МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ

Кафедра прикладной математики

Курсовая работа
по дисциплине «Численные методы»

МКЭ для решения параболической краевой задачи

ФПМИ

Группа	ПМ-91
Студенты	ЗАТОЛОЦКАЯ ЮЛИЯ
Преподаватель	ПЕРСОВА МАРИНА ГЕННАДЬЕВНА
Дата	21.05.2022

Новосибирск

1 Постановка задачи

Реализовать МКЭ для двумерной задачи для параболического уравнения давления в цилиндрической системе координат. Конечные элементы - треугольники с линейными базисными функциями. Схема по времени четырехслойная неявная.

2 Теория

Решаемое уравнение в общем виде:

$$-\operatorname{div}(\sum_{m=1}^{M} K\frac{\kappa^m}{\mu^m}\operatorname{grad} P) + \sigma\frac{\partial P}{\partial t} = 0, \tag{1}$$

где K - структурная проницаемость, $\kappa(S^2)$ - относительная фазовая проницаемость (S^2 – нефтенасыщенность), μ – вязкость фазы, M – количество фаз. Заданным в области Ω с границей $S = S_1 \cup S_2 \cup S_3$, и краевыми условиями

$$P|_{S_1} = P_g, \tag{2}$$

$$\sum_{m=1}^{M} K\frac{\kappa^m}{\mu^m}\frac{dP}{dn}\bigg|_{S_2} = \theta, \tag{3}$$

Решаемое уравнение в цилиндрической двумерной системе координат:

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(r\lambda\frac{\partial P}{\partial r}\right) - \frac{\partial}{\partial z}\left(\lambda\frac{\partial P}{\partial z}\right) + \sigma\frac{\partial P}{\partial t} = 0, \lambda = \sum_{m=1}^{M} K\frac{\kappa^m}{\mu^m} \tag{4}$$

Формулы для линейных базисных функций треугольных элементов:

$$\widehat{\psi_i}(r,\varphi) = \alpha_0^i + \alpha_1^i r + \alpha_2^i z, \quad i = 1,\dots,3 \tag{5}$$

$$\alpha = \frac{1}{\det D}\begin{pmatrix} r_2z_3 - r_3z_2 & z_2 - z_3 & r_3 - r_2 \\ r_3z_1 - r_1z_3 & z_3 - z_1 & r_1 - r_3 \\ r_1z_2 - r_1z_3 & z_1 - z_2 & r_2 - r_1 \end{pmatrix} \tag{6}$$

$$\det D = (r_2 - r_1)(z_3 - z_1) - (r_3 - r_1)(z_2 - z_1) \tag{7}$$

Значение конечно-элементной аппроксимации на конечном элементе равно:

$$u^h = \sum_{j=1}^{n} q_j\psi_j, \tag{8}$$

Аналитические выражения для вычисления локальных матриц:

$$\widehat{\mathbf{G}}_{ij} = \frac{1}{6}\overline{\lambda}\left(\alpha_1^i\alpha_1^j + \alpha_2^i\alpha_2^j\right)(r_1 + r_2 + r_3)|\det\mathbf{D}|, \quad i = 1,\dots,3 \tag{9}$$

$$\mathbf{M} = \frac{|\det\mathbf{D}|}{120}\begin{pmatrix} 6r_1 + 2r_2 + 2r_3 & 2r_1 + 2r_2 + r_3 & 2r_1 + r_2 + 2r_3 \\ 2r_1 + 2r_2 + r_3 & 2r_1 + 6r_2 + 2r_3 & r_1 + 2r_2 + 2r_3 \\ 2r_1 + r_2 + 2r_3 & r_1 + 2r_2 + 2r_3 & 2r_1 + 2r_2 + 6r_3 \end{pmatrix} \tag{10}$$

Четырехслойная неявная схема по времени:

$$u(x,y,t) \approx u^{j-3}\eta_3^j(t) + u^{j-2}\eta_2^j(t) + u^{j-1}\eta_1^j(t) + u^j\eta_0^j(t), \tag{11}$$

$$\eta_3^j(t) = \frac{(t-t_{j-2})(t-t_{j-1})(t-t_j)}{(t_{j-3}-t_{j-2})(t_{j-3}-t_{j-1})(t_{j-3}-t_j)}, \tag{12}$$

$$\eta_2^j(t) = \frac{(t-t_{j-3})(t-t_{j-1})(t-t_j)}{(t_{j-2}-t_{j-3})(t_{j-2}-t_{j-1})(t_{j-2}-t_j)}, \tag{13}$$

$$\eta_1^j(t) = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_j)}{(t_{j-1}-t_{j-3})(t_{j-1}-t_{j-2})(t_{j-1}-t_j)}, \tag{14}$$

$$\eta_0^j(t) = \frac{(t-t_{j-3})(t-t_{j-2})(t-t_{j_1})}{(t_j-t_{j-3})(t_j-t_{j-2})(t_j-t_{j-1})}, \tag{15}$$

$$\frac{\partial}{\partial t}(u^{j-3}\eta_3^j(t) + u^{j-2}\eta_2^j(t) + u^{j-1}\eta_1^j(t) + u^j\eta_0^j(t)) - div(\lambda \operatorname{grad} u^j) = f^j, \tag{16}$$

$$u^{j-3}\frac{\partial\eta_3^j(t)}{\partial t} + u^{j-2}\frac{\partial\eta_2^j(t)}{\partial t} + u^{j-1}\frac{\partial\eta_1^j(t)}{\partial t} + u^j\frac{\partial\eta_0^j(t)}{\partial t} - div(\lambda \operatorname{grad} u^j) = f^j, \tag{17}$$

Подставляя это в уравнение Галёркина, получаем СЛАУ из глобальных матриц:

$$\left(\frac{\partial\eta_0^j(t)}{\partial t}\mathbf{M} + \mathbf{G}\right) q^j = b^j - \frac{\partial\eta_3^j(t)}{\partial t}\mathbf{M}q^{j-3} - \frac{\partial\eta_2^j(t)}{\partial t}\mathbf{M}q^{j-2} - \frac{\partial\eta_1^j(t)}{\partial t}\mathbf{M}q^{j-1}, \tag{18}$$

3 Описание разработанной программы

3.1 Структуры данных

Программа реализована с использованием двух классов: MFE и GRID.

- std::vector <std::pair<double, double> > MeshRZ – массив размером Nuz для хранения координат точек
- std::vector<std::vector<size_t> > FE – двумерный массив размером Nelx3 для хранения конечных элементов, каждый из которых задан номерами 3 – х узлов
- std::vector<double> Mu – массив размером Nph для хранения вязкостей
- std::vector<material> mats – массив размером Nel для хранения свойств материалов, которые хранятся в структуре material, на каждом конечном элементе
- std::vector <std::pair<int, double> > bc1 – массив размером Nbc1, в котором хранятся данные о первых краевых условиях (номер узла, значение исходной функции в этом узле) на удаленной границе
- std::vector<_bc2> bc2 – массив размером Nbc2, в котором хранятся данные о вторых краевых условиях, описанных в структуре _bc2
- std::vector<double> di, gg, std::vector<size_t> ig, jg – массивы для хранения глобальной матрицы
- std::vector<double> F– массив для хранения вектора правой части
- std::vector<double> q – вектор весов (давление по узлам сетки)
- std::vector<std::vector<double> > G – двумерный массив для хранения матрицы жесткости

3.2 Расчет давления

- Входные данные
  - Конечноэлементная сетка
  - Сетка по времени
  - Массивы с информацией о первых и вторых краевых условиях
  - Массив с информацией о свойствах материалов
  - Массив с вязкостями фаз
- Выходные данные:
  - Давление в каждом узле сетки

3.3 Построение сетки

Алгоритм работы программы формирования сетки (треугольники rz, ось z направлена вниз, ось r - вправо)

1. Считываются данные из входных файлов
2. Формируется файл узлов node.txt
   (a) Строится вектор координат сетки по z (число разбиений задано для каждого слоя)
   (b) Добавляются координаты по z, соответствующие краям зон перфорации (если они не совпадают с уже имеющимися разбиениями)
   (c) Строится вектор координат по r по принципу геометрической прогрессии со знаменателем kr (коэф. разрядки), разрядка происходит слева направо (от скважины)
   (d) Если задана вложенная сетка, между каждыми двумя координатами добавляется новая
   (e) Координаты всех узлов записываются в выходной файл
3. Формируется файл конечных элементов elem.txt: зная число разбиений по z и по r, вычисляем глобальные номера узлов для каждого элемента и записываем в выходной файл. Нумеруем слева направо, сверху вниз.
4. Формируется файл mat.txt: зная порядок нумерации элементов и имея векторы координат сетки, легко узнать число элементов в каждом слое. Для каждого слоя записываем значения K, Phi и $S2$ в выходной файл столько раз, сколько конечных элементов в этом слое.
5. Формируется файл первых краевых условий bc1.txt: так как краевое на правой границе задано константой, вычисляются номера узлов на правой границе, и константа заносится в файл для всех этих узлов.
6. Формируется файл вторых краевых условий bc2.txt: для каждой зоны перфорации вычисляются номера конечных элементов, задается ребро локальными номерами узлов, и данные заносятся в выходной файл.

4 Тесты

Во всех исследованиях заданы следующие параметры $\lambda = \gamma = \sigma = \chi = 1$. СЛАУ решается методом сопряженных градиентов для симметричных матриц. Учет первых, вторых краевых.

4.1 Тесты на равномерных сетках

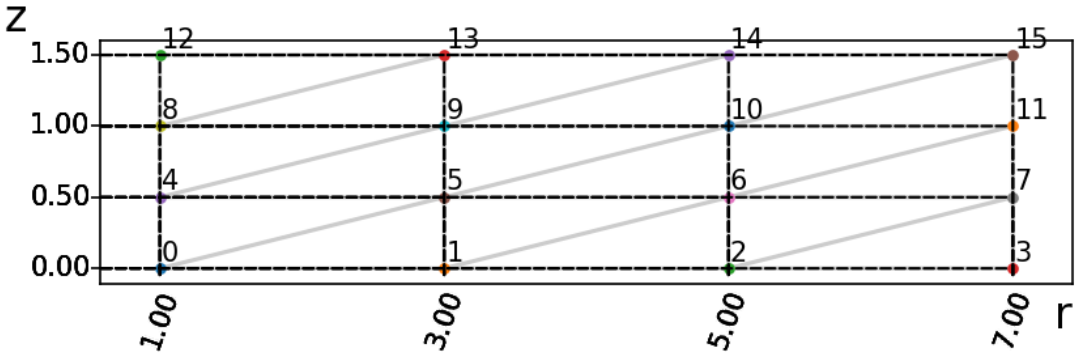

Рис. 1: Сетка по пространству

Сетка по времени: $t \in [1, 8]$, количество элементов сетки 7. Все сетки равномерные. Далее в таблицах будут указаны две функции: $\text{space}(r, z)$ и $\text{time}(t)$, итоговая функция u будет формироваться как $u(r, z, t) = \text{space}(r, z) + \text{time}(t)$. В таблицах указана норма разности векторов q для найденного решения и q, полученного из истинного значения функции.

Таблица 1: Равномерные сетки

time(t) / space(r, z)	0	t	t^2	t^3	t^4
1	1.54e-15	5.10e-15	2.61e-14	8.04e-14	9.53e+00
$r + z$	1.17e+00	1.17e+00	1.17e+00	1.17e+00	9.55e+00
$r^2 + z$	1.12e+00	1.12e+00	0.12e+00	1.12e+00	9.58e+00
$r^2z + z^3$	2.65e+01	2.65e+01	2.65e+01	2.65e+01	3.43e+01
rz^2	1.645e+01	1.65e+01	1.65e+01	1.65e+01	9.64e+00

Вывод: порядок точности по времени равен 3
Вывод: точно аппроксимируются только функции от времени до 3-го порядка. Для проверки правильности программы вектор правой части считался аналитически, равен нулю только при постоянной функции по простанству.

4.2 Тесты на неравномерных сетках

4.2.1 Неравномерная сетка по времени

Сетка по времени: $t \in [1, 8]$, количество элементов сетки 7. Исследуем зависимость от коэффициента разрядки k_t. Сетка по пространству с рис. 1. Функция: $u(r, z, t) = rz^2 + t^4$
Вывод: Наблюдается уменьшение погрешности при уменьшении коэффициента разрядки по времени.

Таблица 2: Расчет для неравномерной сетки по времени

k_t	0.9	0.7	0.5	0.3	0.1
$\|\|q - q^*\|\|$	9.64e+00	1.11e+01	5.66e+00	1.71e+00	3.39e-02

Функция: $u(r, z, t) = r^2 + z + t^3$

Таблица 3: Расчет для неравномерной сетки по времени

k_t	0.9	0.7	0.5	0.3	0.1
$\|\|q - q^*\|\|$	1.07e+00	9.15e-01	5.85-01	1.76e-01	3.47e-03

Вывод: Наблюдается уменьшение погрешности при уменьшении коэффициента разрядки по времени.

4.2.2 Неравномерная сетка по пространству

Сетка по пространству: $(r, z) \in [1, 7] \times [0, 1.5]$, количество разбиений по r и z равно 3. Сетка по времени равномерная: $t \in [1, 8]$, количество элементов сетки 7. Исследуем зависимость от коэффициента разрядки k_r.

Функция: $u(r, z, t) = r^2 + z + t^3$

Таблица 4: Расчет для неравномерной сетки по пространству

k_r	0.9	0.7	0.5	0.3	0.1
$\|\|q - q^*\|\|$	1.11e+00	1.09e+00	1.08e+00	1.07e+00	1.11e+00

Вывод: Сначала наблюдается уменьшение погрешности при уменьшении коэффициента разрядки по времени, затем увеличение. Это может быть связано с тем, что с одного края сетки элементы становятся меньше и там увеличивается точность, а с другой стороны элементы увеличиваются и точность уменьшается.

4.3 Порядок сходимости

4.3.1 Порядок сходимости по пространству

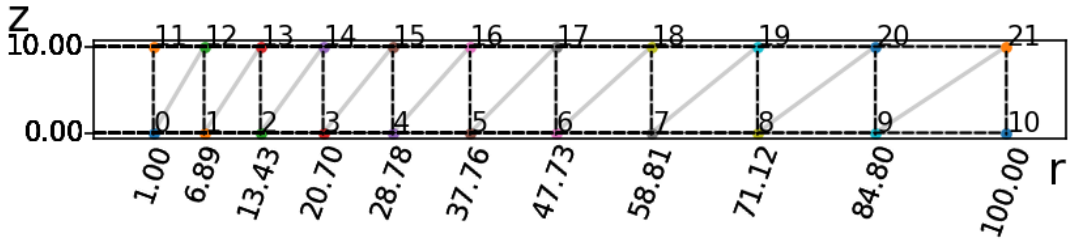

Рис. 2: Первые краевые заданы на правой границе, вторые краевые заданы на левой границе

Задача:

$$-\operatorname{div}(\sum_{m=1}^{2} K\frac{\kappa^m}{\mu^m}\operatorname{grad} P) + \sigma\frac{\partial P}{\partial t} = 0, \sum_{m=1}^{2} K\frac{\kappa^m}{\mu^m} = 1 \tag{19}$$

$$P|_{S_1} = 0, \tag{20}$$

$$\left.\frac{dP}{dn}\right|_{S_2} = 1, \tag{21}$$

Аналитическое решение заданного уравнения имеет вид: $P = C_1 \ln(r) + C_2$. Для $C_1 = 1$ и $C_2 = 4.5$ приведено сравнение аналитического и численного решения на рис. 5.

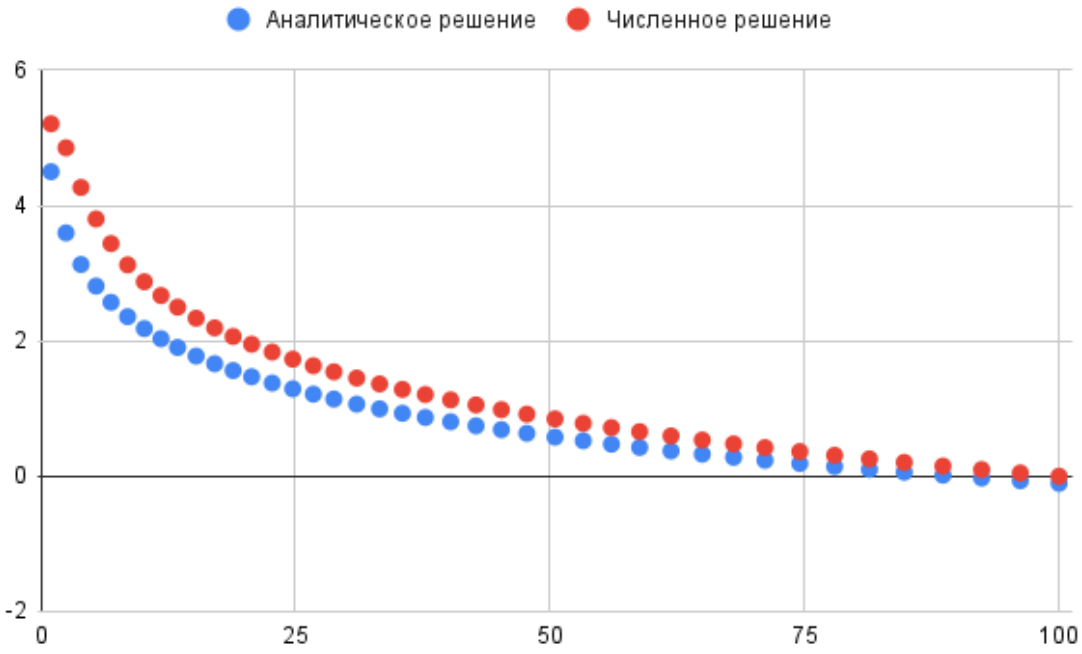

Рис. 3: Сравнение численного и аналитического решений

Для вычисления порядка сходимости посчитаем поле еще на двух сетках с уменьшенным шагом в 2 и 4 раза соответственно (рис. 6 и рис. 7).

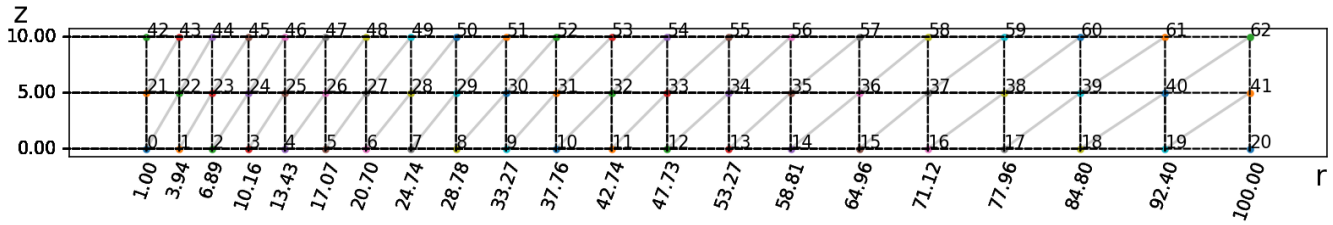

Рис. 4: Сетка с шагом $h/2$

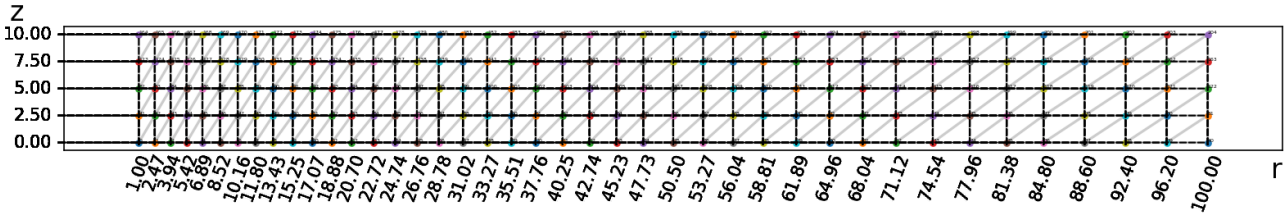

Рис. 5: Сетка с шагом $h/4$

Порядок сходимости вычислим по формуле:

$$2^p = \frac{||P_{h/2} - P_h||}{||P_{h/4} - P_{h/2}||} = 3,501796743 \tag{22}$$

Вывод: $p \approx 2$, что соответствует порядку сходимости для линейных базисных функций.

4.3.2 Порядок сходимости по времени на равномерной сетке

Исследование на функции $u = r^2 + z + t^4$, сетка по пространству равномерная: $(r, z) \in [1, 7] \times [0, 3]$, количество разбиений по r равно 2, по z равно 1. Сетка по времени равномерная: $t \in [1, 8]$, количество элементов сетки 7.

Вычислим относительную норму вектора погрешности значений функции в узлах конечноэлементной сетки по формуле

$$\delta q_h = \frac{\sqrt{\sum_{i=1}^{n}(q_i^h - q_i^*)^2}}{\sqrt{\sum_{i=1}^{n}(q_i^*)^2}} \tag{23}$$

В табл. 5 приведены относительные нормы погрешности при коэффициенте k - во сколько раз дробим сетку. Их отношения являются показателем скорости сходимости численного решения к точному.

Таблица 5: Порядок аппроксимации для равномерной сетки по времени

k	δq_h	$\delta q_h / \delta q_{h/2}$
1	0,003878134	
2	0,000666903	5,81514374
4	0,000203296	3,28044998

Вывод: порядок аппроксимации на равномерной сетке по времени ≈ 2

4.3.3 Порядок сходимости по времени на неравномерной сетке

Исследование на функции $u = r^2 + z + t^4$, сетка по пространству равномерная: $(r, z) \in [1, 7] \times [0, 3]$, количество разбиений по r равно 2, по z равно 1. Сетка по времени неравномерная: $t \in [1, 8]$, количество элементов сетки 7, $k_t = 0.5$.

Таблица 6: Порядок аппроксимации для неравномерной сетки по времени

k	δq_h	$\delta q_h / \delta q_{h/2}$
1	0,029122864	
2	0,007767533	3,749306687
4	0,001551152	5,007589475
8	0,000374921	4,137277591

Вывод: порядок аппроксимации на неравномерной сетке по времени ≈ 2

A Текст программы

Файл main.cpp

```
#include "mfe.h"
#include <iostream>

using namespace std;

int main()
{
  try
  {
    mfe m;

    m.buildPortraitOfMatrix();
    //m.assemblyGlobalMatrix();
    //m.toDense("matrix2.txt");
    ////m.bc_2();
    ////m.bc_3();
    ////m.toDense("matrix2.txt");
    //m.bc_1();
    //m.toDense("matrix2.txt");
    //m.MSG();
    //m.writeToFile(m.q);
    m.iterationProcess();
  }
  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 to write result to file!" << endl;
      break;
    }
  }

  return 0;
}
```

Файл mfe.h

```
#pragma once
#ifndef MFE_H
#define MFE_H
#include <vector>
#include <functional>
#include <fstream>
#include <algorithm>
#include <iomanip>

typedef std::vector <std::pair<double, double>> grid;
typedef std::vector<int> pvector;
typedef std::vector<double> mvector;
```

```
typedef std::vector<std::vector<double>> matrix;
typedef std::vector<std::vector<int>> finiteElem;
typedef std::function<double(double, double) > ScalFunc2D;

enum exceptions { BAD_READ, OUT_OF_AREA, BAD_WRITE };

class mfe
{
  struct material
  {
    double K;
    double Phi;
    double S2;
  };

  struct _bc
  {
    int n_i;
    int loc_node1_i;
    int loc_node2_i;
    double Tetta_i;
  };

public:
  mfe();
  void iterationProcess();
  //=========================================================================
  void buildPortraitOfMatrix();
  void buildLocalG(int ielem);
  void buildLocalG(int ielem, double r1, double r2, double r3, double z1, double
     z2, double z3, double detD);
  void buildLocalG(int ielem, double r[3], double z[3], double detD);
  void buildLocalF(int ielem, double r[3], double z[3], double detD,
    double dt, double t, mvector& q0, mvector& q1, mvector& q2);
  void buildLocalF(int ielem, double r[3], double z[3], double detD, double t,
   mvector& q0, mvector& q1, mvector& q2, double diffn1, double diffn2, double
   diffn3);
  void buildLocalF(int ielem, double dt, double t, double q0[3]);
  void buildLocalF(int ielem);
  void addElementToGlobal(int i, int j, double elem);

  double calcSum(int ielem);
  void assemblyGlobalMatrix();
  void assemblyGlobalMatrix(double t, double dt, std::vector<double>& q_);
  void assemblyGlobalMatrix(double t, double t1, double t2, double t3,
    std::vector<double>& q_0, std::vector<double>& q_1, std::vector<double>& q_2
   );
  void toDense(const std::string _dense);

  double rightPart(int field, double r, double z);
  double rightPart(int field, double r, double z, double t);
  double Lambda(int field);
  double u_beta(double r, double z);
  double u_t(double r, double z, double t);
  double sigma(int field);

  void make_bc_1(double t);
  void bc_1();
  void bc_2();
  void bc_3();
```

```
69
70    //=========================================================================
71    void mult(mvector& x, mvector& y);
72    void LOS();
73    void MSG( );
74    void writeToFile(mvector& q);
75    void writeToFile(mvector& q, double t);
76    double EuclideanNorm(mvector& x);
77
78  public:
79    matrix G; // Матрица жесткости
80    matrix alfa;
81    matrix c;
82    matrix M;
83
84    mvector q;  // текущее решение
85    mvector q0; //предыдущее решение на t-1
86    mvector q1; //решение на t-2
87    mvector q2; //решение на t-3
88    mvector p; // следующий вектор итерации
89    mvector p0;//текущий вектор итерации
90    mvector p_1;
91    mvector Au;
92
93    mvector b_loc;
94    mvector p_loc;
95    mvector F;  // Вектор правой части
96
97    std::vector<int> bc1nodes;
98    std::vector<double> b_2;
99    std::vector<double> b_3;
100   std::vector<double> ub;
101   matrix A3;
102
103   int Nuz;          // Размер сетки
104   grid MeshRZ;        // Сетка
105   std::vector<double> r_coord;
106   std::vector<double> z_coord;
107   int Rsize;
108   int Zsize;
109
110   int Nel;          // Количество КЭ
111   finiteElem FE;        // Конечные элементы
112
113   int Nph;          // Количество фаз
114   mvector Mu;         // Вязкости
115
116
117   std::vector<material> mats; // Материалы
118   std::vector<std::vector<int>> list;
119
120   int Nbc1;
121   std::vector <std::pair<int, double>> bc1;
122
123   int Nbc2;
124   int Nbc3;
125   std::vector<_bc> bc2;
126   std::vector<_bc> bc3;
127
128   // Глобальная матрица
```

```
  mvector di;
  mvector gg;
  mvector gu;
  pvector ig;
  pvector jg;

  matrix mat;

  bool isOrdered(const pvector& v);




  int maxIter;
  double eps;

  mvector time;
  int n;

  //для мсг
  mvector um;
  mvector r;
  mvector z;

};

inline mvector operator+( mvector& a, const mvector& b)
{
  mvector res = a;
  for (int i = 0; i < a.size(); i++)
    res[i] += b[i];
  return res;
}

inline mvector operator-( mvector& a, const mvector& b)
{
  mvector res = a;
  for (int i = 0; i < a.size(); i++)
    res[i] -= b[i];
  return res;
}

inline double operator*(const mvector& a, const mvector& b)
{
  double scalar = 0.0;
  for (int i = 0; i < a.size(); i++)
    scalar += a[i] * b[i];
  return scalar;
}

inline mvector operator*(double c,  mvector& a)
{
  for (int i = 0; i < a.size(); i++)
    a[i] *= c;
  return a;
}

#endif
```

Файл mfe.cpp

```
#include "mfe.h"
```

```cpp
#include <iostream>

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

// Конструктор: читаем сетку, КЭ с номерами их узлов, ые1- и ые2- краевые,
    материалы
// и свойства фаз, параметры для решения СЛАУ. Задаем точки и веса для
    интегрирования
mfe::mfe()
{
  {
    FILE* file;
    fopen_s(&file, "q.txt", "w");
    fclose(file);
  }
  //---------------------------------------------------------
  std::ifstream inCoord("coords.txt");
  if (inCoord.is_open())
  {
    inCoord >> Rsize;
    r_coord.resize(Rsize);
    for (int i = 0; i < Rsize; i++)
    {
      inCoord >> r_coord[i];
    }
    inCoord >> Zsize;
    z_coord.resize(Zsize);
    for (int i = 0; i < Zsize; i++)
    {
      inCoord >> z_coord[i];
    }
    inCoord.close();
  }
  else throw BAD_READ;

  //---------------------------------------------------------
  std::ifstream inGrid("node.txt");
  if (inGrid.is_open())
  {
    inGrid >> Nuz;
    MeshRZ.resize(Nuz);

    double x, y;
    for (int i = 0; i < Nuz; i++)
    {
      inGrid >> x >> y;
      MeshRZ[i] = std::make_pair(x, 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 (int i = 0; i < Nel; i++)
      FE[i].resize(3);

    int n1, n2, n3;
    for (int i = 0; i < Nel; i++)
    {
      inElem >> n1 >> n2 >> n3;
      FE[i] = { n1, n2, n3 };
    }

    inElem.close();
  }
  else throw BAD_READ;

  //---------------------------------------------------------
  std::ifstream inMat("mat.txt");
  if (inMat.is_open())
  {
    mats.resize(Nel);

    double K, Phi, S2;
    for (int i = 0; i < Nel; i++)
    {
      inMat >> K >> Phi >> S2;
      mats[i].K = K;
      mats[i].Phi = Phi;
      mats[i].S2 = S2;
    }
    inMat.close();
  }
  else throw BAD_READ;

  //---------------------------------------------------------
  std::ifstream inPhase("phaseprop.txt");
  if (inPhase.is_open())
  {
    inPhase >> Nph;
    Mu.resize(Nph);

    for (int i = 0; i < Nph; i++)
      inPhase >> Mu[i];

    inPhase.close();
  }
  else throw BAD_READ;


  //---------------------------------------------------------
  std::ifstream bc1File("bc1.txt");
  if (bc1File.is_open())
  {
    bc1File >> Nbc1;
    bc1.resize(Nbc1);
    for (int i = 0; i < Nbc1; i++)
    {
      int n_i;
      double value_bc1;

      bc1File >> n_i >> value_bc1;
      bc1[i] = std::make_pair(n_i, value_bc1);
```

```
    }
    bc1File.close();
  }
  else throw BAD_READ;

  //-------------------------------------------------------
  std::ifstream bc2File("bc2.txt");
  if (bc2File.is_open())
  {
    bc2File >> Nbc2;
    bc2.resize(Nbc2);

    int el_i, loc_node1_i, loc_node2_i;
    double Tetta_i;

    for (int i = 0; i < Nbc2; i++)
    {
      bc2File >> el_i >> loc_node1_i >> loc_node2_i >> Tetta_i;
      bc2[i] = { el_i, loc_node1_i, loc_node2_i, Tetta_i };
    }
    bc2File.close();
  }
  else throw BAD_READ;
  //-------------------------------------------------------
  std::ifstream bc3File("bc3.txt");
  if (bc3File.is_open())
  {
    bc3File >> Nbc3;
    bc3.resize(Nbc3);

    int el_i, loc_node1_i, loc_node2_i;
    double Betta_i;

    for (int i = 0; i < Nbc3; i++)
    {
      bc3File >> el_i >> loc_node1_i >> loc_node2_i >> Betta_i;
      bc3[i] = { el_i, loc_node1_i, loc_node2_i, Betta_i };
    }
    bc3File.close();
  }
  else throw BAD_READ;
  //-------------------------------------------------------
  maxIter = eps = 0;
  std::ifstream slau("kuslau.txt");
  if (slau.is_open())
  {
    slau >> maxIter >> eps;
    slau.close();
  }
  else throw BAD_READ;

  //-------------------------------------------------------
  std::ifstream inTime("time.txt");
  if (inTime.is_open())
  {
    inTime >> n;
    time.resize(n);
    for (int i = 0; i < n; i++)
    {
      inTime >> time[i];
```

```
    }
    inTime.close();
  }
  else throw BAD_READ;
  //---------------------------------------------------
  b_loc.resize(3);
  p_loc.resize(3);


  F.resize(Nuz);

  c.resize(3);
  M.resize(3);
  alfa.resize(3);
  G.resize(3);
  for (int i = 0; i < 3; i++) {
    c[i].resize(3);
    M[i].resize(3);
    alfa[i].resize(3);
    G[i].resize(3);
  }


  list.resize(Nuz);
  ig.resize(Nuz + 1);
  di.resize(Nuz);
  mat.resize(Nuz);
  for (int i = 0; i < mat.size(); i++)
    mat[i].resize(Nuz, 0);
  bc1nodes.resize(Nuz, -1);

  b_2.resize(2);
  b_3.resize(2);
  ub.resize(2);

  A3.resize(2);
  A3[0].resize(2);
  A3[1].resize(2);

  um.resize(Nuz);
  z.resize(Nuz);
  r.resize(Nuz);
  q.resize(Nuz);
  q0.resize(Nuz);
  q1.resize(Nuz);
  q2.resize(Nuz);
}
double mfe::Lambda(int field)                 {return 1;}
double mfe::rightPart(int field, double r, double z, double t)  {return -4 *
    Lambda(field) + 4*t*t*t*sigma(field); }
double mfe::u_beta(double r, double z)              {return 20 * z - 27;}
double mfe::u_t(double r, double z, double t)         { return r * r + z + t * t
    * t * t; }
double mfe::sigma(int field)                  {return 1;}



// Построить локальную матрицу жесткости*
void mfe::buildLocalG(int ielem, double r[3], double z[3], double detD)
{
```

```
  double r1 = r[0];
  double r2 = r[1];
  double r3 = r[2];

  double z1 = z[0];
  double z2 = z[1];
  double z3 = z[2];

  alfa[0] = { r2 * z3 - r3 * z2,    z2 - z3,    r3 - r2 };
  alfa[1] = { r3 * z1 - r1 * z3,    z3 - z1,    r1 - r3 };
  alfa[2] = { r1 * z2 - r2 * z1,    z1 - z2,    r2 - r1 };

  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      alfa[i][j] /= detD;

  double L = Lambda(ielem);
  double multix = L * abs(detD) * (r1 + r2 + r3) / 6.;
  //double multix = L * abs(detD) /2;

  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      G[i][j] = multix * (alfa[i][1] * alfa[j][1] + alfa[i][2] * alfa[j][2]);

}
// Построить локальный вектор правой части*
void mfe::buildLocalF(int ielem, double r[3], double z[3], double detD, double t
    , mvector& q0, mvector& q1, mvector& q2, double diffn1, double diffn2, double
     diffn3)
{
  double r1 = r[0];
  double r2 = r[1];
  double r3 = r[2];

  double z1 = z[0];
  double z2 = z[1];
  double z3 = z[2];

  p_loc[0] = rightPart(ielem, r1, z1, t);
  p_loc[1] = rightPart(ielem, r2, z2, t);
  p_loc[2] = rightPart(ielem, r3, z3, t);
  double sig = sigma(ielem);

  c[0] = { 6 * r1 + 2 * r2 + 2 * r3,    2 * r1 + 2 * r2 + r3,   2 * r1 + r2 + 2
    * r3 };
  c[1] = { 2 * r1 + 2 * r2 + r3,      2 * r1 + 6 * r2 + 2 * r3,   r1 + 2 * r2 +
    2 * r3 };
  c[2] = { 2 * r1 + r2 + 2 * r3,      r1 + 2 * r2 + 2 * r3,   2 * r1 + 2 * r2 +
    6 * r3 };

  double mult = abs(detD) / 120;

  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      c[i][j] *= mult;

  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      M[i][j] = sig * c[i][j];

```

```
291   b_loc[0] = c[0] * p_loc - diffn1 * (M[0] * q0) - diffn2 * (M[0] * q1) - diffn3
        * (M[0] * q2);
292   b_loc[1] = c[1] * p_loc - diffn1 * (M[1] * q0) - diffn2 * (M[1] * q1) - diffn3
        * (M[1] * q2);
293   b_loc[2] = c[2] * p_loc - diffn1 * (M[2] * q0) - diffn2 * (M[2] * q1) - diffn3
        * (M[2] * q2);
294
295 }
296
297 // Добавить элемент в глобальную матрицу
298 void mfe::addElementToGlobal(int i, int j, double elem)
299 {
300   if (i == j)
301   {
302     di[i] += elem;
303     return;
304   }
305   else
306   {
307     for (int ind = ig[i]; ind < ig[i + 1]; ind++)
308       if (jg[ind] == j)
309       {
310         gg[ind] += elem;
311         //gu[ind] += elem;
312         return;
313       }
314   }
315 }
316
317 // Посчитать сумму, которая стоит перед матрицей жесткости
318 double mfe::calcSum(int ielem)
319 {
320   double sum = 0.0;
321
322   if (Nph == 2)
323   {
324     sum = (1 - mats[ielem].S2) / Mu[0] + mats[ielem].S2 / Mu[1];
325     sum *= mats[ielem].K;
326   }
327   else
328   {
329     sum = mats[ielem].S2 / Mu[0];
330     sum *= mats[ielem].K;
331   }
332   return sum;
333 }
334
335 // Сборка глобальной матрицы
336 void mfe::assemblyGlobalMatrix(double t, double t1, double t2, double t3,
337   std::vector<double>& q_0, std::vector<double>& q_1, std::vector<double>& q_2)
338 {
339
340   mvector q0_(3, 0);
341   mvector q1_(3, 0);
342   mvector q2_(3, 0);
343   for (int ielem = 0; ielem < Nel; ielem++)
344   {
345     double r1 = MeshRZ[FE[ielem][0]].first;
346     double r2 = MeshRZ[FE[ielem][1]].first;
347     double r3 = MeshRZ[FE[ielem][2]].first;
```

```
348     double z1 = MeshRZ[FE[ielem][0]].second;
349     double z2 = MeshRZ[FE[ielem][1]].second;
350     double z3 = MeshRZ[FE[ielem][2]].second;
351 
352     double r[3] = { r1,r2,r3 };
353     double z[3] = { z1,z2,z3 };
354 
355     double detD = (r2 - r1) * (z3 - z1) - (r3 - r1) * (z2 - z1);
356 
357     //// Домножаем на коэффициент
358     //double sum = calcSum(ielem);
359     //for (int i = 0; i < G.size(); i++)
360     //for (int j = 0; j < G[i].size(); j++)
361     //    G[i][j] *= sum;
362 
363     int n1 = FE[ielem][0];
364     int n2 = FE[ielem][1];
365     int n3 = FE[ielem][2];
366 
367     q0_[0] = q_0[n1];
368     q0_[1] = q_0[n2];
369     q0_[2] = q_0[n3];
370 
371 
372     q1_[0] = q_1[n1];
373     q1_[1] = q_1[n2];
374     q1_[2] = q_1[n3];
375 
376     q2_[0] = q_2[n1];
377     q2_[1] = q_2[n2];
378     q2_[2] = q_2[n3];
379 
380     double diffn0 = (t2 * (t1 + t3 - 2 * t) + t1 * (t3 - 2 * t) + t * (3 * t - 2
     * t3)) / ((t - t3) * (t - t2) * (t - t1));
381     double diffn1 = (t2 * (t3 - t) + t * (t3 - 2 * t) + t * (3 * t - 2 * t3)) /
    ((t1 - t3) * (t1 - t2) * (t1 - t));
382     double diffn2 = (t1 * (t3 - t) + t * (t3 - 2 * t) + t * (3 * t - 2 * t3)) /
    ((t2 - t3) * (t2 - t1) * (t2 - t));
383     double diffn3 = (t2 * (t1 - t) - t1 * t + t * t) / ((t3 - t2) * (t3 - t1) *
    (t3 - t));
384 
385 
386     buildLocalG(ielem, r, z, detD);
387     buildLocalF(ielem, r, z, detD, t, q0_, q1_, q2_, diffn1, diffn2, diffn3);
388 
389     F[n1] += b_loc[0];
390     F[n2] += b_loc[1];
391     F[n3] += b_loc[2];
392     for (int i = 0; i < 3; i++)
393       for (int j = 0; j <= i; j++) {
394         int n1 = FE[ielem][i];
395         int n2 = FE[ielem][j];
396         addElementToGlobal(n1, n2, G[i][j] + diffn0 * M[i][j]);
397       }
398   }
399 }
400 
401 // Построить портрет глобальной матрицыправильно*
402 void mfe::buildPortraitOfMatrix()
403 {
```

```
404
405   list[0].push_back(0);
406
407   // Идем по всем КЭ
408   for (int ielem = 0; ielem < Nel; ielem++)
409   {
410     // Берем ую1- соответствующую элементу базисную функцию
411     for (int i = 0; i < FE[ielem].size() - 1; i++)
412       // Идем по всем остальным функциям, начиная со второй
413       for (int j = i + 1; j < FE[ielem].size(); j++)
414       {
415         // Нужно добавить первую функциюменьшую() в список ко всем
416         // функциям, относящимся к КЭ
417         // Поэтому определяем позицию, куда будем добавлять в( какой список)
418         int insertPos = FE[ielem][j];
419         // Берем сам элемент, который будем вставлять
420         int element = FE[ielem][i];
421
422         bool isIn = false;
423
424         // Проверим, есть ли уже этот элемент в списке
425         for (int k = 0; k < list[insertPos].size() && !isIn; k++)
426           if (element == list[insertPos][k])
427             isIn = true;
428
429         // Если он в списке не найден, то добавляем его
430         if (!isIn)
431           list[insertPos].push_back(element);
432       }
433   }
434
435   // Сортируем все получившиеся списки по( возрастанию номеров)
436   for (int i = 0; i < Nuz; i++)
437     if (!isOrdered(list[i]))
438       sort(list[i].begin(), list[i].end());
439   //-------------------------------------------------------------------
440   // Формируем массив ig
441
442
443   // ый1- и ой2- элементы всегда равны 1, но мы будем нумеровать с 0
444   ig[0] = 0;
445   ig[1] = 0;
446   for (int i = 1; i < list.size(); i++)
447     ig[i + 1] = ig[i] + list[i].size();
448
449   //-------------------------------------------------------------------
450   // Формируем массив jg
451   jg.resize(ig.back());
452
453   for (int i = 1, j = 0; i < Nuz; i++)
454   {
455     for (int k = 0; k < list[i].size(); k++)
456       jg[j++] = list[i][k];
457   }
458 }
459
460 // Проверка списка на упорядоченность по возрастанию
461 bool mfe::isOrdered(const pvector& v)
462 {
463   if(v.size() == 0)
```

```
464     return true;
465   for (int i = 0; i < v.size() - 1; i++)
466     if (v[i + 1] < v[i])
467       return false;
468   return true;
469 }
470 
471 // Перевод матрицы в плотный формат
472 void mfe::toDense(const std::string _dense)
473 {
474   for (int i = 0; i < mat.size(); i++)
475   {
476     mat[i][i] = di[i];
477     for (int j = ig[i]; j < ig[i + 1]; j++)
478     {
479       mat[i][jg[j]] = gg[j];
480       //mat[jg[j]][i] = gu[j];
481     }
482   }
483   std::ofstream dense(_dense);
484   dense.precision(5);
485   if (dense.is_open())
486   {
487     for (int i = 0; i < mat.size(); i++)
488     {
489       dense << std::left << std::setw(20) << F[i];
490       for (int j = 0; j <= i; j++)
491         dense << std::left << std::setw(10) << mat[i][j];
492 
493       dense << std::endl << std::endl;
494     }
495 
496   }
497 
498 }
499 
500 void mfe::make_bc_1(double t)
501 {
502   bc1.clear();
503   for (int i = 1; i <= z_coord.size(); i++) {
504 
505     bc1.push_back(  std::make_pair(   (i - 1) * r_coord.size(), u_t(r_coord[0],
    z_coord[i - 1], t)));
506     bc1.push_back(  std::make_pair(   i * r_coord.size() - 1,   u_t(r_coord.back
    (), z_coord[i - 1], t)));
507 
508   }
509 
510 }
511 
512 
513 // Учет первых краевых
514 void mfe::bc_1()
515 {
516   for (int i = 0; i < bc1.size(); i++)
517     bc1nodes[bc1[i].first] = i; // В узле задано краевое
518 
519   int k;
520   for (int i = 0; i < Nuz; i++)
521   {
```

```
    if (bc1nodes[i] != -1)
    {
      di[i] = 1.0;
      F[i] = bc1[bc1nodes[i]].second;
      for (int 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 (int 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;
        }
      }
    }
  }
}

// Учет вторых краевых

void mfe::bc_2()
{
  for (int i = 0; i < Nbc2; i++)
  {
    int el_i = bc2[i].n_i;
    int loc_node1_i = bc2[i].loc_node1_i;
    int loc_node2_i = bc2[i].loc_node2_i;
    double Tetta_i = bc2[i].Tetta_i;

    int ind_1 = FE[el_i][loc_node1_i];
    int ind_2 = FE[el_i][loc_node2_i];
    double r1 = MeshRZ[ind_1].first;
    double r2 = MeshRZ[ind_2].first;
    double z1 = MeshRZ[ind_1].second;
    double z2 = MeshRZ[ind_2].second;

    double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
    //b_2[0] = b_2[1] = hm * Tetta_i / 2;

    b_2[0] = (8 * r1 + 4 * r2) * hm * Tetta_i / 24;
    b_2[1] = (8 * r2 + 4 * r1) * hm * Tetta_i / 24 ;
    F[ind_1] += b_2[0];
    F[ind_2] += b_2[1];
  }
}
// Учет третьих краевых

void mfe::bc_3()
{
  for (int i = 0; i < Nbc3; i++)
  {
```

```
    int el_i = bc3[i].n_i;
    int loc_node1_i = bc3[i].loc_node1_i;
    int loc_node2_i = bc3[i].loc_node2_i;
    double Betta_i = bc3[i].Tetta_i;

    int ind_1 = FE[el_i][loc_node1_i];
    int ind_2 = FE[el_i][loc_node2_i];
    double r1 = MeshRZ[ind_1].first;
    double r2 = MeshRZ[ind_2].first;
    double z1 = MeshRZ[ind_1].second;
    double z2 = MeshRZ[ind_2].second;
    double hm = sqrt((r2 - r1) * (r2 - r1) + (z2 - z1) * (z2 - z1));
    double ub1 = u_beta(r1, z1);
    double ub2 = u_beta(r2, z2);
    ub[0] = ub1;
    ub[1] = ub2;



    double k = Betta_i * hm / 24.;
    A3[0] = { k * (6 * r1 + 2 * r2), k * 2 * (r1 + r2) };
    A3[1] = { k * 2 * (r1 + r2), k * (2 * r1 + 6 * r2) };

    b_3[0] = A3[0] * ub;
    b_3[1] = A3[1] * ub;

    addElementToGlobal(ind_1, ind_1, A3[0][0]);
    addElementToGlobal(ind_1, ind_2, A3[0][1]);
    addElementToGlobal(ind_2, ind_1, A3[1][0]);
    addElementToGlobal(ind_2, ind_2, A3[1][1]);
    F[ind_1] += b_3[0];
    F[ind_2] += b_3[1];

  }
}



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

// Решение СЛАУ
void mfe::mult(mvector& x, mvector& y) {

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


  for (int i = 0; i < Nuz; i++) {
    y[i] = di[i] * x[i];
    for (int k = ig[i]; k < ig[i + 1]; k++) {
      int j = jg[k];
      y[i] += gg[k] * x[j];
      y[j] += gg[k] * x[i];
    }
  }

}
```

```
640 double mfe::EuclideanNorm(mvector & x) {
641   double scalar = 0;
642   for (int i = 0; i < Nuz; i++)
643     scalar += x[i] * x[i];
644   scalar = sqrt(scalar);
645   return scalar;
646 }
647
648
649 // Метод сопряженных градиентов
650 void mfe::MSG() {
651
652   for (int i = 0; i < Nuz; i++)
653     um[i] = z[i] = r[i] = 0;
654
655   double scal1 = 0;
656   double scal2 = 0;
657   double scal3 = 0;
658   double alfa = 0;
659   double beta = 0;
660   int k = 0;
661   mult(q, um);
662   for (int i = 0; i < Nuz; i++)
663     r[i] = F[i] - um[i];
664   z = r;
665   double bnorm = EuclideanNorm(F);
666   double residual = EuclideanNorm(r) / bnorm;
667   if (residual > eps) {
668     scal1 = 0;
669     scal2 = 0;
670     scal3 = 0;
671     alfa = 0;
672     beta = 0;
673     mult(z, um);
674     for (int i = 0; i < Nuz; i++) {
675       scal1 += r[i] * r[i];
676       scal2 += um[i] * z[i];
677     }
678     alfa = scal1 / scal2;
679     for (int i = 0; i < Nuz; i++) {
680       q[i] += alfa * z[i];
681       r[i] -= alfa * um[i];
682     }
683     for (int i = 0; i < Nuz; i++)
684       scal3 += r[i] * r[i];
685     beta = scal3 / scal1;
686     for (int i = 0; i < Nuz; i++)
687       z[i] = r[i] + beta * z[i];
688     residual = EuclideanNorm(r) / bnorm;
689   }
690
691   for (k = 1; k < maxIter && residual>eps; k++) {
692     scal1 = scal3;
693     scal2 = 0;
694     scal3 = 0;
695     alfa = 0;
696     beta = 0;
697     mult(z, um);
698     for (int i = 0; i < Nuz; i++) {
699       scal2 += um[i] * z[i];
```

```
    }
    alfa = scal1 / scal2;
    for (int i = 0; i < Nuz; i++) {
      q[i] += alfa * z[i];
      r[i] -= alfa * um[i];
    }
    for (int i = 0; i < Nuz; i++)
      scal3 += r[i] * r[i];
    beta = scal3 / scal1;
    for (int i = 0; i < Nuz; i++)
      z[i] = r[i] + beta * z[i];
    residual = EuclideanNorm(r) / bnorm;

  }
}

// Записать результат в файл
void mfe::writeToFile(mvector& q, double t)
{


  FILE* File;
  fopen_s(&File, "q.txt", "a");


  if (File) {
    //fprintf(File, "%-20s%-20s%-20s%-20s%\n", "t", "q", "u", "|q-u|");
    fprintf(File, "%-20.2f%-20.5f%-20.5f%-20.10f%\n", t, q[0], u_t(r_coord[0],
   z_coord[0], t), abs(q[0] - u_t(r_coord[0], z_coord[0], t)));
    for (int i = 1; i < Nuz; i++) {
      int rn = i % Rsize;
      int zn = i / Rsize;
      fprintf(File, "%-20s%-20.5f%-20.5f%-20.10f%\n", " ", q[i], u_t(r_coord[rn
   ], z_coord[zn], t), abs(q[i] - u_t(r_coord[rn], z_coord[zn], t)));
    }
    mvector u(Nuz);
    for (int i = 0; i < Nuz; i++) {
      int rn = i % Rsize;
      int zn = i / Rsize;
      u[i]= u_t(r_coord[rn], z_coord[zn], t);
    }
    mvector diff = u - q;
    double e1 = EuclideanNorm(diff);
    double e2 = EuclideanNorm(u);
    //fprintf(File, "%-20e%\n", e1/e2);
    fclose(File);
  }
  else
    std::cout << "File q.txt is not opened!";
}


void mfe::iterationProcess() {

  for (int i = 0; i < Nuz; i++) {
    int rn = i % Rsize;
    int zn = i / Rsize;
    q2[i] = u_t(r_coord[rn],z_coord[zn], time[0]);//U(t0)
    q1[i] = u_t(r_coord[rn], z_coord[zn], time[1]);//U(t1)
    q0[i] = u_t(r_coord[rn], z_coord[zn], time[2]);//U(t2)
```

```
  }
  gg.resize(ig.back(), 0);

  for (int s = 3; s < n; s++)
  {
    for (int i = 0; i < gg.size(); i++)
      gg[i] = 0;
    for (int i = 0; i < Nuz; i++)
      di[i] = F[i] = 0;
    double t = time[s]; //tj
    double t1 = time[s-1]; //tj-1
    double t2 = time[s-2]; //tj-2
    double t3 = time[s-3]; //tj-3

    // q0 = q(s-1)
    // q = q(s)
    assemblyGlobalMatrix(t, t1, t2, t3, q0, q1, q2);
    toDense("matrix.txt");
    make_bc_1(t);
    bc_1();
    toDense("matrix1.txt");
    MSG();
    q2 = q1;
    q1 = q0;
    q0 = q;
    writeToFile(q, t);
  }
}
```