|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
|  | | | |
| Курсовой проект по курсу | | | |
| **«Численные методы»** | | | |
|  | | | |
|  | Группа | ПМ-13 |
|  |  |
| Студент | Киреев д. н. |
|  |  |
|
|  |
| Новосибирск | | | |

2023

1. **Условие задачи.**

МКЭ для расчета двумерного поля давление в цилиндрической системе координат. Базисные функции билинейные на прямоугольниках. Краевые условия первого и второго рода.

1. **Теоретическая часть.**
   1. **Математическая модель для расчета многофазной фильтрации.**

Рассматриваем случай несжимаемых изотермических и неперемешиваемых фаз, где фазой будем считать часть смеси, входящие в которую компоненты фильтруются совместно.

Скорость фильтрации m-й фазы имеет вид:

, где - индивидуальный множитель к структурной проницаемости матрицы-породы, – вязкость m-й фазы, – давление, – капиллярное давление для m-й фазы, – плотность m-й фазы, – ускорение свободного падения, – координата вертикальной оси и K – тензор структурной проницаемости, для учета “трещиноватости” – резко повышенной проницаемости в одном из направлений, необязательно совпадающем с одной из осей координат.

Из закона сохранения массы вещества получаем для каждой фазы:

где – плотность источника и – пористость.

Так как плотность m-й фазы в случае несжимаемых изотермических и неперемешиваемых фаз для каждой фазы является константой, то разделим на неё и просуммируем по фазам:

Сумма насыщенностей всех фаз должна быть равна единице, поэтому уравнение преобразуется к виду:

На отдельных частях удаленной (внешней) границы расчетной области могут быть заданы главные краевые условия – пластовой давление, а на границе зон перфорации вторые краевые условия:

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

Эквивалентная вариационная постановка для уравнения выше имеет вид:

где – пробная функция из гильбертова пространства функций, имеющий суммируемые с квадратом производные. Искомую функцию P представим в виде линейного разложения: , где – веса базисных функций, аналогично пробная функция. В итоге для i и j получим:

В результате получается система конечноэлементных уравнений **Gp = b**, где **p** – вектор искомых весов, а компоненты матрицы **G** и вектора **b** определяются в виде:

При решении задачи влияние капиллярного давления пренебрежительно мало, поэтому его в расчете вектора можно не учитывать, учет гравитационной составляющей также не является усложнение, поэтому влияние гравитации тоже можно не учитывать, с учетом сказанного вектор b будет вычисляться в виде:

Локальная нумерация узлов выглядит следующим образом

**1**

**2**

**3**

**4**

Насчитывать глобальную матрицу будет с помощью локальной. После того, как посчитаем элементы локальной матрицы G, они должны быть добавлены в глобальную матрицу в соответствии с глобальной нумерацией узлов расчетной области. Глобальная нумерация происходит так же, как локальная (увеличение номера узла слева направо и снизу вверх).

Введем локальные базисные функции:

Тогда базисные функции будут иметь следующий вид:

Введем обозначение , где мы рассматриваем случай, когда – константа. Тогда уравнение для i-го j-го элемента матрицы G примет вид:

где

Распишем подынтегральное выражение:

Тогда, подставляя базисные функции, имеем:

где

Заметим, что одномерные интегралы по r и z это элементы матриц жесткости и массы для одномерных случаев, тогда мы имеем:

где

Матрицы H – вспомогательные матрицы, хранящие интегралы от произведения функции на ее производную.

Вектор правой части b будем считать с помощью представления функции плотности источника F в виде билинейного интерполянта по нашему базису:

, где – значения функции F в соответствующих узлах конечного элемента расчетной области. Тогда i-ая компонента вектора b имеет вид:

* 1. **Учет краевых условий.**

Аналогично тому, как считается вектор b (интерполяция функции F), учитываются вторые краевые условия после сборки глобальной матрицы, где используется интерполяция функции по базисным функциям () и считается “добавка” к вектору правой части:

– если 2-ое краевое лежит на горизонтальной границе

– если 2-ое краевое лежит на вертикальной границе

Главные краевые условия учитываются после полной сборки глобальной матрицы и учета 2-ых краевых условий следующим образом: в строке матрицы, соответствующие узлам расчетной области, которые лежат на границе с главным краевым условием, зануляются все внедиагональные элементы, на диагональ ставится единица, а в векторе b в соответствующем элементе значение заменяется на значение функции, которым задано главное краевое условие на данной границе, в узлах на этой границе. Отметим, что вторые краевые в нашей задаче присутствуют только на границах зон перфорации (в случае цилиндрических координат это левая граница расчетной области).

1. **Описание разработанных программ.**
   1. **funcForArea.cpp**

Содержит функции для работы с расчетной областью. Расчетная область считывается из файла и хранится в виде двух одномерных массивов rW и zW, которые хранят координатные линии, ограничивающие область, по r и z соответственно, и двумерный массив MW, который хранит номер границ подобластей расчетной области, на которых отличаются параметры задачи, и чему равны (по каким формулам считать) параметры задачи на подобласти.

Данные для разбиения расчетной области хранятся в файле, в котором сначала идут nrW – 1 пар для разбиения по r, потом nzW - 1 пар для разбиения по z, первое число в паре означает количество интервалов, на которое нужно разбить изначальный интервал, второе – коэффициент сжатия (растяжения) длины интервалов, на которые разбиваем. Сетка хранится в двух одномерных массивах r и z.

Параметры для каждой подобласти так же передаются файлом, со следующем структурой: первое число идет количество подобластей расчетной области, на которых параметры различаются, далее идет номер подобласти, для нее идет тензор структурной проницаемости в виде матрицы 2х2, потом идет m - количество фаз, далее m значений и m значений .

Для нахождения нужной подобласти используется сравнение по номерам узлов: составляются массивы Ir и Iz, которые содержат номера изначальных координатных линий расчетной области из массивов r и z соответственно. Для определения принадлежит ли конечный элемент подобласти, номера p, p + 1, s и s + 1 с номерами координатных линий подобласти из массивов Ir и Iz.

* 1. **funcForGlobalMatrix.cpp**

Содержит функции генерации глобальной матрицы. Сначала генерируется портрет матрицы путем составления для каждой базисной функции списка смежных с ней функций. Далее по этому списку строится портрет матрицы для дальнейшей генерации Глобальной матрицы.

Генерации глобальной матрицы и вектора правой части происходит путем расчета локальных матриц и вектора для каждого конечного элемента сетки по вышеуказанным формулам в разделе 2.2. Добавление в глобальную матрицу происходит в соответствии локальной нумерации глобальной с использованием заранее построенного портрета матрицы. В случае наличия фиктивных узлов, конечный элемент может не принадлежать расчетной области, поэтому тут же идет проверка на принадлежность, если проверку не прошло, программа находит, какие узлы на конечном элементе оказались фиктивными и ставит в строке соответствующего глобального номера узла на диагональ единицу (внедиагональные не зануляются, так как не пройдя проверку на принадлежность, в эти строки ничего не насчитается, поэтому вся строка уже будет нулевая), а вектор b ноль, чтобы соответствующий вес разложения после решения СЛАУ был равен нулю.

Далее идет функции учета краевых условий. Краевые условия передаются файлом, каждое краевое условие описывается 6 целыми числами:

* тип краевого условие (первое или второе);
* номер границы, или по-другому номер формул, задающих параметры краевого условия;
* номер элемента в массиве rW, с которого начинается краевое условие по r;
* номер элемента в массиве rW, которым заканчивается краевое условие по r;
* номер элемента в массиве zW, с которого начинается краевое условие по z;
* номер элемента в массиве zW, которым заканчивается краевое условие по z;

Сначала учитывается 2-ое краевое условие методом, описанным в разделе 2.3, проходя по всем узлам границы, на которой задана 2-ое краевое. После того как учлись 2-ые краевые условия, учитываются главные краевые условия методом, также описанным в разделе 2.3.

* 1. **LOS.cpp**

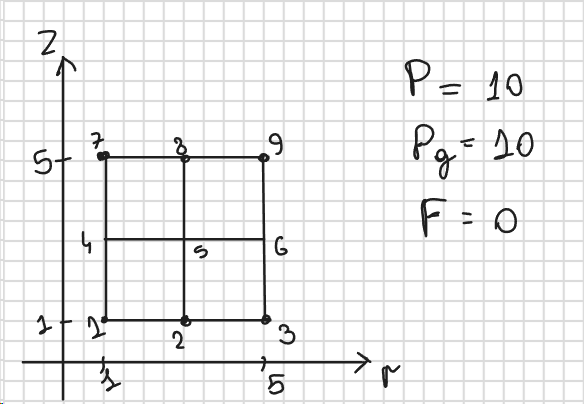
После построения глобальной матрицы и вектора правой части и учета всех краевых, решаем СЛАУ итерационным методом – локально-оптимальной схемой с факторизацией LU. Этот файл содержит все функции, необходимые для работы метода, основные – неполное LU разложение и реализация самого метода.

Для выхода из итерационного процесса при решении СЛАУ используется значение относительной невязки, которое должно стать меньше заданной точности. Также в случае аварийных ситуаций (например, метод не способен получить решение заданной точности) предусмотрен аварийный выход по количеству итераций, который задаем сами.

* 1. **mainProg.cpp**

В нем находится главное тело программы, в котором вызываются все нужные функции для ее работы. Также содержит функцию для расчета давления в любой точке расчетной области через полученные в результате решения СЛАУ веса разложения. Функция в сетке через дихотомию находит конечный элемент, которому принадлежит заданная точка, далее значения весов в соответствии с глобальными номерами узлов конечного элемента, значения границ, длины интервалов по r и z найденного конечного элемента и значения заданной точки подставляются в базисные функции , описанные в разделе 2.2, и находится приближенное значение давления в заданной точке.

1. **Тестирования программы.**
   1. **Тестирование первого краевого условия.**

Искомая функция , функция правой части , первое краевое условие на всех границах , тензор – единичная матрица, одна фаза и равны единице.



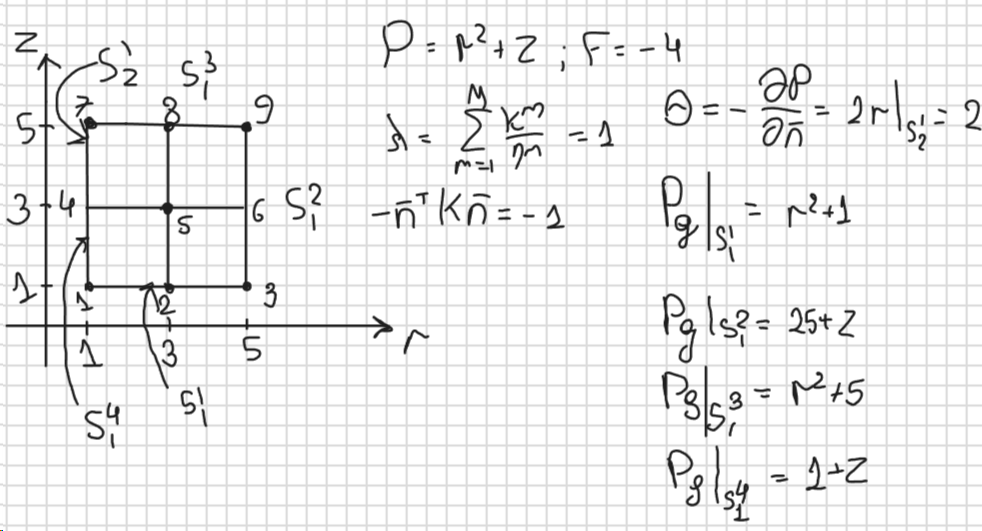
Ожидается вектор точного решения.

|  |  |  |
| --- | --- | --- |
| p\* | p | |p-p\*| |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |
| 10 | 10,000000000000 | 0,00E+00 |

p\* - вектор точного решения.

* 1. **Тестирование вторых краевых вместе с первыми краевыми.**

Искомая функция , функция правой части , первые краевые заданы функциями, перечисленными на рисунке, вторые краевые заданы функцией , тензор - единичная матрица, одна фаза и равны единице.



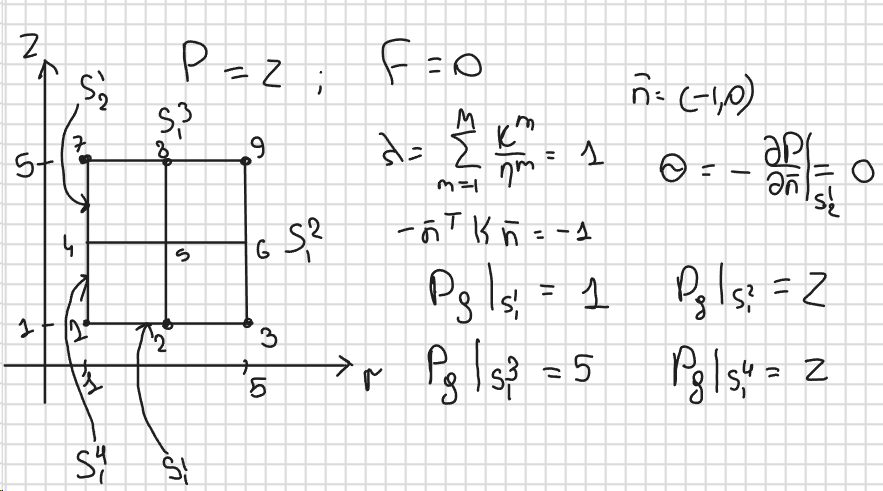
Ожидается вектор точного решения.

|  |  |  |
| --- | --- | --- |
| p\* | p | |p-p\*| |
| 2 | 2,0000000000000 | 0,00E+00 |
| 10 | 10,0000000000000 | 0,00E+00 |
| 26 | 26,0000000000000 | 0,00E+00 |
| 4 | 4,0000000000000 | 0,00E+00 |
| 12 | 12,0000000000000 | 0,00E+00 |
| 28 | 28,0000000000000 | 0,00E+00 |
| 6 | 6,0000000000000 | 0,00E+00 |
| 14 | 14,0000000000000 | 0,00E+00 |
| 30 | 30,0000000000000 | 0,00E+00 |

p\* - вектор точного решения.

1. **Исследования и выводы.**
   1. **Порядок аппроксимации.**

Искомая функция , функция правой части , первые краевые заданы функциями, перечисленными на рисунке, вторые краевые заданы функцией , тензор - единичная матрица, одна фаза и равны единице.

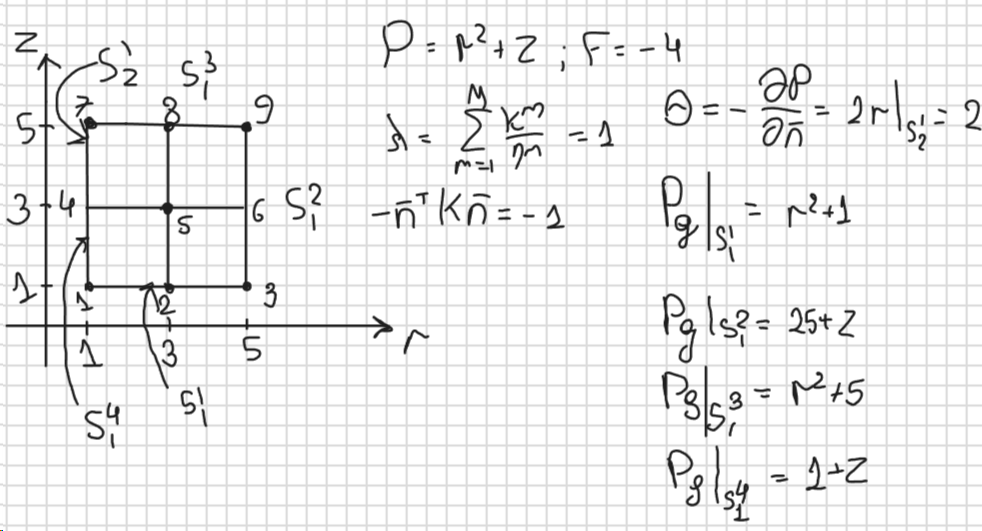




Ожидается нулевая погрешность в любых точка расчетной области.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | P(r, z) | P\*(r, z) | |P(r, z) - P\*(r, z)| |
| 1 | 1 | 1,00000000000000 | 1 | 0,00E+00 |
| 3 | 1 | 1,00000000000000 | 1 | 0,00E+00 |
| 5 | 1 | 1,00000000000000 | 1 | 0,00E+00 |
| 1 | 2 | 2,00000000000000 | 2 | 0,00E+00 |
| 3 | 2 | 2,00000000000000 | 2 | 0,00E+00 |
| 5 | 2 | 2,00000000000000 | 2 | 0,00E+00 |
| 1 | 3 | 3,00000000000000 | 3 | 0,00E+00 |
| 3 | 3 | 3,00000000000000 | 3 | 0,00E+00 |
| 5 | 3 | 3,00000000000000 | 3 | 0,00E+00 |
| 1 | 4 | 4,00000000000000 | 4 | 0,00E+00 |
| 3 | 4 | 4,00000000000000 | 4 | 0,00E+00 |
| 5 | 4 | 4,00000000000000 | 4 | 0,00E+00 |
| 1 | 5 | 5,00000000000000 | 5 | 0,00E+00 |
| 3 | 5 | 5,00000000000000 | 5 | 0,00E+00 |
| 5 | 5 | 5,00000000000000 | 5 | 0,00E+00 |

Погрешность во всех точках (даже на неузловых) равна нулю, как и ожидалось.

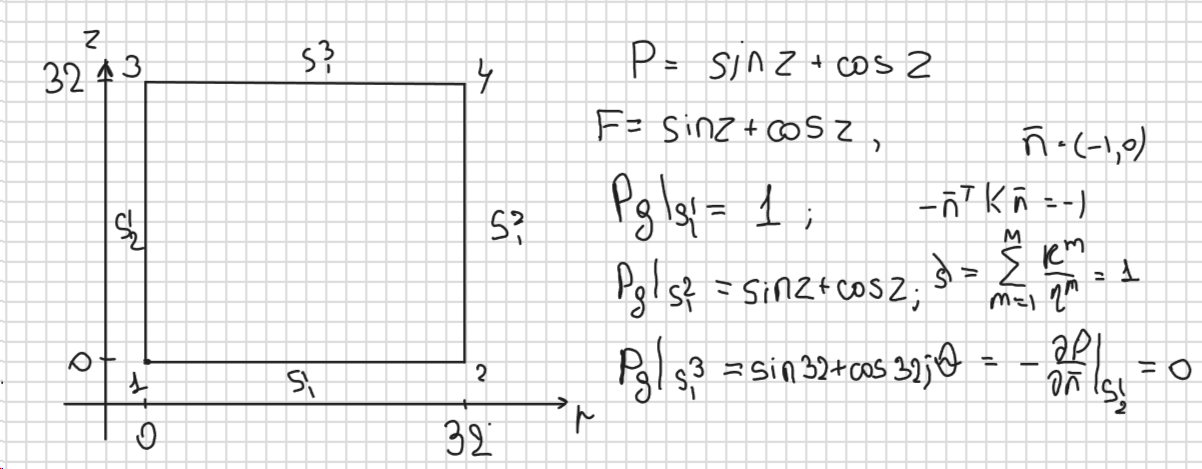
Искомая функция , функция правой части , первые краевые заданы функциями, перечисленными на рисунке, вторые краевые заданы функцией , тензор - единичная матрица, одна фаза и равны единице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | P(r, z) | P\*(r, z) | |P(r, z) - P\*(r, z)| |
| 1 | 1 | 2,00000000000000 | 2 | 0,00E+00 |
| 2 | 1 | 6,00000000000000 | 5 | 1,00E+00 |
| 3 | 1 | 10,0000000000000 | 10 | 0,00E+00 |
| 4 | 1 | 18,0000000000000 | 17 | 1,00E+00 |
| 5 | 1 | 26,0000000000000 | 26 | 0,00E+00 |
| 1 | 2 | 3,00000000000000 | 3 | 0,00E+00 |
| 2 | 2 | 7,00000000000000 | 6 | 1,00E+00 |
| 3 | 2 | 11,0000000000000 | 11 | 0,00E+00 |
| 4 | 2 | 19,0000000000000 | 18 | 1,00E+00 |
| 5 | 2 | 27,0000000000000 | 27 | 0,00E+00 |
| 1 | 3 | 4,00000000000000 | 4 | 0,00E+00 |
| 2 | 3 | 8,00000000000000 | 7 | 1,00E+00 |
| 3 | 3 | 12,0000000000000 | 12 | 0,00E+00 |
| 4 | 3 | 20,0000000000000 | 19 | 1,00E+00 |
| 5 | 3 | 28,0000000000000 | 28 | 0,00E+00 |
| 1 | 4 | 5,00000000000000 | 5 | 0,00E+00 |
| 2 | 4 | 9,00000000000000 | 8 | 1,00E+00 |
| 3 | 4 | 13,0000000000000 | 13 | 0,00E+00 |
| 4 | 4 | 21,0000000000000 | 20 | 1,00E+00 |
| 5 | 4 | 29,0000000000000 | 29 | 0,00E+00 |
| 1 | 5 | 6,00000000000000 | 6 | 0,00E+00 |
| 2 | 5 | 10,0000000000000 | 9 | 1,00E+00 |
| 3 | 5 | 14,0000000000000 | 14 | 0,00E+00 |
| 4 | 5 | 22,0000000000000 | 21 | 1,00E+00 |
| 5 | 5 | 30,0000000000000 | 30 | 0,00E+00 |

Ожидается погрешность в неузловых точках.

Погрешность в неузловых точках стала ненулевой, отсюда можно сделать вывод, что порядок аппроксимации равен порядку базисных функций, то есть в данной случае порядок аппроксимации равен единице.

* 1. **Порядок сходимости.**

Искомая функция , функция правой части , первые краевые заданы функциями, перечисленными на рисунке, вторые краевые заданы функцией , тензор - единичная матрица, одна фаза и равны единице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P\*(r, z) | P (r, z) | | |
|  | hr/z = 4 | hr/z = 2 | hr/z = 1 |
| r = 2, z = 0,5 | 1,35700810049458 | 0,35432407671342 | 0,90360597692278 | 1,17519962986454 |
| r = 2, z = 1 | 1,38177329067604 | -0,29135184657316 | 0,80721195384556 | 1,35039925972908 |
| r = 2, z = 1,5 | 1,06823218827176 | -0,93702776985974 | 0,71081793076835 | 0,94058898492372 |
| r = 2, z = 2 | 0,49315059027854 | -1,58270369314632 | 0,61442390769113 | 0,53077871011835 |
| |P\*(r, z)-P(r, z)| | - | 1,00E+00 | 4,53E-01 | 1,82E-01 |
| 1,67E+00 | 5,75E-01 | 3,14E-02 |
| 2,01E+00 | 3,57E-01 | 1,28E-01 |
| 2,08E+00 | 1,21E-01 | 3,76E-02 |
| Отношение  погрешностей 2h/h | - | 2,21E+00 | 2,49E+00 | - |
| 2,91E+00 | 1,83E+01 |
| 5,61E+00 | 2,80E+00 |
| 1,71E+01 | 3,22E+00 |

Отсюда видно, что при дроблении сетки погрешность уменьшалась примерно в 2 раза, так как разбиение сетки составляло в два раза, то получаем, что 2k = 2, где k – порядок сходимости, отсюда порядок сходимости равен единице. В некоторых местах погрешность уменьшается в какое-то другое количество раз, потому что вычисления происходят для тригонометрических функций, из-за чего с погрешностью аппроксимации искомой функции появляется еще и погрешность вычислений.

1. **Текст программы.**

**pch.h**

#ifndef PCH\_H

#define PCH\_H

#include <iostream>

#include <vector>

#include <set>

#include <functional>

#include <fstream>

#include <cmath>

#include "functions.h"

#endif

**pch.cpp**

#include "pch.h"

**funcForReading.cpp**

#include "pch.h"

void readingArea(estimatedArea &eA)

{

auto &rW = eA.rW, &zW = eA.zW;

auto &MW = eA.MW;

std::ifstream fin("estimatedArea.txt");

int nrW = 0, nzW = 0, L = 0;

fin >> nrW;

rW.resize(nrW);

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

fin >> rW[i];

fin >> nzW;

zW.resize(nzW);

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

fin >> zW[i];

fin >> L;

MW.resize(L);

for (int l = 0; l < L; l++)

{

MW[l].resize(5);

fin >> MW[l][0];

for (int i = 1; i < 5; i++)

{

int num = 0;

fin >> num;

MW[l][i] = num - 1;

}

}

fin.close();

}

void readingParamersOfAreas(vector <parametersOfAreas> &par)

{

std::ifstream fin("parametersOfAreas.txt");

int countAreas = 0;

fin >> countAreas;

par.resize(countAreas);

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

{

auto &tensor = par[i].tensor;

auto &kappa = par[i].kappa, &etta = par[i].etta;

auto &numArea = par[i].numArea;

int mCount = 0;

fin >> numArea;

tensor.resize(2);

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

{

tensor[ki].resize(2);

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

fin >> tensor[ki][kj];

}

fin >> mCount;

kappa.resize(mCount);

etta.resize(mCount);

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

fin >> kappa[m];

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

fin >> etta[m];

}

}

void readingBoundaryConditions(vector <vector <int>> &bC)

{

int countConditions = 0;

std::ifstream fin("boundaryConditions.txt");

fin >> countConditions;

bC.resize(countConditions);

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

{

bC[i].resize(6);

fin >> bC[i][0] >> bC[i][1];

int p0 = 0, p1 = 0, s0 = 0, s1 = 0;

fin >> p0 >> p1 >> s0 >> s1;

bC[i][2] = p0 - 1;

bC[i][3] = p1 - 1;

bC[i][4] = s0 - 1;

bC[i][5] = s1 - 1;

}

}

**funcForArea.cpp**

#include "pch.h"

void createGrid(estimatedArea &eA, grid &g, IArrays &I)

{

auto &r = g.r, &z = g.z, &rW = eA.rW, &zW = eA.zW;

auto &Ir = I.Ir, &Iz = I.Iz;

const int nR = rW.size() - 1, nZ = zW.size() - 1;

Ir.resize(nR + 1);

Iz.resize(nZ + 1);

std::ifstream fin("grid.txt");

int nRk = 0;

r.resize(1, rW[0]);

for (int i = 0, j = 1; i < nR; i++, j++)

{

int countInterval = 0;

double coef = 0, step = 0;

fin >> countInterval >> coef;

nRk += countInterval;

r.resize(nRk + 1);

if (coef != 1)

{

double sumProg = (pow(coef, countInterval) - 1.0) / (coef - 1.0);

step = (rW[i + 1] - rW[i]) / sumProg;

}

else

step = (rW[i + 1] - rW[i]) / countInterval;

int jk = 1;

for (j; j < nRk; j++, jk++)

if (coef != 1)

r[j] = rW[i] + step \* (pow(coef, jk) - 1.0) / (coef - 1.0);

else

r[j] = rW[i] + step \* jk;

r[j] = rW[i + 1];

Ir[i + 1] = j;

}

int nZk = 0;

z.resize(1, zW[0]);

for (int i = 0, j = 1; i < nZ; i++, j++)

{

int countInterval = 0;

double coef = 0, step = 0;

fin >> countInterval >> coef;

nZk += countInterval;

z.resize(nZk + 1);

if (coef != 1)

{

double sumProg = (pow(coef, countInterval) - 1.0) / (coef - 1.0);

step = (zW[i + 1] - zW[i]) / sumProg;

}

else

step = (zW[i + 1] - zW[i]) / countInterval;

int jk = 1;

for (j; j < nZk; j++, jk++)

if (coef != 1)

z[j] = zW[i] + step \* (pow(coef, jk) - 1.0) / (coef - 1.0);

else

z[j] = zW[i] + step \* jk;

z[j] = zW[i + 1];

Iz[i + 1] = j;

}

fin.close();

}

int numberOfEstimatedSubArea(IArrays &I, vector <vector <int>> &MW, int p, int s, int &l)

{

auto &Ir = I.Ir, &Iz = I.Iz;

const int L = MW.size(), L1 = l;

for (l; l < L; l++)

{

int mr0 = Ir[MW[l][1]], mr1 = Ir[MW[l][2]],

mz0 = Iz[MW[l][3]], mz1 = Iz[MW[l][4]], m = MW[l][0];

if (mr0 <= p && p <= mr1 && mr0 <= (p + 1) && (p + 1) <= mr1 &&

mz0 <= s && s <= mz1 && mz0 <= (s + 1) && (s + 1) <= mz1)

return m;

}

for (l = 0; l < L1; l++)

{

int mr0 = Ir[MW[l][1]], mr1 = Ir[MW[l][2]],

mz0 = Iz[MW[l][3]], mz1 = Iz[MW[l][4]], m = MW[l][0];

if (mr0 <= p && p <= mr1 && mr0 <= (p + 1) && (p + 1) <= mr1 &&

mz0 <= s && s <= mz1 && mz0 <= (s + 1) && (s + 1) <= mz1)

return m;

}

return -1;

}

bool IsFictiousNode(IArrays &I, vector <vector <int>> &MW, int p, int s, int &l)

{

auto &Ir = I.Ir, &Iz = I.Iz;

const int L = MW.size(), L1 = l;

for (l; l < L; l++)

{

int mr0 = Ir[MW[l][1]], mr1 = Ir[MW[l][2]],

mz0 = Iz[MW[l][3]], mz1 = Iz[MW[l][4]];

if (mr0 <= p && p <= mr1 && mz0 <= s && s <= mz1)

return false;

}

for (l = 0; l < L1; l++)

{

int mr0 = Ir[MW[l][1]], mr1 = Ir[MW[l][2]],

mz0 = Iz[MW[l][3]], mz1 = Iz[MW[l][4]];

if (mr0 <= p && p <= mr1 && mz0 <= s && s <= mz1)

return false;

}

return true;

}

**funcForGlobalMatrix.cpp**

#include "pch.h"

void generatePortrait(matrixProfile &mp, SLAE &slae, int nR, int nZ)

{

auto &ig = mp.ig, &jg = mp.jg;

const int n = nR \* nZ;

vector <std::set<int>> list{ }; // Список, который содержит глобальные номера j баз-ых ф-ий,

list.resize(n); // связанных с баз-ой ф-ей с глобальным номером i

slae.di.resize(n);

for (int s = 0; s < nZ - 1; s++)

for (int p = 0; p < nR - 1; p++)

{

vector <int> globalNum{ (s + 1) \* nR + p + 1, (s + 1) \* nR + p,

s \* nR + p + 1, s \* nR + p };

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

{

int ind1 = globalNum[i];

for (int j = i + 1; j < 4; j++)

{

int ind2 = globalNum[j];

list[ind1].insert(ind2);

}

}

}

// Теперь по списку строим портрет

ig.resize(n + 1);

ig[0] = 0;

ig[1] = 0;

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

ig[i + 1] = ig[i] + list[i].size();

int ign = ig[n];

jg.resize(ign);

slae.ggl.resize(ign);

slae.ggu.resize(ign);

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

for (auto j : list[i])

{

jg[k] = j;

k++;

}

}

void addLocalElement(matrixProfile &mp, SLAE &slae, double elem, int i, int j)

{

auto &di = slae.di, &ggl = slae.ggl, &ggu = slae.ggu;

auto &ig = mp.ig, &jg = mp.jg;

if (i == j)

di[i] += elem;

else

{

if (i > j)

{

int beg = ig[i], end = ig[i + 1] - 1;

while (jg[beg] != j)

{

int ind = (beg + end) / 2;

if (jg[ind] < j)

beg = ind + 1;

else

end = ind;

}

ggl[beg] += elem;

}

else

{

int beg = ig[j], end = ig[j + 1] - 1;

while (jg[beg] != i)

{

int ind = (beg + end) / 2;

if (jg[ind] < i)

beg = ind + 1;

else

end = ind;

}

ggu[beg] += elem;

}

}

}

double calcLambda(vector <double> &kappa, vector <double> &etta)

{

int mCount = kappa.size();

double lambda = 0;

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

lambda += kappa[m] / etta[m];

return lambda;

}

int mu(int i)

{

return i % 2;

}

int nu(int i)

{

return i / 2;

}

void boundaryConditions(IArrays &I, grid &g, matrixProfile &mp, SLAE &slae, vector <vector <int>> &bC, functionsBC &func)

{

auto &ig = mp.ig, &jg = mp.jg, &Ir = I.Ir, &Iz = I.Iz;

auto &r = g.r, &z = g.z, &di = slae.di, &ggl = slae.ggl, &ggu = slae.ggu, &b = slae.b;

auto &firstBC = func.firstBC, &secondBC = func.secondBC;

const int countCond = bC.size(), nR = r.size(), nZ = z.size(), n = ig[di.size()];

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

{

int typeCond = bC[k][0];

if (typeCond == 2)

{

int numFunc = bC[k][1] - 1, p0 = bC[k][2], p1 = bC[k][3], s0 = bC[k][4], s1 = bC[k][5];

if (p0 == p1)

{

int p = Ir[p0], s = Iz[s0];

s1 = Iz[s1];

for (s; s < s1; s++)

{

double tetta1 = secondBC[numFunc](r[p], z[s]),

tetta2 = secondBC[numFunc](r[p], z[s + 1]), coef = (z[s + 1] - z[s]) / 6.0;

int globalNum1 = nR \* s + p, globalNum2 = nR \* (s + 1) + p;

b[globalNum1] += coef \* (2 \* tetta1 + tetta2);

b[globalNum2] += coef \* (tetta1 + 2 \* tetta2);

}

}

if (s0 == s1)

{

int s = Iz[s0], p = Ir[p0];

p1 = Ir[p1];

for (p; p < p1; p++)

{

double tetta1 = secondBC[numFunc](r[p], z[s]),

tetta2 = secondBC[numFunc](r[p + 1], z[s]), hr = r[p + 1] - r[p],

coef1 = hr \* r[p] / 6.0, coef2 = hr \* hr / 12.0;

int globalNum1 = nR \* s + p, globalNum2 = nR \* s + p + 1;

b[globalNum1] += coef1 \* (2 \* tetta1 + tetta2) + coef2 \* (tetta1 + tetta2);

b[globalNum2] += coef1 \* (tetta1 + tetta2) + coef2 \* (tetta1 + 3 \* tetta2);

}

}

}

}

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

{

int typeCond = bC[k][0];

if (typeCond == 1)

{

int numFunc = bC[k][1] - 1, p0 = bC[k][2], p1 = bC[k][3], s0 = bC[k][4], s1 = bC[k][5];

if (p0 == p1)

{

int p = Ir[p0], s = Iz[s0];

s1 = Iz[s1];

for (s; s <= s1; s++)

{

int globalNum = nR \* s + p, i0 = ig[globalNum], i1 = ig[globalNum + 1];

for (i0; i0 < i1; i0++)

ggl[i0] = 0;

int j0 = ig[globalNum + 1];

for (j0; j0 < n; j0++)

if (jg[j0] == globalNum) ggu[j0] = 0;

di[globalNum] = 1;

b[globalNum] = firstBC[numFunc](r[p], z[s]);

}

}

if (s0 == s1)

{

int s = Iz[s0], p = Ir[p0];

p1 = Ir[p1];

for (p; p <= p1; p++)

{

int globalNum = nR \* s + p, i0 = ig[globalNum], i1 = ig[globalNum + 1];

for (i0; i0 < i1; i0++)

ggl[i0] = 0;

int j0 = ig[globalNum + 1];

for (j0; j0 < n; j0++)

if (jg[j0] == globalNum) ggu[j0] = 0;

di[globalNum] = 1;

b[globalNum] = firstBC[numFunc](r[p], z[s]);

}

}

}

}

}

void calcGlobalMatrixAndVector(matrixProfile &mp, SLAE &slae, grid &g, IArrays &I, estimatedArea &eA, vector <parametersOfAreas> &par, matrices &M)

{

auto &r = g.r, &z = g.z, &b = slae.b;

auto &G1 = M.G1, &M1 = M.M1, &M2 = M.M2, &H1 = M.H1, &H2 = M.H2;

auto &Ir = I.Ir, &Iz = I.Iz;

auto &MW = eA.MW;

auto &F = slae.F;

int nR = r.size(), nZ = z.size();

const int n = nR \* nZ;

b.resize(n);

int l = 0, countAreas = par.size();

for (int s = 0; s < nZ - 1; s++)

{

for (int p = 0; p < nR - 1; p++)

{

int numArea = numberOfEstimatedSubArea(I, MW, p, s, l);

vector <int> globalNumbers = { nR \* s + p, nR \* s + p + 1, nR \* (s + 1) + p, nR \* (s + 1) + p + 1 };

if (numArea != -1)

{

double hz = z[s + 1] - z[s], hr = r[p + 1] - r[p], rp = r[p], coefGr = (2.0 \* rp + hr) / (2.0 \* hr),

coefGz = 1.0 / hz, coefMr1 = hr \* rp / 6.0, coefMr2 = hr \* hr / 12.0,

coefMz = hz / 6.0, coefHr1 = rp, coefHr2 = hr / 6.0;

int k = 0;

for (k = 0; numArea != par[k].numArea; k++);

auto &kappa = par[k].kappa, &etta = par[k].etta;

auto &tensor = par[k].tensor;

double lambda = calcLambda(kappa, etta);

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

{

vector <double> f = { F[k](r[p], z[s]), F[k](r[p + 1], z[s]),

F[k](r[p], z[s + 1]), F[k](r[p + 1], z[s + 1]) };

double sumbi = 0;

int mui = mu(i), nui = nu(i);

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

{

int muj = mu(j), nuj = nu(j);

double elemGij =

lambda \* (tensor[0][0] \* coefGr \* G1[mui][muj] \* coefMz \* M1[nui][nuj] +

tensor[0][1] \* (coefHr1 \* H1[mui][muj] + coefHr2 \* H2[mui][muj]) \* H1[nuj][nui] +

tensor[1][0] \* (coefHr1 \* H1[muj][mui] + coefHr2 \* H2[muj][mui]) \* H1[nui][nuj] +

tensor[1][1] \* (coefMr1 \* M1[mui][muj] + coefMr2 \* M2[mui][muj]) \* coefGz \* G1[nui][nuj]);

addLocalElement(mp, slae, elemGij, globalNumbers[i], globalNumbers[j]);

sumbi += f[j] \* (coefMr1 \* M1[mui][muj] + coefMr2 \* M2[mui][muj]) \* coefMz \* M1[nui][nuj];

}

b[globalNumbers[i]] += sumbi;

}

}

else

{

auto &di = slae.di;

vector <vector<int>> localNum = { { p, s }, { p + 1, s }, { p, s + 1 }, { p + 1, s + 1 } };

int l1 = 0;

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

if (IsFictiousNode(I, MW, localNum[i][0], localNum[i][1], l1))

di[globalNumbers[i]] = 1;

}

}

}

}

**LOS.cpp**

#include "pch.h"

void calcLU(matrixProfile &mp, SLAE &A, SLAE &LU)

{

auto &ig = mp.ig, &jg = mp.jg;

auto &ggl = A.ggl, &ggu = A.ggu, &di = A.di, &L = LU.ggl, &U = LU.ggu, &diL = LU.di;

LU.b = A.b;

const int n = di.size();

diL.resize(n);

const int sizeMat = ig[n];

L.resize(sizeMat);

U.resize(sizeMat);

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

{

double sumDi = 0;

int i0 = ig[i];

int i1 = ig[i + 1];

for (int k = i0; k < i1; k++)

{

double suml = 0, sumu = 0;

int j = jg[k];

int j0 = ig[j];

int j1 = ig[j + 1];

for (int ik = i0, kj = j0; ik < i1 && kj < j1; )

{

if (jg[ik] > jg[kj]) kj++;

else if (jg[ik] < jg[kj]) ik++;

else

{

suml += L[ik] \* U[kj];

sumu += L[kj] \* U[ik];

ik++;

kj++;

}

}

L[k] = (ggl[k] - suml);

U[k] = (ggu[k] - sumu) / diL[j];

sumDi += L[k] \* U[k];

}

diL[i] = di[i] - sumDi;

}

}

void localOptimalSchemeLU(matrixProfile &mp, SLAE &A, SLAE &LU, LOS &v, vector <double> &p, int maxIter, double eps)

{

auto &r1 = v.r1, &z1 = v.z1, &p1 = v.p1, &mult = v.mult, &rk = v.rk, &Ar = v.Ar, &q = v.q;

const int n = A.di.size();

double normb = 0;

p.resize(n);

r1.resize(n);

z1.resize(n);

p1.resize(n);

mult.resize(n);

rk.resize(n);

Ar.resize(n);

q.resize(n);

calcDiscrepancy(mp, A, v, p, normb);

calcY(mp, LU, r1, r1);

calcX(mp, LU, r1, z1);

multOfMatrix(mp, A, z1, p1);

calcY(mp, LU, p1, p1);

double scalarr = scalarMult(r1, r1),

discrepancy = sqrt(scalarr / normb);

for (int k = 1; k < maxIter && discrepancy > eps; k++)

{

double scalarp = scalarMult(p1, p1),

alpha = scalarMult(p1, r1) / scalarp;

calcVectorMultCoef(z1, alpha, mult);

calcSumVectors(p, v.mult, p);

calcVectorMultCoef(p1, -alpha, mult);

calcSumVectors(r1, mult, r1);

calcX(mp, LU, r1, rk);

multOfMatrix(mp, A, rk, Ar);

calcY(mp, LU, Ar, q);

double betta = -scalarMult(p1, q) / scalarp;

calcVectorMultCoef(z1, betta, mult);

calcSumVectors(rk, mult, z1);

calcVectorMultCoef(p1, betta, mult);

calcSumVectors(q, mult, p1);

discrepancy = sqrt(scalarMult(r1, r1) / scalarr);

std::cout << k << " " << discrepancy << std::endl;

}

normb = 0;

calcDiscrepancy(mp, A, v, p, normb);

discrepancy = sqrt(scalarMult(r1, r1) / normb);

std::cout << "Final discrepancy: " << discrepancy << std::endl;

}

void multOfMatrix(matrixProfile &mp, SLAE &A, vector <double> &x, vector <double> &F)

{

auto &ig = mp.ig, &jg = mp.jg;

auto &di = A.di, &ggl = A.ggl, &ggu = A.ggu;

const int n = di.size();

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

{

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

int i0 = ig[i], i1 = ig[i + 1];

for (i0; i0 < i1; i0++)

{

int j = jg[i0];

F[i] += ggl[i0] \* x[j];

F[j] += ggu[i0] \* x[i];

}

}

}

void calcY(matrixProfile &mp, SLAE &LU, vector <double> &b, vector <double> &y)

{

auto &ig = mp.ig, &jg = mp.jg;

auto &di = LU.di, &L = LU.ggl;

const int n = di.size();

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

{

double sum = 0;

int i0 = ig[i], i1 = ig[i + 1];

for (i0; i0 < i1; i0++)

{

int j = jg[i0];

sum += L[i0] \* y[j];

}

y[i] = (b[i] - sum) / di[i];

}

}

void calcX(matrixProfile &mp, SLAE &LU, vector <double> &y, vector <double> &x)

{

auto &ig = mp.ig, &jg = mp.jg;

auto &U = LU.ggu;

const int n = ig.size() - 1;

vector <double> v = y;

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

{

x[i] = v[i];

int i0 = ig[i], i1 = ig[i + 1];

for (i0; i0 < i1; i0++)

{

int j = mp.jg[i0];

v[j] -= x[i] \* U[i0];

}

}

}

void calcDiscrepancy(matrixProfile &mp, SLAE &A, LOS &v, vector <double> &x, double &normb)

{

auto &ig = mp.ig, &jg = mp.jg;

auto &ggl = A.ggl, &ggu = A.ggu, &di = A.di, &b = A.b, &r1 = v.r1;

const int n = di.size();

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

{

normb += b[i] \* b[i];

r1[i] = b[i] - di[i] \* x[i];

int i0 = ig[i], i1 = ig[i + 1];

for (i0; i0 < i1; i0++)

{

int j = mp.jg[i0];

r1[i] -= ggl[i0] \* x[j];

r1[j] -= ggu[i0] \* x[i];

}

}

}

void calcVectorMultCoef(vector <double> &a, double coef, vector <double> &res)

{

const int n = a.size();

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

res[i] = a[i] \* coef;

}

void calcSumVectors(vector <double> &a, vector <double> &b, vector <double> &res)

{

const int n = a.size();

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

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

}

double scalarMult(vector <double> &a, vector <double> &b)

{

int n = a.size();

double res = 0;

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

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

return res;

}

**functions.h**

#include "pch.h"

using std::vector;

struct estimatedArea

{

vector <double> rW{ }, zW{ };

vector <vector <int>> MW{ };

};

struct parametersOfAreas

{

int numArea = 0;

vector <vector <double>> tensor{ };

vector <double> kappa{ }, etta{ };

};

struct grid

{

vector <double> r{ }, z{ };

};

struct matrices

{

vector <vector <double>> G1 = { { 1, -1 }, { -1, 1 } },

M1 = { { 2, 1 }, { 1, 2 } },

M2 = { { 1, 1 }, { 1, 3 } },

H1 = { { -0.5, 0.5 }, { -0.5, 0.5 } },

H2 = { { -1, 1 }, { -2, 2 } };

};

struct matrixProfile

{

vector <int> ig{ }, jg{ };

};

struct SLAE

{

vector <double> di{ }, ggl{ }, ggu{ }, b{ };

vector <std::function<double(double, double)>> F

{

[](double r, double z) { return sin(z) + cos(z); },

[](double r, double z) { return 0; },

[](double r, double z) { return 0; }

};

};

struct functionsBC

{

vector <std::function<double(double, double)>> firstBC

{

[](double r, double z) { return 1; },

[](double r, double z) { return cos(z) + sin(z); },

[](double r, double z) { return cos(32) + sin(32); },

[](double r, double z) { return 1 + z; }

};

vector <std::function<double(double, double)>> secondBC

{

[](double r, double z) { return 0; },

[](double r, double z) { return 7 + z; },

[](double r, double z) { return r + 5; },

[](double r, double z) { return 1 + z; }

};

};

struct LOS

{

vector <double> r1{ }, rk{ }, z1{ }, p1{ }, Ar{ }, q{ }, mult{ };

};

struct IArrays

{

vector <int> Ir{ }, Iz{ };

};

void readingArea(estimatedArea &eA);

void readingParamersOfAreas(vector <parametersOfAreas> &par);

void readingBoundaryConditions(vector <vector <int>> &bC);

void createGrid(estimatedArea &eA, grid &g, IArrays &I);

int numberOfEstimatedSubArea(IArrays &I, vector <vector <int>> &MW, int p, int s, int &l);

bool IsFictiousNode(IArrays &I, vector <vector <int>> &MW, int p, int s, int &l);

void generatePortrait(matrixProfile &mp, SLAE &slae, int nR, int nZ);

void addLocalElement(matrixProfile &mp, SLAE &slae, double elem, int i, int j);

double calcLambda(vector <double> &kappa, vector <double> &etta);

int mu(int i);

int nu(int i);

void boundaryConditions(IArrays &I, grid &g, matrixProfile &mp, SLAE &slae, vector <vector <int>> &bC, functionsBC &func);

void calcGlobalMatrixAndVector(matrixProfile &mp, SLAE &slae, grid &g, IArrays &I, estimatedArea &eA, vector <parametersOfAreas> &par, matrices &M);

void calcLU(matrixProfile &mp, SLAE &A, SLAE &LU);

void localOptimalSchemeLU(matrixProfile &mp, SLAE &A, SLAE &LU, LOS &v, vector <double> &p, int maxIter, double eps);

void multOfMatrix(matrixProfile &mp, SLAE &A, vector <double> &x, vector <double> &F);

void calcY(matrixProfile &mp, SLAE &LU, vector <double> &y, vector <double> &b);

void calcX(matrixProfile &mp, SLAE &LU, vector <double> &y, vector <double> &x);

void calcDiscrepancy(matrixProfile &mp, SLAE &A, LOS &v, vector <double> &x, double &normb);

void calcVectorMultCoef(vector <double> &a, double coef, vector <double> &res);

void calcSumVectors(vector <double> &a, vector <double> &b, vector <double> &res);

double scalarMult(vector <double> &a, vector <double> &b);

double valueFuncAtPoint(grid &g, double r, double z, vector <double> &p);

void outputVector(vector <double> &p);

**mainProg.cpp**

#include "pch.h"

using std::vector;

int main()

{

estimatedArea eA{ };

vector <parametersOfAreas> par{ };

grid g{ };

IArrays I{ };

matrices M{ };

vector <vector <int>> bC{ };

functionsBC func{ };

vector <double> p{ };

SLAE slae{ }, LU{ };

matrixProfile mp{ };

LOS v{ };

readingArea(eA);

readingParamersOfAreas(par);

createGrid(eA, g, I);

generatePortrait(mp, slae, g.r.size(), g.z.size());

calcGlobalMatrixAndVector(mp, slae, g, I, eA, par, M);

readingBoundaryConditions(bC);

boundaryConditions(I, g, mp, slae, bC, func);

calcLU(mp, slae, LU);

localOptimalSchemeLU(mp, slae, LU, v, p, 10000, 1e-15);

double r = 1.1, z = 1.1;

//double valueFunc = valueFuncAtPoint(g, r, z, p);

//std::cout << valueFunc << std::endl;

outputVector(p);

//printf\_s("\n%.15lf\n", valueFunc);

printf\_s("\n");

//for (double z = 0; z <= 2; z += 1)

for (double z = 0.5, r = 2; z <= 2.; z += 0.5)

{

double valueFunc = valueFuncAtPoint(g, r, z, p);

printf\_s("%.15lf\n", valueFunc);

}

return 0;

}

double valueFuncAtPoint(grid &g, double r, double z, vector <double> &p)

{

auto &gR = g.r, &gZ = g.z;

const int nR = gR.size(), nZ = gZ.size();

if (r < gR[0] || r > gR[nR - 1] || z < gZ[0] || z > gZ[nZ - 1])

{

std::cout << "The point does not belong to the area." << std::endl;

return -1;

}

int begR = 0, endR = nR - 1, begZ = 0, endZ = nZ - 1;

while (!(gR[begR] <= r && r <= gR[begR + 1]))

{

int indR = (begR + endR) / 2;

if (gR[indR] < r)

begR = indR;

else

endR = indR;

}

while (!(gZ[begZ] <= z && z <= gZ[begZ + 1]))

{

int indZ = (begZ + endZ) / 2;

if (gZ[indZ] < z)

begZ = indZ;

else

endZ = indZ;

}

vector <int> globalNumbers = { begZ \* nR + begR, begZ \* nR + begR + 1, (begZ + 1) \* nR + begR, (begZ + 1) \* nR + begR + 1 };

double r0 = gR[begR], r1 = gR[begR + 1], z0 = gZ[begZ], z1 = gZ[begZ + 1], hr = r1 - r0, hz = z1 - z0;

double valueFuncP = p[globalNumbers[0]] \* (r1 - r) / hr \* (z1 - z) / hz +

p[globalNumbers[1]] \* (r - r0) / hr \* (z1 - z) / hz +

p[globalNumbers[2]] \* (r1 - r) / hr \* (z - z0) / hz +

p[globalNumbers[3]] \* (r - r0) / hr \* (z - z0) / hz;

return valueFuncP;

}

void outputVector(vector <double> &p)

{

const int n = p.size();

printf\_s("p:\n");

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

printf\_s("%.15lf\n", p[i]);

}