1, const vector<double>& vec2)

{

for(size\_t i = 0; i < vec1.size(); i++)

vec1[i] -= vec2[i];

return vec1;

}

double ScalarMult(const vector<double>& vec1, const vector<double>& vec2)

{

double res = 0;

for(int i = 0; i < vec1.size(); i++)

res += vec1[i] \* vec2[i];

return res;

}

double operator \* (const vector<double>& vec1, const vector<double>& vec2)

{

return ScalarMult(vec1, vec2);

}

double Norm(const vector<double>& vec)

{

return sqrt(ScalarMult(vec, vec));

}

**Файл “Matrix.h”**

#pragma once

#include <vector>

#include <fstream>

using namespace std;

class Matrix

{

public:

int size; // Размер матрицы

int tr\_size; // Количество элементов в треугольнике

vector<int> ind; // Указатели начала строк

vector<int> columns\_ind; // Номера столбцов внедиагональных элементов

vector<double> top\_tr; // Верхний треугольник

vector<double> bot\_tr; // Нижний треугольник

vector<double> diag; // Диагональ

Matrix(const int& t\_size, const int& t\_tr\_size) : size(t\_size), tr\_size(t\_tr\_size)

{

top\_tr = vector<double>(tr\_size);

bot\_tr = vector<double>(tr\_size);

columns\_ind = vector<int>(tr\_size);

diag = vector<double>(size);

ind = vector<int>(size + 1);

}

Matrix(const int& t\_size) : size(t\_size)

{

tr\_size = size \* (size - 1) / 2;

top\_tr.resize(tr\_size);

bot\_tr.resize(tr\_size);

columns\_ind.resize(tr\_size);

diag.resize(size);

ind.resize(size + 1);

ind[0] = ind[1] = 0;

diag[0] = 1.0;

for (int i = 1; i < size; i++)

{

int i0 = ind[i + 0];

int i1 = ind[i + 1] = ind[i + 0] + i;

for (int j = 0, k = i0; j < i; j++, k++)

columns\_ind[k] = j;

}

}

Matrix(const Matrix& mat)

{

size = mat.size;

tr\_size = mat.tr\_size;

top\_tr = mat.top\_tr;

bot\_tr = mat.bot\_tr;

diag = mat.diag;

ind = mat.ind;

columns\_ind = mat.columns\_ind;

}

Matrix()

{

}

// Получение диагональной факторизации матрицы

void DiagFact(Matrix& fact)

{

fact.diag = diag;

for (int i = 0; i < size + 1; i++)

fact.ind[i] = 0;

}

// Функция умножения матрицы на вектор vec, результат в res

void MatVecMult(const vector<double>& vec, vector<double>& res,

const vector<double>& bot\_tr, const vector<double>& top\_tr)

{

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

res[i] = 0;

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

{

res[i] += vec[i] \* diag[i];

int prof\_len = ind[i + 1] - ind[i];

for (int k = 0; k < prof\_len; k++)

{

int i\_in\_gg = ind[i] + k;

int j = columns\_ind[i\_in\_gg];

res[i] += vec[j] \* bot\_tr[i\_in\_gg];

res[j] += vec[i] \* top\_tr[i\_in\_gg];

}

}

}

// Функция считывания диагонали и треугольников из файлов

void ReadDiTr(const string& file\_name)

{

ifstream fin;

fin.open(file\_name);

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

fin >> diag[i];

for(int i = 0; i < tr\_size; i++)

{

double t;

fin >> t;

top\_tr[i] = bot\_tr[i] = t;

}

fin.close();

}

// Зануление всех элементов матрицы

void ResetValues()

{

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

diag[i] = 0;

for(int i = 0; i < tr\_size; i++)

{

bot\_tr[i] = 0;

top\_tr[i] = 0;

}

}

// Умножение матрицы на число

void Mult(const double& val)

{

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

diag[i] \*= val;

for(int i = 0; i < tr\_size; i++)

{

bot\_tr[i] \*= val;

top\_tr[i] \*= val;

}

}

};

**Файл “SLAE.h”**

#pragma once

#include "Vector.h"

#include "Matrix.h"

using namespace std;

class SLAE

{

public:

int maxiter; // Максимальное количество итераций

double eps; // Велечина требуемой относительной невязки

vector<double> b; // Вектор правой части

vector<double> t; // Вспомогательный вектор для МСГ

vector<double> tt; // Вспомогательный вектор для МСГ

vector<double> rk1; // Вектор невязки на перд. итерации МСГ

vector<double> zk1; // Вектор спуска на пред. итерации МСГ

vector<double> AtAzk1; // Вспомогательный вектор для МСГ

SLAE(int size, int \_maxiter, double \_eps)

{

maxiter = \_maxiter;

eps = \_eps;

b.resize(size);

t.resize(size);

tt.resize(size);

rk1.resize(size);

zk1.resize(size);

AtAzk1.resize(size);

}

SLAE()

{

}

// Метод сопряженных градиентов, возвращает количество итераций

int ConjGradMethod(vector<double>& xk1, vector<double>& res, Matrix& mat)

{

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

res[i] = xk1[i] = 0;

mat.MatVecMult(xk1, t, mat.bot\_tr, mat.top\_tr); // t = A \* x0

mat.MatVecMult(b - t, rk1, mat.top\_tr, mat.bot\_tr); // r0 = AT(f - A \* x0)

zk1 = rk1;

int k = 1;

while(k < maxiter)

{

mat.MatVecMult(zk1, t, mat.bot\_tr, mat.top\_tr); // t = A \* zk-1

mat.MatVecMult(t, AtAzk1, mat.top\_tr, mat.bot\_tr); // AtAzk1 = At \* A \* zk-1

double ak = (rk1 \* rk1) / (AtAzk1 \* zk1);

xk1 = xk1 + ak \* zk1;

double bk = rk1 \* rk1;

rk1 = rk1 - ak \* AtAzk1;

bk = (rk1 \* rk1) / bk;

zk1 = rk1 + bk \* zk1;

double disc = Norm(rk1) / Norm(b); // Относительная невязка

if(disc < eps)

break;

else

k++;

}

res = xk1;

return k;

}

// Метод сопряженных градиентов с предобусловденной неполной

// факторизацией матрицей, возвращает количество итераций

int ConjGradPredMethod(vector<double>& xk1, vector<double>& res, Matrix& mat,

SLAE& fac\_slae, Matrix& fac\_mat)

{

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

res[i] = xk1[i] = 0;

mat.MatVecMult(xk1, t, mat.bot\_tr, mat.top\_tr); // t = A \* x0

mat.MatVecMult(b - t, rk1, mat.top\_tr, mat.bot\_tr); // r0 = AT(f - A \* x0)

// Решаем z0 = M-1 \* r0

fac\_slae.b = rk1;

fac\_slae.ConjGradMethod(t, zk1, fac\_mat);

int k = 1;

while(k < maxiter)

{

// Решаем tt = M-1 \* rk-1

fac\_slae.b = rk1;

fac\_slae.ConjGradMethod(t, tt, fac\_mat);

mat.MatVecMult(zk1, t, mat.bot\_tr, mat.top\_tr); // t = A \* zk-1

mat.MatVecMult(t, AtAzk1, mat.top\_tr, mat.bot\_tr); // AtAzk1 = At \* A \* zk-1

double ak = (tt \* rk1) / (AtAzk1 \* zk1);

xk1 = xk1 + ak \* zk1;

double bk = tt \* rk1;

rk1 = rk1 - ak \* AtAzk1;

// Решаем tt = M-1 \* rk

fac\_slae.b = rk1;

fac\_slae.ConjGradMethod(t, tt, fac\_mat);

bk = (tt \* rk1) / bk;

zk1 = tt + bk \* zk1;

double disc = Norm(rk1) / Norm(b);

if(disc < eps)

break;

else

k++;

}

res = xk1;

return k;

}

};

**Файл “Test.h”**

#pragma once

#pragma once

using namespace std;

class Test

{

public:

int N, I, J, L, P;

Test(const int& n, const int& i, const int& j, const int& l, const int& p) : N(n), I(i),

J(j), L(l), P(p) {};

Test() : I(0), J(0), L(0), P(0) {};

double us(const double& x, const double& y, const double& z)

{

switch(N)

{

case(0): return 2.0;

case(1): return y + z;

case(2): return y \* z;

};

}

double uc(const double& x, const double& y, const double& z)

{

switch(N)

{

case(0): return 5.0;

case(1): return x + y;

case(2): return x \* y;

};

}

double fs(const double& x, const double& y, const double& z)

{

return -1 \* divlambdagradus(x, y, z) +

-1 \* omega() \* sigma() \* uc(x, y, z) -

omega() \* omega() \* chi() \* us(x, y, z);

}

double fc(const double& x, const double& y, const double& z)

{

return -1 \* divlambdagraduc(x, y, z) +

omega() \* sigma() \* us(x, y, z) -

omega() \* omega() \* chi() \* uc(x, y, z);

}

double divlambdagradus(const double& x, const double& y, const double& z)

{

switch(N)

{

default: return 0;

};

}

double divlambdagraduc(const double& x, const double& y, const double& z)

{

switch(N)

{

default: return 0;

};

}

double lambda()

{

switch (I)

{

case(0): return 1e2;

case(1): return 1e4;

case(2): return 8e5;

case(3): return 1;

};

}

double sigma()

{

switch (J)

{

case(0): return 0;

case(1): return 1e4;

case(2): return 1e8;

case(3): return 1;

};

}

double chi()

{

switch (L)

{

case(0): return 8.81e-12;

case(1): return 1e-11;

case(2): return 1e-10;

case(3): return 1;

};

}

double omega()

{

switch (P)

{

case(0): return 1e-4;

case(1): return 1e3;

case(2): return 1e9;

case(3): return 1;

};

}

void Print(ofstream& fout)

{

fout << scientific << omega() << "\t" << lambda() << "\t";

fout << sigma() << "\t" << chi() << "\t";

}

};

**Файл “HarmonicProblem.cpp”**

#pragma once

#include <iostream>

#include "Matrix.h"

#include "SLAE.h"

#include "Test.h"

#include <ctime>

using namespace std;

class HarmonicProblem

{

public:

Matrix global; // Глобальная матрица

Matrix fac\_global; // Неполная факторизация глобальной матрицы

Matrix GMM, MGM, MMG, MMM; // Вспомогательные матрицы для вычисления элементов

// локальных матриц и векторов правых частей

SLAE slae; // Решатель системы без предобуславливания

SLAE fac\_slae; // Решатель системы c предобуславливанием

vector<double> b; // Глобальный вектор правой части

vector<double> loc\_b; // Локальный вектор правой части

vector<double> true\_solution; // Точное решение

vector<double> solution; // Решение

vector<int> location; // Положение на сетке для каждого узла

Test test; // Информация о значениях функции,

// парматров лямбда и гамма

int elems\_count; // Количество конечных элементов

int nodes\_count; // Количество узлов

vector<double> x\_nodes; // Координаты сетки по x

vector<double> y\_nodes; // Координаты сетки по y

vector<double> z\_nodes; // Координаты сетки по z

int x\_nodes\_count; // Количество узлов по x

int y\_nodes\_count; // Количество узлов по y

int z\_nodes\_count; // Количество узлов по z

vector<vector<int>> regions; // Информация о подобластях (регионах)

int regions\_count; // Количество регионов

vector<int> x\_cords\_i; //Индексы исходных координатных линий в векторе сетки по x

vector<int> y\_cords\_i; //Индексы исходных координатных линий в векторе сетки по y

vector<int> z\_cords\_i; //Индексы исходных координатных линий в векторе сетки по z

vector<vector<int>> boundaries; // Информация о краевых условиях

int bound\_count; // Количество краевых условий

// Вспомогательная функция для формирования сетки

void ReadFormGridHelp(int& t\_nodes\_count, vector<double>& t\_nodes,

vector<int>& t\_cords\_i, ifstream& fin)

{

int t\_coords\_count;

fin >> t\_coords\_count;

vector<double> t\_coords(t\_coords\_count);

for(int i = 0; i < t\_coords\_count; i++)

fin >> t\_coords[i];

t\_nodes\_count = 1;

t\_nodes = vector<double>(1);

t\_nodes[0] = t\_coords[0];

vector<int> n\_t(t\_coords\_count - 1);

for(int i = 0; i < t\_coords\_count - 1; i++)

{

fin >> n\_t[i];

t\_nodes.resize(t\_nodes\_count + n\_t[i]);

double h = (t\_coords[i + 1] - t\_coords[i]) / n\_t[i];

for(int j = 0; j < n\_t[i]; j++)

t\_nodes[j + t\_nodes\_count] = t\_nodes[t\_nodes\_count - 1] + h \* (j + 1);

t\_nodes\_count += n\_t[i];

}

// Пересчет индексов границ подобластей

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

n\_t[i] += n\_t[i - 1];

t\_cords\_i = vector<int>(n\_t.size() + 1);

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

t\_cords\_i[i + 1] = n\_t[i];

elems\_count \*= n\_t[n\_t.size() - 1];

}

// Считывание и формирование сетки из файла file\_name

void ReadFormGrid(const string& file\_name)

{

ifstream fin(file\_name);

string fake;

elems\_count = 1;

// Считывание координатных линий и формирование сетки по x

fin >> fake;

ReadFormGridHelp(x\_nodes\_count, x\_nodes, x\_cords\_i, fin);

// Считывание координатных линий и формирование сетки по y

fin >> fake;

ReadFormGridHelp(y\_nodes\_count, y\_nodes, y\_cords\_i, fin);

// Считывание координатных линий и формирование сетки по z

fin >> fake;

ReadFormGridHelp(z\_nodes\_count, z\_nodes, z\_cords\_i, fin);

// Считывание информации о подобластях

fin >> fake;

fin >> regions\_count;

regions = vector<vector<int>>(regions\_count, vector<int>(6));

for(int i = 0; i < regions\_count; i++)

fin >> regions[i][0] >> regions[i][1] >> regions[i][2] >> regions[i][3]

>> regions[i][4] >> regions[i][5];

fin.close();

// Перерасчет индексов границ подобластей в соответствии с разбиением сетки

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

regions[reg\_i][0] = x\_cords\_i[regions[reg\_i][0]];

regions[reg\_i][1] = x\_cords\_i[regions[reg\_i][1]];

regions[reg\_i][2] = y\_cords\_i[regions[reg\_i][2]];

regions[reg\_i][3] = y\_cords\_i[regions[reg\_i][3]];

regions[reg\_i][4] = z\_cords\_i[regions[reg\_i][4]];

regions[reg\_i][5] = z\_cords\_i[regions[reg\_i][5]];

}

nodes\_count = x\_nodes\_count \* y\_nodes\_count \* z\_nodes\_count;

}

// Считывание информации о краевых условиях из файла file\_name

void ReadBoundaries(const string& file\_name)

{

ifstream fin(file\_name);

fin >> bound\_count;

boundaries = vector<vector<int>>(bound\_count, vector<int>(7));

for(int bound\_i = 0; bound\_i < bound\_count; bound\_i++)

{

for(int i = 0; i < 7; i++)

fin >> boundaries[bound\_i][i];

boundaries[bound\_i][1] = x\_cords\_i[boundaries[bound\_i][1]];

boundaries[bound\_i][2] = x\_cords\_i[boundaries[bound\_i][2]];

boundaries[bound\_i][3] = y\_cords\_i[boundaries[bound\_i][3]];

boundaries[bound\_i][4] = y\_cords\_i[boundaries[bound\_i][4]];

boundaries[bound\_i][5] = z\_cords\_i[boundaries[bound\_i][5]];

boundaries[bound\_i][6] = z\_cords\_i[boundaries[bound\_i][6]];

}

fin.close();

}

// Считывание вспомогательных матриц для формирования

// матриц жесткости и массы

void ReadMatrices()

{

GMM = Matrix(8);

GMM.ReadDiTr("data/GMM.txt");

MGM = Matrix(8);

MGM.ReadDiTr("data/MGM.txt");

MMG = Matrix(8);

MMG.ReadDiTr("data/MMG.txt");

MMM = Matrix(8);

MMM.ReadDiTr("data/MMM.txt");

}

// Выделение памяти под массивы

void InitializeMemory()

{

slae = SLAE(nodes\_count \* 2, 30000, 1e-20);

fac\_slae = SLAE(nodes\_count \* 2, 30000, 1e-20);

global.ind = vector<int>(nodes\_count \* 2 + 1);

b = vector<double>(nodes\_count \* 2);

solution = vector<double>(nodes\_count \* 2);

true\_solution = vector<double>(nodes\_count \* 2);

location = vector<int>(nodes\_count \* 2);

fac\_global = Matrix(nodes\_count \* 2, 0);

}

// Заполнение массива global\_indices индексами, соответствующими глобальной номерации

// узлов конечного элемента с номером elem\_index(индексация с нуля)

void CalcGlobalIndices(int elem\_i, vector<int>& global\_indices)

{

int n\_coords = x\_nodes\_count / 2 + 1;

int k = elem\_i % (x\_nodes\_count - 1) + x\_nodes\_count \* floor(elem\_i /

(x\_nodes\_count - 1));

k = k % (x\_nodes\_count \* (y\_nodes\_count - 1)) + (x\_nodes\_count \* y\_nodes\_count) \*

floor(k / (x\_nodes\_count \* (y\_nodes\_count - 1)));

global\_indices[0] = (k + 0);

global\_indices[1] = (k + 1);

global\_indices[2] = (k + x\_nodes\_count + 0);

global\_indices[3] = (k + x\_nodes\_count + 1);

global\_indices[4] = (k + x\_nodes\_count \* y\_nodes\_count + 0);

global\_indices[5] = (k + x\_nodes\_count \* y\_nodes\_count + 1);

global\_indices[6] = (k + x\_nodes\_count \* y\_nodes\_count + x\_nodes\_count + 0);

global\_indices[7] = (k + x\_nodes\_count \* y\_nodes\_count + x\_nodes\_count + 1);

}

// Поиск региона по номеру конечного элемента

int CalcRegionIndex(const int& elem\_i)

{

int n\_coords = x\_nodes\_count / 2 + 1;

int x0 = (elem\_i) % (n\_coords - 1) \* 2 + 1;

int y0 = floor((elem\_i) / (n\_coords - 1)) \* 2 + 1;

int found\_reg\_i = -1;

for(int reg\_i = 0; reg\_i < regions\_count; reg\_i++)

{

if(regions[reg\_i][0] <= x0 && x0 <= regions[reg\_i][1] &&

regions[reg\_i][2] <= y0 && y0 <= regions[reg\_i][3])

{

found\_reg\_i = reg\_i;

break;

}

}

return found\_reg\_i;

}

// Вспомогательная функция для формирования портрета

void IncertToRow(Matrix& mat, const int& r, const int& c)

{

int i\_in\_jg = mat.ind[r];

int prof\_len = mat.ind[r + 1] - mat.ind[r];

bool found = false;

for(int k = i\_in\_jg; k < i\_in\_jg + prof\_len; k++)

if(mat.columns\_ind[k] == c)

{

found = true;

break;

}

if(!found)

{

for(int l = r + 1; l < mat.ind.size(); l++)

mat.ind[l]++;

int k = i\_in\_jg;

while((k < i\_in\_jg + prof\_len) && mat.columns\_ind[k] < c)

k++;

mat.columns\_ind.insert(mat.columns\_ind.begin() + k, c);

}

}

// Формирование портрета глобальной матрицы с учетом блочной специфики

void FormGlobalPortrait()

{

global.ind[0] = global.ind[1] = 0;

for(int elem\_i = 0; elem\_i < elems\_count; elem\_i++)

{

int reg\_i = CalcRegionIndex(elem\_i);

//if(reg\_i != -1)

{

vector<int> global\_indices(8);

CalcGlobalIndices(elem\_i, global\_indices);

vector<vector<vector<int>>> help(8);

for(int i = 0; i < 8; i++)

help[i].resize(8);

for(int i = 0; i < 8; i++)

for(int j = 0; j < 8; j++)

help[i][j] = { global\_indices[i], global\_indices[j] };

for(int i = 0; i < 8; i++)

{

IncertToRow(global, help[i][i][0] \* 2 + 1, help[i][i][1] \* 2);

}

for(int i = 0; i < 8; i++)

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

{

IncertToRow(global, help[i][j][0] \* 2, help[i][j][1] \* 2);

IncertToRow(global, help[i][j][0] \* 2, help[i][j][1] \* 2 + 1);

IncertToRow(global, help[i][j][0] \* 2 + 1, help[i][j][1] \* 2);

IncertToRow(global, help[i][j][0] \* 2 + 1, help[i][j][1] \* 2 + 1);

}

}

}

global.size = global.ind.size() - 1;

global.diag.resize(global.size);

global.tr\_size = global.columns\_ind.size();

global.bot\_tr.resize(global.tr\_size);

global.top\_tr.resize(global.tr\_size);

}

// Формирование портрета матрицы mat

void FormPortrait(Matrix& mat)

{

mat.ind[0] = mat.ind[1] = 0;

for(int elem\_i = 0; elem\_i < elems\_count; elem\_i++)

{

int reg\_i = CalcRegionIndex(elem\_i);

//if(reg\_i != -1)

{

vector<int> mat\_indices(8);

CalcGlobalIndices(elem\_i, mat\_indices);

vector<vector<vector<int>>> help(8);

for(int i = 0; i < 8; i++)

help[i].resize(8);

for(int i = 0; i < 8; i++)

for(int j = 0; j < 8; j++)

help[i][j] = { mat\_indices[i], mat\_indices[j] };

for(int i = 0; i < 8; i++)

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

IncertToRow(mat, help[i][j][0], help[i][j][1]);

}

}

mat.size = mat.ind.size() - 1;

mat.diag.resize(mat.size);

mat.tr\_size = mat.columns\_ind.size();

mat.bot\_tr.resize(mat.tr\_size);

mat.top\_tr.resize(mat.tr\_size);

}

// Добавление к элменту матрицы с индексом [row\_i][col\_i] значения val

void AddToMat(Matrix& mat, const int& row\_i, const int& col\_i, const double& val\_l,

const double& val\_u)

{

if(row\_i == col\_i)

mat.diag[row\_i] += val\_l;

else

{

int beg\_prof = mat.ind[row\_i];

int end\_prof = mat.ind[row\_i + 1];

for(int i\_in\_prof = beg\_prof; i\_in\_prof < end\_prof; i\_in\_prof++)

{

if(mat.columns\_ind[i\_in\_prof] == col\_i)

{

mat.bot\_tr[i\_in\_prof] += val\_l;

mat.top\_tr[i\_in\_prof] += val\_u;

break;

}

}

}

}

// Находит координаты узлов конечного элемента

// с номером elem\_index(индексация с нуля)

void CalcElemNodes(const int& elem\_i, vector<double>& x\_elem\_nodes,

vector<double>& y\_elem\_nodes, vector<double>& z\_elem\_nodes)

{

int n\_x = x\_nodes\_count - 1;

int n\_y = y\_nodes\_count - 1;

int n\_z = z\_nodes\_count - 1;

int x0 = elem\_i % n\_x;

int y0 = (int)floor(elem\_i / n\_x) % n\_y;

int z0 = (int)floor(elem\_i / (n\_x \* n\_y));

x\_elem\_nodes[0] = x\_nodes[x0];

x\_elem\_nodes[1] = x\_nodes[x0 + 1];

y\_elem\_nodes[0] = y\_nodes[y0];

y\_elem\_nodes[1] = y\_nodes[y0 + 1];

z\_elem\_nodes[0] = z\_nodes[z0];

z\_elem\_nodes[1] = z\_nodes[z0 + 1];

//cout << x0 << " " << y0 << " " << z0 << endl;

}

// Сборка глобальной матрицы

void AssembleGlobalMatrix()

{

vector<int> global\_indices(8);

for(int i = 0; i < global.tr\_size; i++)

{

global.bot\_tr[i] = global.top\_tr[i] = 0;

}

for(int i = 0; i < nodes\_count \* 2; i++)

{

global.diag[i] = b[i] = 0;

}

for(int elem\_i = 0; elem\_i < elems\_count; elem\_i++)

{

int reg\_i = CalcRegionIndex(elem\_i);

CalcGlobalIndices(elem\_i, global\_indices);

{

vector<double> x\_elem\_nodes(2); // Координаты конечного элемента по x

vector<double> y\_elem\_nodes(2); // Координаты конечного элемента по y

vector<double> z\_elem\_nodes(2); // Координаты конечного элемента по z

CalcElemNodes(elem\_i, x\_elem\_nodes, y\_elem\_nodes, z\_elem\_nodes);

double hx = x\_elem\_nodes[1] - x\_elem\_nodes[0];

double hy = y\_elem\_nodes[1] - y\_elem\_nodes[0];

double hz = z\_elem\_nodes[1] - z\_elem\_nodes[0];

vector<double> local\_fs(8);

vector<double> local\_fc(8);

for(int i = 0; i < 8; i++)

{

double x = x\_elem\_nodes[i % 2];

double y = y\_elem\_nodes[(int)floor(i / 2) % 2];

double z = z\_elem\_nodes[(int)floor(i / 4) % 4];

local\_fs[i] = test.fs(x, y, z);

local\_fc[i] = test.fc(x, y, z);

true\_solution[global\_indices[i] \* 2 ] = test.us(x, y, z);

true\_solution[global\_indices[i] \* 2 + 1] = test.uc(x, y, z);

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

{

double p = 0;

double c = 0;

if(i == j)

{

p = test.lambda() \* (hy \* hz / (hx \* 36) \* GMM.diag[i] + hx \* hz /

(hy \* 36) \* MGM.diag[i] + hx \* hy / (hz \* 36) \* MMG.diag[i]);

p -= test.omega() \* test.omega() \* test.chi() \* hx \* hy \* hz / 216.0 \*

MMM.diag[i];

c = test.omega() \* test.sigma() \* hx \* hy \* hz / 216.0 \* MMM.diag[i];

}

else

{

int tr\_i = MMM.ind[i] + j;

p = test.lambda() \* (hy \* hz / (hx \* 36) \* GMM.bot\_tr[tr\_i] + hx \* hz /

(hy \* 36) \* MGM.bot\_tr[tr\_i] + hx \* hy / (hz \* 36) \* MMG.bot\_tr[tr\_i]);

p -= test.omega() \* test.omega() \* test.chi() \* hx \* hy \* hz / 216.0 \*

MMM.bot\_tr[tr\_i];

c = test.omega() \* test.sigma() \* hx \* hy \* hz / 216.0 \*

MMM.bot\_tr[tr\_i];

}

AddToMat(global, global\_indices[i] \* 2, global\_indices[j] \* 2, p, p);

AddToMat(global, global\_indices[i] \* 2 + 1, global\_indices[j]\*2+1, p, p);

AddToMat(global, global\_indices[i] \* 2 + 1, global\_indices[j] \* 2, c, -c);

AddToMat(global, global\_indices[i] \* 2, global\_indices[j] \* 2 + 1, -c, c);

}

}

vector<double> local\_bs(8);

vector<double> local\_bc(8);

MMM.MatVecMult(local\_fs, local\_bs, MMM.bot\_tr, MMM.top\_tr);

MMM.MatVecMult(local\_fc, local\_bc, MMM.bot\_tr, MMM.top\_tr);

for(int i = 0; i < 8; i++)

{

b[global\_indices[i] \* 2 ] += hx \* hy \* hz / 216.0 \* local\_bs[i];

b[global\_indices[i] \* 2 + 1] += hx \* hy \* hz / 216.0 \* local\_bc[i];

}

}

}

}

// Учет первых краевых условий на строкес с номером line\_i

void FirstBoundOnLine(int line\_i, const double& x, const double& y, const double& z)

{

line\_i \*= 2;

global.diag[line\_i ] = 1;

global.diag[line\_i + 1] = 1;

true\_solution[line\_i ] = test.us(x, y, z);

true\_solution[line\_i + 1] = test.uc(x, y, z);

b[line\_i ] = test.us(x, y, z);

b[line\_i + 1] = test.uc(x, y, z);

for(int prof\_i = global.ind[line\_i]; prof\_i < global.ind[line\_i + 1]; prof\_i++)

global.bot\_tr[prof\_i] = 0;

for(int prof\_i = global.ind[line\_i + 1]; prof\_i < global.ind[line\_i + 2]; prof\_i++)

global.bot\_tr[prof\_i] = 0;

for(int i = 0; i < nodes\_count \* 2; i++)

{

for(int prof\_i = global.ind[i]; prof\_i < global.ind[i + 1]; prof\_i++)

{

if(global.columns\_ind[prof\_i] == line\_i || global.columns\_ind[prof\_i]==line\_i+1)

global.top\_tr[prof\_i] = 0;

}

}

}

// Учет первых краевых условий

void AccountFirstBound()

{

for(int x\_i = 0; x\_i < x\_nodes\_count; x\_i++)

{

for(int y\_i = 0; y\_i < y\_nodes\_count; y\_i++)

{

for(int z\_i = 0; z\_i < z\_nodes\_count; z\_i++)

{

for(int bound\_i = 0; bound\_i < bound\_count; bound\_i++)

{

if(boundaries[bound\_i][1] <= x\_i && x\_i <= boundaries[bound\_i][2] &&

boundaries[bound\_i][3] <= y\_i && y\_i <= boundaries[bound\_i][4] &&

boundaries[bound\_i][5] <= z\_i && z\_i <= boundaries[bound\_i][6])

{

int i = x\_i + y\_i \* x\_nodes\_count + z\_i \* x\_nodes\_count \*

y\_nodes\_count;

FirstBoundOnLine(i, x\_nodes[x\_i], y\_nodes[y\_i], z\_nodes[z\_i]);

location[i \* 2 ] = 1;

location[i \* 2 + 1] = 1;

break;

}

}

}

}

}

}

// Нахождение решения

void Solve(ofstream& fout)

{

slae.b = b;

global.DiagFact(fac\_global);

vector<double> x0(nodes\_count \* 2);

double start\_time = clock();

int iter = slae.ConjGradPredMethod(x0, solution, global, fac\_slae, fac\_global);

double end\_time = clock();

fout << iter << "\t" << (end\_time - start\_time) / CLOCKS\_PER\_SEC << "\t";

}

// Вывод решения на временном слое t в поток fout

void PrintSolution(ofstream& fout)

{

/\*fout << setw(14) << "x" << setw(14) << "y" << setw(14) << "z";

fout << setw(14) << "prec" << setw(14) << "calc" << setw(14) << "diff";

fout << setw(5) << "n" << " loc" << endl;\*/

double norm = 0, norm\_u = 0;

for(int z\_i = 0; z\_i < z\_nodes\_count; z\_i++)

{

for(int y\_i = 0; y\_i < y\_nodes\_count; y\_i++)

{

for(int x\_i = 0; x\_i < x\_nodes\_count; x\_i++)

{

int i = x\_i + y\_i \* x\_nodes\_count + z\_i \* x\_nodes\_count \* y\_nodes\_count;

double prec = true\_solution[i \* 2];

double calc = solution[i \* 2];

//if(x\_i % 32 == 0 && y\_i % 32 == 0)

/\*{

fout << scientific;

fout << setw(14) << x\_nodes[x\_i];

fout << setw(14) << y\_nodes[y\_i];

fout << setw(14) << z\_nodes[z\_i];

fout << setw(14) << prec;

fout << setw(14) << calc;

fout << setw(14) << abs(true\_solution[i \* 2] - solution[i \* 2]);

fout << fixed << setw(5) << i \* 2;

if(location[i \* 2] == 2)

fout << " outer";

else if(location[i \* 2] == 1)

fout << " border";

else

fout << " inner";

fout << endl;

}\*/

prec = true\_solution[i \* 2 + 1];

calc = solution[i \* 2 + 1];

//if(x\_i % 32 == 0 && y\_i % 32 == 0)

/\*{

fout << scientific;

fout << setw(14) << x\_nodes[x\_i];

fout << setw(14) << y\_nodes[y\_i];

fout << setw(14) << z\_nodes[z\_i];

fout << setw(14) << prec;

fout << setw(14) << calc;

fout << setw(14) << abs(true\_solution[i \* 2 + 1] - solution[i \* 2 + 1]);

fout << fixed << setw(5) << i \* 2 + 1;

if(location[i \* 2 + 1] == 2)

fout << " outer";

else if(location[i \* 2 + 1] == 1)

fout << " border";

else

fout << " inner";

fout << endl;

}\*/

norm\_u += prec \* prec;

norm += abs(prec - calc) \* abs(prec - calc);

}

}

}

/\*fout << "||u-u\*||/||u\*|| = " << scientific << sqrt(norm) / sqrt(norm\_u) << endl;

fout << "||u-u\*|| = " << scientific << sqrt(norm) << endl;\*/

fout << scientific << sqrt(norm) << "\t";

fout << scientific << sqrt(norm) / sqrt(norm\_u) << endl;

}

};

**Файл “main.cpp”**

#include <iostream>

#include "HarmonicProblem.h"

using namespace std;

int main()

{

HarmonicProblem hp;

hp.ReadFormGrid("data/grid.txt");

hp.ReadBoundaries("data/boundaries.txt");

hp.ReadMatrices();

ofstream fout("result.txt");

//for (int p = 0; p < 3; p++)

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

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

//for (int l = 0; l < 3; l++)

{

hp.InitializeMemory();

hp.FormGlobalPortrait();

// hp.test = Test(1, i, j, l, p);

hp.test = Test(1, 0, 2, 0, 2);

hp.AssembleGlobalMatrix();

hp.AccountFirstBound();

hp.test.Print(fout);

hp.Solve(fout);

hp.PrintSolution(fout);

}

fout.close();

}