|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Курсовой проект | | |
| по дисциплине «Численные методы» | | |
| **0** | | |
|  | | |
|  | Группа | ПМ-95 |
| Студент | Левковский кирилл |
|  |
| Преподаватель | патрушев и.и. |
|  |  |
| Дата | 28.02.2022 |
|  |  |
| Новосибирск | | |

1. **Задание**

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции квадратичные на треугольниках. Коэффициент диффузии разложить по линейным базисным функциям. Матрицу СЛАУ генерировать в разложенном строчном формате. Для решения СЛАУ использовать МСГ и ЛОС с неполной факторизацией.

1. Постановка задачи

Эллиптическая краевая задача для функции 𝒖 определяется дифференциальным уравнением:

−𝑑𝑖𝑣(𝜆𝑔𝑟𝑎𝑑𝑢) + 𝛾𝑢 = 𝑓

Заданным в некоторой области 𝛀 с границей 𝑺 = 𝑺𝟏 ∪ 𝑺𝟐 ∪ 𝑺𝟑, и краевыми условиями:

𝑢|𝑠1 = 𝑢𝑔

𝜆 | 𝑠2 = 𝜃

𝜆 | 𝑠3 + 𝛽(𝑢|𝑠3 − 𝑢𝛽) = 0

где 𝑢|𝑆1− значение искомой функции u на границе 𝑆1, |𝑆𝑖− значение производной функции u по направлению внешней нормали к поверхности на границе 𝑆𝑖,𝑖=2,3.

− (𝜆 ) − (𝜆 ) + 𝛾𝑢 = 𝑓

1. **Теоретическая часть**
   1. **Вариационная постановка в форме уравнения Галеркина**

Воспользуемся вариационной постановкой в форме уравнения Галеркина. Введём гильбертово пространство ℍ = 𝑳𝟐(𝛀). Исходное уравнения можно записать в операторной форме 𝐿𝑢=𝑓.

Для уравнения постановка примет вид:

∫Ω-div(𝜆𝑔𝑟𝑎𝑑𝑢) 𝜈𝑑Ω + ∫Ω (𝛾𝑢 − 𝑓)𝜈𝑑Ω = 0, ∀𝜈 ∈ ℍ0

Далее применяя формулу Грина получим:

∫Ω 𝜆𝑔𝑟𝑎𝑑𝑢 ∗ 𝑔𝑟𝑎𝑑𝜈𝑑Ω − ∫𝑆 𝜆 𝜈𝑑𝑆 + ∫Ω (𝛾𝑢 − 𝑓)𝜈𝑑Ω = 0

где 𝑺 = 𝑺𝟏 ∪ 𝑺𝟐 ∪ 𝑺𝟑.

Так как 𝑢|𝑠1 = 0, следовательно ∫ 𝑆1𝜆 𝜈𝑑𝑆 =0, то можно переписать интегральное соотношение:

∫Ω 𝜆𝑔𝑟𝑎𝑑𝑢 ∗ 𝑔𝑟𝑎𝑑𝜈𝑑Ω − ∫𝑆2𝜃𝜈𝑑𝑆 − ∫𝑆3 𝛽(𝑢 − 𝑢𝛽)𝜈𝑑𝑆 + ∫Ω (𝛾𝑢 − 𝑓)𝜈𝑑Ω = 0

В пространстве ℍ01 выделим конечномерное подпространство 𝑽 𝒉 , которое определяется как линейное пространство, натянутое на базисные функции 𝝍𝒊 ,𝒊 =. Заменим функцию 𝒖 аппроксимирующей ее функцией 𝒖 𝒉 , а функцию 𝝂𝟎 - функцией 𝝂𝟎 𝒉 , получим аппроксимацию уравнения Галеркина:

∫Ω 𝜆𝑔𝑟𝑎𝑑𝑢 ℎ ∗ 𝑔𝑟𝑎𝑑𝜈0 ℎ𝑑Ω + ∫Ω 𝛾𝑢 ℎ𝜈0 ℎ𝑑Ω + ∫𝑆3𝛽𝑢 ℎ𝜈0 ℎ𝑑𝑆 = ∫Ω 𝑓𝜈0 ℎ𝑑Ω + ∫𝑆2 𝜃𝜈0 ℎ𝑑𝑆

+ ∫𝑆3𝛽𝑢𝛽𝜈0 ℎ𝑑𝑆

Так как 𝝂𝟎 𝒉 представима в виде линейной комбинации:

𝜈0 ℎ =

Приводим уравнение к следующей системе уравнений:

∫Ω 𝜆𝑔𝑟𝑎𝑑𝑢 ℎ ∗ 𝑔𝑟𝑎𝑑𝜓𝑖𝑑Ω + ∫Ω 𝛾𝑢 ℎ𝜓𝑖𝑑Ω + ∫𝑆3𝛽𝑢 ℎ𝜓𝑖𝑑𝑆 = ∫Ω 𝑓𝜓𝑖𝑑Ω + ∫𝑆2 𝜃𝜓𝑖𝑑𝑆

+ ∫𝑆3𝛽𝑢𝛽𝜓𝑖𝑑𝑆 , 𝑖 ∈ 𝑁0

Так же 𝒖 𝒉 представим в виде:

𝑢 ℎ =

Причём 𝒏 − 𝑵𝟎 компонент вектора весов 𝒒 = (𝒒𝟏, … , 𝒒𝒏 ) 𝑻 могут быть фиксированы и определены из первого краевого условия 𝒖𝒉 | 𝒔𝟏 = 𝒖𝒈.

Подставим разложение (15) в систему (14) и получим СЛАУ для компонент 𝒒𝒋 вектора весов 𝒒:

 (∫Ω 𝜆𝑔𝑟𝑎𝑑𝜓𝑗 ∗ 𝑔𝑟𝑎𝑑𝜓𝑖𝑑Ω + ∫Ω 𝛾𝜓𝑗𝜓𝑖𝑑Ω + ∫𝑆3𝛽𝜓𝑗𝜓𝑖𝑑𝑆 ) 𝑞𝑗 = ∫Ω 𝑓𝜓𝑖𝑑Ω + ∫𝑆2𝜃𝜓𝑖𝑑𝑆

+ ∫𝑆3𝛽𝑢𝛽𝜓𝑖𝑑𝑆 ,𝑖 ∈ 𝑁0 (16)

Таким образом СЛАУ может записана в виде 𝑨𝒒 = 𝒃, где:

𝐴𝑖𝑗 =

𝑏𝑖 =

* 1. **Построение базисных функций**

Рассмотрим треугольник Ωm с вершинами (x1 ,y1), (x2 ,y2), (x3 ,y3). Введем на нем три линейные функции

такие, что  равна единице в вершине  и нулю во всех остальных вершинах,  равна единице в вершине  и нулю во всех остальных вершинах,  равна единице в вершине  и нулю во всех остальных вершинах.

При построении квадратичный базисный функций, узлами сетки являются не только вершины треугольников, но и точки на их ребрах. Базисные функции квадратичного элемента удобно выразить через L:

;

;

.

Учитывая построение - функций, получаем следующие соотношения:



Т.е. имеем систему:



Отсюда находим коэффициенты линейных функций 





При вычислении интегралов от произведений вида  по треугольнику или любому его ребру  можно использовать следующие соотношения:



где - это площадь треугольника,  - матрица координат его вершин.

* 1. **Построение локальных матриц**

Чтобы получить выражения для локальных матриц жёсткости G и массы M каждого конечного элемента , перейдём к решению локальной задачи на каждом конечном элементе. Полученное уравнение для области  представим в виде суммы интегралов по областям  без учёта краевых условий. Тогда на каждом конечном элементе будем решать локальную задачу построения матриц жёсткости и массы и вектора правой части.



Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 3x3 (по числу узлов на конечном элементе).

**3.3.1 Построение матрицы жесткости**

Рассмотрим первый член в выражении для k-го конечного элемента:



Так как подынтегральные выражения для матрицы жесткости содержат производные базисных функций покажем, как можно получить эти производные через L-координаты. Частные производные для L

Тогда частные производные для (\*) вычисляются так:

В поставленной задаче требуется разложить λ по линейным базисным функциям:

,

где – значения коэффициента диффузии λ в соответствующих узлах, а  - линейные базисные функции, которые определяются следующим образом:

таким образом:

* + 1. **Построение правой части**

Рассмотрим правую часть выражения для k-го конечного элемента:

 представим в виде, где - значения в узлах. Получим:

Таким образом, ,.

* 1. **Сборка глобальной матрицы и вектора правой части**

При формировании глобальной матрицы из локальных, полученных суммированием соответствующих матриц массы и жесткости, учитываем соответствие локальной и глобальной нумераций каждого конечного элемента. Глобальная нумерация каждого конечного элемента однозначно определяет позиции вклада его локальной матрицы в глобальную. Поэтому, зная глобальные номера соответствующих узлов конечного элемента, определяем и то, какие элементы глобальной матрицы изменятся при учете текущего конечного элемента. Аналогичным образом определяется вклад локального вектора правой части в глобальный. При учете текущего локального вектора изменятся те элементы глобального вектора правой части, номера которых совпадают с глобальными номерами узлов, присутствующих в этом конечном элементе.

* 1. **Краевые условия первого рода**

Краевые условия первого рода будем учитывать с поправкой на то, что на границе, где указаны краевые условия функции ψ равные нулю. Реализовать данные условия можно путем внесения в глобальную матрицу на главную диагональ в соответствующий номер этой диагонали (номер узла в глобальной нумерации, для которого в данный момент учитывается краевое условие первого рода) ставится число , то есть число намного большее, чем элементы глобальной матрицы. В вектор правой части ставится число , которое соответствует произведению точного значения функции в данном узле на относительно большое число, что приводит нас к решению обычного линейного уравнения, решением которого является значение функции в данном узле.

1. **Тесты**

3

4

10

8

9

7

11

1

2

5

6

0

|  |  |
| --- | --- |
| Координаты узлов | Конечные элементы |
| 2 0  2 1  3 1  2 4  7 4  2 0.5  2.5 0.5  2.5 1  2 2.5  2.5 2.5  4.5 4  5 2.5 | 0: 0 1 2 5 7 6  1: 2 1 3 7 8 9  2: 2 3 4 9 10 11 |

Тест для общей проверки программы на константе с краевыми условиями 1ого рода.

|  |  |  |
| --- | --- | --- |
| Полученное решение | Точное решение | Погрешность |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |
| 1337.000000 | 1337.000000 | 0.000000 |

Тест для общей проверки программы на полиномах первой степени с краевыми условиями 1ого рода. U = x + y;

|  |  |  |
| --- | --- | --- |
| Полученное решение | Точное решение | Погрешность |
| 2.0000000000 | 2.0000000000 | 0.0000000000 |
| 3.0000000000 | 3.0000000000 | 0.0000000000 |
| 4.0000000000 | 4.0000000000 | 0.0000000000 |
| 6.0000000000 | 6.0000000000 | 0.0000000000 |
| 11.0000000000 | 11.0000000000 | 0.0000000000 |
| 2.5000000000 | 2.5000000000 | 0.0000000000 |
| 3.0000000000 | 3.0000000000 | 0.0000000000 |
| 3.5000000000 | 3.5000000000 | 0.0000000000 |
| 4.5000000000 | 4.5000000000 | 0.0000000000 |
| 5.0000000000 | 5.0000000000 | 0.0000000000 |
| 8.5000000000 | 8.5000000000 | 0.0000000000 |
| 7.5000000000 | 7.5000000000 | 0.0000000000 |

Тест для общей проверки программы на полиномах второй степени с краевыми условиями 1ого рода. U = ;

|  |  |  |
| --- | --- | --- |
| Полученное решение | Точное решение | Погрешность |
| 4.0000000000 | 4.0000000000 | 0.0000000000 |
| 5.0000000000 | 5.0000000000 | 0.0000000000 |
| 10.0000000000 | 10.0000000000 | 0.0000000000 |
| 20.0000000000 | 20.0000000000 | 0.0000000000 |
| 65.0000000000 | 65.0000000000 | 0.0000000000 |
| 4.2500000000 | 4.2500000000 | 0.0000000000 |
| 6.5000000000 | 6.5000000000 | 0.0000000000 |
| 7.1985294118 | 7.2500000000 | 0.0514705882 |
| 10.2500000000 | 10.2500000000 | 0.0000000000 |
| 12.9889705882 | 12.5000000000 | 0.4889705882 |
| 36.2500000000 | 36.2500000000 | 0.0000000000 |
| 31.2500000000 | 31.2500000000 | 0.0000000000 |

Тест для общей проверки программы на полиномах третьей степени с краевыми условиями 1ого рода. U = ;

|  |  |  |
| --- | --- | --- |
| Полученное решение | Точное решение | Погрешность |
| 8.0000000000 | 8.0000000000 | 0.0000000000 |
| 9.0000000000 | 9.0000000000 | 0.0000000000 |
| 28.0000000000 | 28.0000000000 | 0.0000000000 |
| 72.0000000000 | 72.0000000000 | 0.0000000000 |
| 407.0000000000 | 407.0000000000 | 0.0000000000 |
| 8.1250000000 | 8.1250000000 | 0.0000000000 |
| 15.7500000000 | 15.7500000000 | 0.0000000000 |
| 22.9014705882 | 16.6250000000 | 6.2764705882 |
| 23.6250000000 | 23.6250000000 | 0.0000000000 |
| 49.9235294118 | 31.2500000000 | 18.6735294118 |
| 155.1250000000 | 155.1250000000 | 0.0000000000 |
| 140.6250000000 | 140.6250000000 | 0.0000000000 |

1. **Листинг программы**

Main.cpp

#include "mfe.h"

#include <iostream>

**using** **namespace** std**;**

int main**()**

**{**

**try**

**{**

mfe m**;**

m**.**buildPortraitOfMatrix**();**

m**.**assemblyGlobalMatrix**();**

m**.**bc\_1\_set**();**

/\*m.bc\_2\_set();\*/

m**.**F\_set**();**

/\*m.toDense("matrix.txt");\*/

/\*m.bc\_2();\*/

m**.**toDense**(**"matrix1.txt"**);**

m**.**bc\_1**();**

m**.**toDense**(**"matrix2.txt"**);**

m**.**LOS**();**

m**.**writeToFile**(**m**.**q**);**

**}**

**catch** **(**int error**)**

**{**

**switch** **(**error**)**

**{**

**case** 0**:**

cout **<<** "Unable to read file!" **<<** endl**;**

**break;**

**case** 1**:**

cout **<<** "Point is out of range!" **<<** endl**;**

**break;**

**case** 2**:**

cout **<<** "Unable write result to file!" **<<** endl**;**

**break;**

**}**

**}**

**return** 0**;**

**}**

Mfe.cpp

#include "mfe.h"

//===========================================================================================

// Конструктор: читаем сетку, КЭ с номерами их узлов.

mfe**::**mfe**()**

**{**

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

std**::**ifstream inGrid**(**"node.txt"**);**

**if** **(**inGrid**.**is\_open**())**

**{**

inGrid **>>** Nuz**;**

MeshXY**.**resize**(**Nuz**);**

double x**,** y**;**

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

**{**

inGrid **>>** x **>>** y**;**

MeshXY**[**i**].**x **=** x**;**

MeshXY**[**i**].**y **=** y**;**

**}**

inGrid**.**close**();**

**}**

**else** **throw** BAD\_READ**;**

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

// Конечный элемент задается номерами его узлов

std**::**ifstream inElem**(**"elem.txt"**);**

**if** **(**inElem**.**is\_open**())**

**{**

inElem **>>** Nel**;**

FE**.**resize**(**Nel**);**

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

FE**[**i**].**resize**(**6**);**

size\_t n1**,** n2**,** n3**,** n4**,** n5**,** n6**;**

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

**{**

inElem **>>** n1 **>>** n2 **>>** n3 **>>** n4 **>>** n5 **>>** n6**;**

FE**[**i**]** **=** **{** n1**,** n2**,** n3**,** n4**,** n5**,** n6 **};**

**}**

inElem**.**close**();**

**}**

**else** **throw** BAD\_READ**;**

maxIter **=** 100000**;**

eps **=** 1e-14**;**

mats**.**lambda **=** 1**;**

mats**.**gamma **=** 0**;**

//Nbc2 = 1;

// Матрица жесткости

G**.**resize**(**6**);**

**for** **(**size\_t i **=** 0**;** i **<** 6**;** i**++)**

G**[**i**].**resize**(**6**);**

// Матрица масс

M**.**resize**(**6**);**

**for** **(**size\_t i **=** 0**;** i **<** 6**;** i**++)**

M**[**i**].**resize**(**6**);**

F\_local**.**resize**(**6**);**

F**.**resize**(**Nuz**);**

**}**

// Построить локальную матрицу жесткости

void mfe**::**buildLocalG**(**size\_t ielem**)**

**{**

double x1 **=** MeshXY**[**FE**[**ielem**][**0**]].**x**;**

double x2 **=** MeshXY**[**FE**[**ielem**][**1**]].**x**;**

double x3 **=** MeshXY**[**FE**[**ielem**][**2**]].**x**;**

double y1 **=** MeshXY**[**FE**[**ielem**][**0**]].**y**;**

double y2 **=** MeshXY**[**FE**[**ielem**][**1**]].**y**;**

double y3 **=** MeshXY**[**FE**[**ielem**][**2**]].**y**;**

double det **=** **(**x2 **-** x1**)** **\*** **(**y3 **-** y1**)** **-** **(**x3 **-** x1**)** **\*** **(**y2 **-** y1**);**

const double h1 **=** mats**.**lambda**;**

const double h2 **=** mats**.**lambda**;**

const double h3 **=** mats**.**lambda**;**

const double a11 **=** y2 **-** y3**;**

const double a12 **=** x3 **-** x2**;**

const double a21 **=** y3 **-** y1**;**

const double a22 **=** x1 **-** x3**;**

const double a31 **=** y1 **-** y2**;**

const double a32 **=** x2 **-** x1**;**

const double tt **=** 1.0 **/** 30.0 **/** det**;**

G**[**0**][**0**]** **=** tt **\*** **(**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**9**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**3**));**

G**[**1**][**0**]** **=** tt **\*** **(**a21 **\*** a11 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(-**2**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(-**1**));**

G**[**1**][**1**]** **=** tt **\*** **(**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**9**)** **+** h3 **\*** **(**3**));**

G**[**2**][**0**]** **=** tt **\*** **(**a31 **\*** a11 **+** a12 **\*** a32**)** **\*** **(**h1 **\*** **(-**2**)** **+** h2 **\*** **(-**1**)** **+** h3 **\*** **(-**2**));**

G**[**2**][**1**]** **=** tt **\*** **(**a31 **\*** a21 **+** a22 **\*** a32**)** **\*** **(**h1 **\*** **(-**1**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(-**2**));**

G**[**2**][**2**]** **=** tt **\*** **(**a31 **\*** a31 **+** a32 **\*** a32**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**9**));**

G**[**3**][**0**]** **=** tt **\*** **((**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(-**1**))** **+** **(**a11 **\*** a21 **+**

a12 **\*** a22**)** **\*** **(**h1 **\*** **(**14**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**3**)));**

G**[**3**][**1**]** **=** tt **\*** **((**a11 **\*** a21 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**14**)** **+** h3 **\*** **(**3**))** **+** **(**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(-**

2**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(-**1**)));**

G**[**3**][**2**]** **=** tt **\*** **((**a11 **\*** a31 **+** a12 **\*** a32**)** **\*** **(**h1 **\*** **(-**1**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(**3**))** **+** **(**a21 **\*** a31 **+**

a22 **\*** a32**)** **\*** **(**h1 **\*** **(-**2**)** **+** h2 **\*** **(-**1**)** **+** h3 **\*** **(**3**)));**

G**[**4**][**0**]** **=** tt **\*** **((**a21 **\*** a11 **+** a22 **\*** a12**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(-**1**)** **+** h3 **\*** **(-**2**))** **+** **(**a31 **\*** a11 **+**

a32 **\*** a12**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(-**1**)));**

G**[**4**][**1**]** **=** tt **\*** **((**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(-**1**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(-**2**))** **+** **(**a31 **\*** a21 **+**

a32 **\*** a22**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**14**)** **+** h3 **\*** **(**3**)));**

G**[**4**][**2**]** **=** tt **\*** **((**a21 **\*** a31 **+** a22 **\*** a32**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**14**))** **+** **(**a31 **\*** a31 **+** a32 **\*** a32**)** **\*** **(**h1 **\*** **(-**

1**)** **+** h2 **\*** **(-**2**)** **+** h3 **\*** **(**3**)));**

G**[**5**][**0**]** **=** tt **\*** **((**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(-**1**)** **+** h3 **\*** **(-**2**))** **+** **(**a31 **\*** a11 **+**

a32 **\*** a12**)** **\*** **(**h1 **\*** **(**14**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**3**)));**

G**[**5**][**1**]** **=** tt **\*** **((**a11 **\*** a21 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(-**1**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(-**2**))** **+** **(**a31 **\*** a21 **+**

a32 **\*** a22**)** **\*** **(**h1 **\*** **(-**2**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(-**1**)));**

G**[**5**][**2**]** **=** tt **\*** **((**a11 **\*** a31 **+** a12 **\*** a32**)** **\*** **(**h1 **\*** **(**3**)** **+** h2 **\*** **(**3**)** **+** h3 **\*** **(**14**))** **+** **(**a31 **\*** a31 **+** a32 **\*** a32**)** **\*** **(**h1 **\*** **(-**

2**)** **+** h2 **\*** **(-**1**)** **+** h3 **\*** **(**3**)));**

G**[**3**][**3**]** **=** tt **\*** **((**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**24**)** **+** h3 **\*** **(**8**))** **+** **(**a11 **\*** a21 **+**

a12 **\*** a22**)** **\*** **(**h1 **\*** **(**16**)** **+** h2 **\*** **(**16**)** **+** h3 **\*** **(**8**))** **+** **(**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(**24**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**)));**

G**[**4**][**3**]** **=** tt **\*** **((**a11 **\*** a21 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(**4**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**))** **+** **(**a11 **\*** a31 **+** a12 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)**

**+** h2 **\*** **(**24**)** **+** h3 **\*** **(**8**))** **+** **(**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**4**)** **+** h3 **\*** **(**8**))** **+** **(**a21 **\*** a31 **+** a22 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**8**)** **+**

h3 **\*** **(**4**)));**

G**[**4**][**4**]** **=** tt **\*** **((**a21 **\*** a21 **+** a22 **\*** a22**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**24**))** **+** **(**a21 **\*** a31 **+**

a22 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**16**)** **+** h3 **\*** **(**16**))** **+** **(**a31 **\*** a31 **+** a32 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**24**)** **+** h3 **\*** **(**8**)));**

G**[**5**][**3**]** **=** tt **\*** **((**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**4**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**))** **+** **(**a11 **\*** a21 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(**8**)**

**+** h2 **\*** **(**4**)** **+** h3 **\*** **(**8**))** **+** **(**a11 **\*** a31 **+** a12 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**4**))** **+** **(**a21 **\*** a31 **+** a22 **\*** a32**)** **\*** **(**h1 **\*** **(**24**)** **+** h2 **\*** **(**8**)** **+**

h3 **\*** **(**8**)));**

G**[**5**][**4**]** **=** tt **\*** **((**a11 **\*** a21 **+** a12 **\*** a22**)** **\*** **(**h1 **\*** **(**24**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**))** **+** **(**a11 **\*** a31 **+**

a12 **\*** a32**)** **\*** **(**h1 **\*** **(**4**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**))** **+** **(**a21 **\*** a31 **+** a22 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**4**)** **+** h3 **\*** **(**8**))** **+** **(**a31 **\*** a31 **+**

a32 **\*** a32**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**4**)));**

G**[**5**][**5**]** **=** tt **\*** **((**a11 **\*** a11 **+** a12 **\*** a12**)** **\*** **(**h1 **\*** **(**8**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**24**))** **+** **(**a11 **\*** a31 **+**

a12 **\*** a32**)** **\*** **(**h1 **\*** **(**16**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**16**))** **+** **(**a31 **\*** a31 **+** a32 **\*** a32**)** **\*** **(**h1 **\*** **(**24**)** **+** h2 **\*** **(**8**)** **+** h3 **\*** **(**8**)));**

**}**

// Построить локальную матрицу масс

void mfe**::**buildLocalM**(**size\_t ielem**)**

**{**

double x1 **=** MeshXY**[**FE**[**ielem**][**0**]].**x**;**

double x2 **=** MeshXY**[**FE**[**ielem**][**1**]].**x**;**

double x3 **=** MeshXY**[**FE**[**ielem**][**2**]].**x**;**

double y1 **=** MeshXY**[**FE**[**ielem**][**0**]].**y**;**

double y2 **=** MeshXY**[**FE**[**ielem**][**1**]].**y**;**

double y3 **=** MeshXY**[**FE**[**ielem**][**2**]].**y**;**

double detD **=** **(**x2 **-** x1**)** **\*** **(**y3 **-** y1**)** **-** **(**x3 **-** x1**)** **\*** **(**y2 **-** y1**);**

double k **=** fabs**(**detD**)** **\*** mats**.**gamma **/** 360.0**;**

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

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

**if** **(**i **==** j**)**

M**[**i**][**j**]** **=** 6 **\*** k**;**

**else**

M**[**i**][**j**]** **=** **-**k**;**

**for** **(**int i **=** 3**;** i **<** 6**;** i**++)**

**for** **(**int j **=** 3**;** j **<** 6**;** j**++)**

**if** **(**i **==** j**)**

M**[**i**][**j**]** **=** 32 **\*** k**;**

**else**

M**[**i**][**j**]** **=** 16 **\*** k**;**

M**[**0**][**4**]** **=** M**[**1**][**5**]** **=** M**[**2**][**3**]** **=** M**[**3**][**2**]** **=** M**[**4**][**0**]** **=** M**[**5**][**1**]** **=** **-**4 **\*** k**;**

**}**

void mfe**::**buildLocalF**(**size\_t ielem**)**

**{**

double x1 **=** MeshXY**[**FE**[**ielem**][**0**]].**x**;**

double x2 **=** MeshXY**[**FE**[**ielem**][**1**]].**x**;**

double x3 **=** MeshXY**[**FE**[**ielem**][**2**]].**x**;**

double y1 **=** MeshXY**[**FE**[**ielem**][**0**]].**y**;**

double y2 **=** MeshXY**[**FE**[**ielem**][**1**]].**y**;**

double y3 **=** MeshXY**[**FE**[**ielem**][**2**]].**y**;**

double detD **=** **(**x2 **-** x1**)** **\*** **(**y3 **-** y1**)** **-** **(**x3 **-** x1**)** **\*** **(**y2 **-** y1**);**

double tt **=** detD **/** 360.0**;**

**for** **(**int i **=** 0**;** i **<** 6**;** i**++)**

**{**

F\_local**[**i**]** **=** f**(**MeshXY**[**FE**[**ielem**][**i**]].**x**,** MeshXY**[**FE**[**ielem**][**i**]].**y**);**

**}**

double f1 **=** F\_local**[**0**];**

double f2 **=** F\_local**[**1**];**

double f3 **=** F\_local**[**2**];**

double f4 **=** F\_local**[**3**];**

double f5 **=** F\_local**[**4**];**

double f6 **=** F\_local**[**5**];**

F\_local**[**0**]** **=** **(**6**)** **\*** f1 **+** **(-**1**)** **\*** f2 **+** **(-**1**)** **\*** f3 **+** **(**0**)** **\*** f4 **+** **(-**4**)** **\*** f5 **+** **(**0**)** **\*** f6**;**

F\_local**[**1**]** **=** **(-**1**)** **\*** f1 **+** **(**6**)** **\*** f2 **+** **(-**1**)** **\*** f3 **+** **(**0**)** **\*** f4 **+** **(**0**)** **\*** f5 **+** **(-**4**)** **\*** f6**;**

F\_local**[**2**]** **=** **(-**1**)** **\*** f1 **+** **(-**1**)** **\*** f2 **+** **(**6**)** **\*** f3 **+** **(-**4**)** **\*** f4 **+** **(**0**)** **\*** f5 **+** **(**0**)** **\*** f6**;**

F\_local**[**3**]** **=** **(**0**)** **\*** f1 **+** **(**0**)** **\*** f2 **+** **(-**4**)** **\*** f3 **+** **(**32**)** **\*** f4 **+** **(**16**)** **\*** f5 **+** **(**16**)** **\*** f6**;**

F\_local**[**4**]** **=** **(-**4**)** **\*** f1 **+** **(**0**)** **\*** f2 **+** **(**0**)** **\*** f3 **+** **(**16**)** **\*** f4 **+** **(**32**)** **\*** f5 **+** **(**16**)** **\*** f6**;**

F\_local**[**5**]** **=** **(**0**)** **\*** f1 **+** **(-**4**)** **\*** f2 **+** **(**0**)** **\*** f3 **+** **(**16**)** **\*** f4 **+** **(**16**)** **\*** f5 **+** **(**32**)** **\*** f6**;**

**for** **(**int i **=** 0**;** i **<** 5**;** i**++)**

F\_local**[**i**]** **\*=** tt**;**

**}**

double mfe**::**f**(**double x**,** double y**)**

**{**

/\*return -6 \* x - 6 \* y;\*/

**return** 0**;**

**}**

double mfe**::**u**(**double x**,** double y**)**

**{**

/\*return x \* x \* x + y \* y \* y;\*/

**return** 1337**;**

**}**

// Добавить элемент в глобальную матрицу

void mfe**::**addElementToGlobal**(**size\_t i**,** size\_t j**,** double elem**)**

**{**

**if** **(**i **==** j**)**

**{**

di**[**i**]** **+=** elem**;**

**return;**

**}**

**else**

**{**

**for** **(**size\_t ind **=** ig**[**i**];** ind **<** ig**[**i **+** 1**];** ind**++)**

**if** **(**jg**[**ind**]** **==** j**)**

**{**

gg**[**ind**]** **+=** elem**;**

**return;**

**}**

**}**

**}**

// Сборка глобальной матрицы

void mfe**::**assemblyGlobalMatrix**()**

**{**

di**.**resize**(**Nuz**);**

gg**.**resize**(**ig**.**back**());**

**for** **(**size\_t ielem **=** 0**;** ielem **<** Nel**;** ielem**++)**

**{**

buildLocalG**(**ielem**);**

buildLocalM**(**ielem**);**

**for** **(**size\_t i **=** 0**;** i **<** FE**[**ielem**].**size**();** i**++)**

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

addElementToGlobal**(**FE**[**ielem**][**i**],** FE**[**ielem**][**j**],** G**[**i**][**j**]+** M**[**i**][**j**]);**

**}**

G**.**clear**();**

**}**

// Построить портрет глобальной матрицы

void mfe**::**buildPortraitOfMatrix**()**

**{**

std**::**vector**<**std**::**vector**<**size\_t**>>** list**;**

list**.**resize**(**Nuz**);**

list**[**0**].**push\_back**(**0**);**

// Идем по всем КЭ

**for** **(**size\_t ielem **=** 0**;** ielem **<** Nel**;** ielem**++)**

**{**

// Берем 1-ую соответствующую элементу базисную функцию

**for** **(**size\_t i **=** 0**;** i **<** FE**[**ielem**].**size**()** **-** 1**;** i**++)**

// Идем по всем остальным функциям, начиная со второй

**for** **(**size\_t j **=** i **+** 1**;** j **<** FE**[**ielem**].**size**();** j**++)**

**{**

// Нужно добавить первую функцию(меньшую) в список ко всем

// функциям, относящимся к КЭ

// Поэтому определяем позицию, куда будем добавлять (в какой список)

size\_t insertPos **=** FE**[**ielem**][**j**];**

// Берем сам элемент, который будем вставлять

size\_t element **=** FE**[**ielem**][**i**];**

bool isIn **=** **false;**

// Проверим, есть ли уже этот элемент в списке

**for** **(**size\_t k **=** 0**;** k **<** list**[**insertPos**].**size**()** **&&** **!**isIn**;** k**++)**

**if** **(**element **==** list**[**insertPos**][**k**])**

isIn **=** **true;**

// Если он в списке не найден, то добавляем его

**if** **(!**isIn**)**

list**[**insertPos**].**push\_back**(**element**);**

**}**

**}**

// Сортируем все получившиеся списки (по возрастанию номеров)

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

**if** **(!**isOrdered**(**list**[**i**]))**

sort**(**list**[**i**].**begin**(),** list**[**i**].**end**());**

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

// Формируем массив ig

ig**.**resize**(**Nuz **+** 1**);**

// 1-ый и 2-ой элементы всегда равны 1, но мы будем нумеровать с 0

ig**[**0**]** **=** 0**;**

ig**[**1**]** **=** 0**;**

**for** **(**size\_t i **=** 1**;** i **<** list**.**size**();** i**++)**

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

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

// Формируем массив jg

jg**.**resize**(**ig**.**back**());**

**for** **(**size\_t i **=** 1**,** j **=** 0**;** i **<** Nuz**;** i**++)**

**{**

**for** **(**size\_t k **=** 0**;** k **<** list**[**i**].**size**();** k**++)**

jg**[**j**++]** **=** list**[**i**][**k**];**

**}**

**}**

// Проверка списка на упорядоченность по возрастанию

bool mfe**::**isOrdered**(**const pvector**&** v**)**

**{**

**if(**v**.**size**()** **==** 0**)**

**return** **true;**

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

**if** **(**v**[**i **+** 1**]** **<** v**[**i**])**

**return** **false;**

**return** **true;**

**}**

// Перевод матрицы в плотный формат

void mfe**::**toDense**(**const std**::**string \_dense**)**

**{**

mat**.**resize**(**MeshXY**.**size**());**

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

mat**[**i**].**resize**(**MeshXY**.**size**(),** 0**);**

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

**{**

mat**[**i**][**i**]** **=** di**[**i**];**

**for** **(**size\_t j **=** ig**[**i**];** j **<** ig**[**i **+** 1**];** j**++)**

**{**

mat**[**i**][**jg**[**j**]]** **=** gg**[**j**];**

**}**

**}**

std**::**ofstream dense**(**\_dense**);**

dense**.**precision**(**2**);**

**if** **(**dense**.**is\_open**())**

**{**

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

**{**

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

dense **<<** std**::**left **<<** std**::**setw**(**10**)** **<<** mat**[**i**][**j**];**

dense **<<** std**::**endl **<<** std**::**endl**;**

**}**

**}**

mat**.**clear**();**

**}**

void mfe**::**bc\_1**()**

**{**

std**::**vector**<**int**>** bc1nodes**(**Nuz**,** **-**1**);**

**for** **(**size\_t i **=** 0**;** i **<** bc1**.**size**();** i**++)**

bc1nodes**[**bc1**[**i**].**n\_i**]** **=** i**;** // В узле задано краевое

size\_t k**;**

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

**{**

**if** **(**bc1nodes**[**i**]** **!=** **-**1**)**

**{**

di**[**i**]** **=** 1.0**;**

F**[**i**]** **=** bc1**[**bc1nodes**[**i**]].**Ug\_i**;**

**for** **(**size\_t j **=** ig**[**i**];** j **<** ig**[**i **+** 1**];** j**++)**

**{**

k **=** jg**[**j**];**

**if** **(**bc1nodes**[**k**]** **==** **-**1**)**

F**[**k**]** **-=** gg**[**j**]** **\*** F**[**i**];**

gg**[**j**]** **=** 0.0**;**

**}**

**}**

**else**

**{**

**for** **(**size\_t j **=** ig**[**i**];** j **<** ig**[**i **+** 1**];** j**++)**

**{**

k **=** jg**[**j**];**

**if** **(**bc1nodes**[**k**]** **!=** **-**1**)**

**{**

F**[**i**]** **-=** gg**[**j**]** **\*** F**[**k**];**

gg**[**j**]** **=** 0.0**;**

**}**

**}**

**}**

**}**

**}**

//устанавливаем первые краевые по всей границе

void mfe**::**bc\_1\_set**()**

**{**

std**::**vector**<**int**>** nodes**{**0**,** 6**,** 2**,** 11**,** 4**,** 10**,** 3**,** 8**,** 1**,** 5 **};**

**for** **(**int n **:** nodes**)** **{**

\_bc1 b**;**

b**.**n\_i **=** n**;**

b**.**Ug\_i **=** u**(**MeshXY**[**n**].**x**,** MeshXY**[**n**].**y**);**

bc1**.**push\_back**(**b**);**

**}**

**}**

void mfe**::**F\_set**()**

**{**

**for** **(**int i **=** 0**;** i **<** Nuz**;** i**++)**

**{**

F**[**i**]** **=** f**(**MeshXY**[**i**].**x**,** MeshXY**[**i**].**y**);**

**}**

**}**

// Учет вторых краевых

//void mfe::bc\_2()

//{

// for (size\_t i = 0; i < Nbc2; i++)

// {

// size\_t el\_i = bc2[i].n\_i;

// size\_t loc\_node1\_i = bc2[i].loc\_node1\_i;

// size\_t loc\_node2\_i = bc2[i].loc\_node2\_i;

// size\_t loc\_node3\_i = bc2[i].loc\_node3\_i;

// double t1= bc2[i].Tetta\_i[0];

// double t2= bc2[i].Tetta\_i[1];;

// double t3= bc2[i].Tetta\_i[2];;

//

// size\_t ind\_1 = FE[el\_i][loc\_node1\_i];

// size\_t ind\_2 = FE[el\_i][loc\_node2\_i];

// size\_t ind\_3 = FE[el\_i][loc\_node3\_i];

// double x1 = MeshXY[ind\_1].x;

// double x3 = MeshXY[ind\_3].x;

// double y1 = MeshXY[ind\_1].y;

//

// double y3 = MeshXY[ind\_3].y;

// std::vector<double> b\_loc(3);

// double hm = sqrt((x1 - x3) \* (x1 - x3) + (y1 - y3) \* (y1 - y3)) / 30;

//

// b\_loc[0] = hm \* ((4) \* t1 + (2) \* t2 + (-1) \* t3);

// b\_loc[1] = hm \* ((2) \* t1 + (16) \* t2 + (2) \* t3);

// b\_loc[0] = hm \* ((-1) \* t1 + (2) \* t2 + (4) \* t3);

// F[ind\_1] += b\_loc[0];

// F[ind\_2] += b\_loc[1];

// F[ind\_3] += b\_loc[2];

// }

//}

//установить вторые краевые на одной из границ

//void mfe::bc\_2\_set()

//{

// \_bc2 b;

// b.loc\_node1\_i = 1;

// b.loc\_node2\_i = 4;

// b.loc\_node3\_i = 2;

// b.n\_i = 2;

// mvector Tetta = { -1, -1, -1 };

// b.Tetta\_i = Tetta;

// bc2.push\_back(b);

//}

//===========================================================================================

// Решение СЛАУ

mvector mfe**::**mult**(**const mvector **&**v**)**

**{**

mvector res**(**v**.**size**(),** 0**);**

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

**{**

res**[**i**]** **=** di**[**i**]** **\*** v**[**i**];**

**for** **(**size\_t j **=** ig**[**i**];** j **<** ig**[**i **+** 1**];** j**++)**

**{**

res**[**i**]** **+=** gg**[**j**]** **\*** v**[**jg**[**j**]];**

res**[**jg**[**j**]]** **+=** gg**[**j**]** **\*** v**[**i**];**

**}**

**}**

**return** res**;**

**}**

double mfe**::**EuclideanNorm**(**mvector **&** x**)** **{**

double scalar **=** 0**;**

**for** **(**int i **=** 0**;** i **<** Nuz**;** i**++)**

scalar **+=** x**[**i**]** **\*** x**[**i**];**

scalar **=** sqrt**(**scalar**);**

**return** scalar**;**

**}**

// Локально-оптимальная схема

void mfe**::**LOS**()**

**{**

q**.**resize**(**di**.**size**(),** 0**);**

r**.**resize**(**di**.**size**());**

double alpha**,** beta**;**

mvector Ar**;**

r **=** F **-** mult**(**q**);**

z **=** r**;**

p **=** mult**(**z**);**

double rr **=** **(**r **\*** r**);**

**for** **(**size\_t k **=** 0**;** k **<** maxIter **&&** rr **>=** eps**;** k**++)**

**{**

double pp **=** **(**p **\*** p**);**

alpha **=** **(**p **\*** r**)** **/** pp**;**

q **=** q **+** alpha **\*** z**;**

rr **=** **(**r **\*** r**)** **-** alpha **\*** alpha **\*** pp**;**

r **=** r **-** alpha **\*** p**;**

Ar **=** mult**(**r**);**

beta **=** **-(**p **\*** Ar**)** **/** pp**;**

z **=** r **+** beta **\*** z**;**

p **=** Ar **+** beta **\*** p**;**

**}**

r**.**clear**();**

p**.**clear**();**

z**.**clear**();**

**}**

// Метод сопряженных градиентов

void mfe**::**MSG**()** **{**

q**.**resize**(**di**.**size**(),** 0**);**

r**.**resize**(**di**.**size**());**

double normF **=** sqrt**(**F **\*** F**);**

double alpha**,** beta**;**

r **=** F **-** mult**(**q**);**

z **=** r**;**

double rr **=** **(**r **\*** r**);**

**for** **(**size\_t k **=** 0**;** k **<** maxIter **&&** sqrt**(**rr**)** **/** normF **>=** eps**;** k**++)**

**{**

alpha **=** **(**r **\*** r**)** **/** **(**mult**(**z**)** **\*** z**);**

q **=** q **+** alpha **\*** z**;**

r **=** r **-** alpha **\*** mult**(**z**);**

beta **=** **(**r **\*** r**)** **/** rr**;**

rr **=** r **\*** r**;**

z **=** r **+** beta **\*** z**;**

**}**

r**.**clear**();**

z**.**clear**();**

**}**

// Записать результат в файл

void mfe**::**writeToFile**(**mvector q**)**

**{**

FILE **\***File**;**

fopen\_s**(&**File**,** "q.txt"**,** "w"**);**

**if** **(**File**)** **{**

fprintf**(**File**,** "%-20s%-20s%-20s%\n"**,** "q"**,** "u"**,** "|q-u|"**);**

**for** **(**int i **=** 0**;** i **<** Nuz**;** i**++)**

fprintf**(**File**,** "%-20.10f%-20.10f%-20.10f%\n"**,** q**[**i**],** u**(**MeshXY**[**i**].**x**,** MeshXY**[**i**].**y**),** abs**(**q**[**i**]** **-** u**(**MeshXY**[**i**].**x**,** MeshXY**[**i**].**y**)));**

fclose**(**File**);**

**}**

**else**

**throw** BAD\_WRITE**;**

**}**

Mfe.h

#pragma once

#ifndef MFE\_H

#define MFE\_H

#include <vector>

#include <functional>

#include <fstream>

#include <algorithm>

#include <iomanip>

struct coordinates

**{**

double x**;**

double y**;**

**};**

**typedef** std**::**vector **<**coordinates**>** grid**;**

**typedef** std**::**vector**<**size\_t**>** pvector**;**

**typedef** std**::**vector**<**double**>** mvector**;**

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

**typedef** std**::**vector**<**std**::**vector**<**size\_t**>>** finiteElem**;**

**typedef** std**::**function**<**double**(**double**,** double**)** **>** ScalFunc2D**;**

enum exceptions **{** BAD\_READ**,** OUT\_OF\_AREA**,** BAD\_WRITE **};**

class mfe

**{**

struct material

**{**

double gamma**;**

double lambda**;**

**};**

struct \_bc1

**{**

size\_t n\_i**;**

double Ug\_i**;**

**};**

struct \_bc2

**{**

size\_t n\_i**;**

size\_t loc\_node1\_i**;**

size\_t loc\_node2\_i**;**

size\_t loc\_node3\_i**;**

mvector Tetta\_i**;**

**};**

public**:**

mfe**();**

//========================================================================

void buildPortraitOfMatrix**();**

void buildLocalG**(**size\_t ielem**);**

void buildLocalM**(**size\_t ielem**);**

void buildLocalF**(**size\_t ielem**);**

void addElementToGlobal**(**size\_t i**,** size\_t j**,** double elem**);**

void assemblyGlobalMatrix**();**

void toDense**(**const std**::**string \_dense**);**

double f**(**double x**,** double y**);** //функция правой части

double u**(**double x**,** double y**);** //функция которую ищем

void bc\_1**();**

void bc\_1\_set**();**

void F\_set**();**

void bc\_2**();**

void bc\_2\_set**();**

//========================================================================

mvector mult**(**const mvector **&**v**);**

void LOS**();**

void MSG**(** **);**

void writeToFile**(**mvector q**);**

double EuclideanNorm**(**mvector**&** x**);**

public**:**

matrix G**;** // Матрица жесткости

matrix M**;** // Матрица масс

mvector q**;** // Вектор весов

mvector F**;** // Вектор правой части

mvector F\_local**;** // Вектор правой части локальный

size\_t Nuz**;** // Размер сетки

grid MeshXY**;** // Сетка

size\_t Nel**;** // Количество КЭ

finiteElem FE**;** // Конечные элементы

material mats**;** // Материалы

std**::**vector **<**\_bc1**>** bc1**;**

size\_t Nbc2**;**

std**::**vector**<**\_bc2**>** bc2**;**

// Глобальная матрица

mvector di**;**

mvector gg**;**

pvector ig**;**

pvector jg**;**

matrix mat**;**

bool isOrdered**(**const pvector**&** v**);**

mvector r**;**

mvector z**;**

mvector p**;**

size\_t maxIter**;**

double eps**;**

**};**

inline mvector **operator+(**const mvector**&** a**,** const mvector**&** b**)**

**{**

mvector res **=** a**;**

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

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

**return** res**;**

**}**

inline mvector **operator-(**const mvector**&** a**,** const mvector**&** b**)**

**{**

mvector res **=** a**;**

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

res**[**i**]** **-=** b**[**i**];**

**return** res**;**

**}**

inline double **operator\*(**const mvector**&** a**,** const mvector**&** b**)**

**{**

double scalar **=** 0.0**;**

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

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

**return** scalar**;**

**}**

inline mvector **operator\*(**double c**,** const mvector**&** a**)**

**{**

std**::**vector**<**double**>** res **=** a**;**

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

res**[**i**]** **\*=** c**;**

**return** res**;**

**}**

#endif