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»

Вариант №6

Группа	ПМ-83
Студент	УСТЬЯНЧИК ГЕОГРИЙ
Преподаватель	ПАТРУШЕВ ИЛЬЯ ИГОРЕВИЧ
Дата	27.12.2021

Новосибирск

2021

1. Задание:

МКЭ для одномерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции квадратичные и кубические (лагранжевы). Краевые условия всех типов. Коэффициент диффузии λ разложить по квадратичным базисным функциям. Матрицу СЛАУ генерировать в профильном формате. Для решения СЛАУ использовать LL^T –разложение.

2. Математическая модель

Для одномерной краевой задачи эллиптическое уравнение в декартовой системе координат выглядит так:

$$-\frac{d}{dx}\left(\lambda\frac{du}{dx}\right)+\gamma u=f$$

Расчетная область Ω в данном случае это отрезок $[a,b]$.

Краевые условия на его границах (левая x=a и правая x=b):

$u(a)=u_a$ или $u(b)=u_b$ - краевое условие первого рода

$-\lambda\frac{du}{dx}|(x=a)=\theta_a$ или $\lambda\frac{du}{dx}|(x=b)=\theta_b$ - краевое условие второго рода

$-\lambda\frac{du}{dx}|x=a+\beta_a\left(u(a)-u_{\beta_a}\right)=0$ или

$\lambda\frac{du}{dx}|x=b+\beta_b\left(u(b)-u_{\beta_b}\right)=0$ - краевое условие третьего рода

Функцию u будем искать в виде разложения по некоторым базисным функциям ψ_j с весами q_j:

$$u=\sum_{j=0}^{n}q_j\psi_j$$

Перебирая ψ_j, получаем систему линейных относительно весов q_j уравнений вида:

$$\sum_{j=1}^{n}\left[\int_\Omega\lambda\frac{d\psi_i}{dx}\frac{d\psi_j}{dx}d\Omega+\int_\Omega\gamma\psi_j\psi_i d\Omega+\int_{S_3}\beta\psi_j\psi_i dS\right]q_j$$

$$=\int_\Omega f\psi_i d\Omega+\int_{S_2}\theta\psi_i dS+\int_{S_3}\beta u_\beta\psi_i dS$$

По условию требуется, чтобы λ была разложена по базисным функциям, следовательно:

$$\lambda = \sum_{i=1}^{n} \lambda_i \psi_i$$

Учитывая это, глобальная матрица без учета краевых условий:

$$\sum_{j=1}^{n} q_j \left[\int_\Omega \left(\sum_{i=1}^{n} \lambda_i \psi_i \right) \frac{d\psi_i}{dx} \frac{d\psi_j}{dx} d\Omega + \int_\Omega \gamma \psi_j \psi_i d\Omega \right] = \int_\Omega f \psi_i d\Omega$$

Получаем СЛАУ относительно весов q_j:

$$Aq = F,$$

где A=G+M,

$$G = \int_\Omega \left(\sum_{i=1}^{n} \lambda_i \psi_i \right) \frac{d\psi_i}{dx} \frac{d\psi_j}{dx} d\Omega$$

- матрица жесткости, которая в дальнейшем будет разбита на сумму трех

$M = \int_\Omega \gamma \psi_j \psi_i d\Omega$ - матрица массы

$F = \int_\Omega f \psi_i d\Omega$ - правая часть

Решив СЛАУ $Aq = F$, найдем веса q_j искомой функции u.

3. Базисные функции

Квадратичные базисные функции:

Разобьем область Ω узлами $x_1, x_3, \ldots, x_{2n+1}$ на конечные элементы $\Omega_k = [x_{2k-1}, x_{2k+1}], k = \overline{1,n}$, $h_k = x_{2k+1} - x_{2k-1}$ - длина элемента Ω_k. Каждый элемент состоит из трех узлов (два граничных и один внутренний). Координата внутреннего узла: $x_{2k} = (x_{2k+1} + x_{2k-1})/2$.

Глобальные кусочно-квадратичные базисные функций строятся из шаблонных квадратичных базисных функций с помощью замены переменной $\xi = (x - x_{2k-1})/h_k$ и получаются:

$$\psi_1(\xi) = 2(\xi - 0.5)(\xi - 1)$$
$$\psi_2(\xi) = -4\xi(\xi - 1)$$
$$\psi_3(\xi) = 2\xi(\xi - 0.5)$$

Каждая базисная функция равна 1 только в одном узле и 0 в остальных.

Соответственно, получаем локальные матрицы жесткости, массы и локальный вектор правой части:

$$G_{ij} = G_{1_{ij}} + G_{2_{ij}} + G_{3_{ij}} =$$
$$= \int_0^1 \frac{\lambda_1}{h_k}\psi_1 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi + \int_0^1 \frac{\lambda_2}{h_k}\psi_2 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi + \int_0^1 \frac{\lambda_3}{h_k}\psi_3 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi$$

$$M_{ij} = \int_0^1 \gamma h_k \psi_i(\xi)\psi_j(\xi)d\xi$$

$$G_{1_{ij}} = \begin{pmatrix} \frac{37}{30} & -\frac{22}{15} & \frac{7}{30} \\ -\frac{22}{15} & \frac{8}{5} & -\frac{2}{15} \\ \frac{7}{30} & -\frac{2}{15} & -\frac{1}{10} \end{pmatrix}$$

$$G_{2_{ij}} = \begin{pmatrix} \frac{6}{5} & -\frac{16}{15} & -\frac{2}{15} \\ -\frac{16}{15} & \frac{32}{15} & -\frac{16}{15} \\ -\frac{2}{15} & -\frac{16}{15} & \frac{6}{5} \end{pmatrix} G_{3_{ij}} = \begin{pmatrix} -\frac{1}{10} & -\frac{2}{15} & \frac{7}{30} \\ -\frac{2}{15} & \frac{8}{5} & -\frac{22}{15} \\ \frac{7}{30} & -\frac{22}{15} & \frac{37}{30} \end{pmatrix}$$

$$M_{ij} = \begin{pmatrix} \frac{2}{15} & \frac{1}{15} & -\frac{1}{30} \\ \frac{1}{15} & \frac{8}{15} & \frac{1}{15} \\ -\frac{1}{30} & \frac{1}{15} & \frac{2}{15} \end{pmatrix}$$

$$F_i = (f_1 M_{i1} + f_2 M_{i2} + f_3 M_{i3})h_k$$

<u>Кубические базисные функции:</u>

Разобьем область Ω узлами $x_1, x_4, \dots, x_{3n+1}$ на конечные элементы $\Omega_k = [x_{3k-2}, x_{3k+1}], k = \overline{1,n},\ h_k = x_{3k+1} - x_{3k-2}$ - длина элемента Ω_k. Каждый элемент состоит из четырех узлов (два граничных и два внутренних). Координаты внутренних узлов: $x_{3k-1} = x_{3k-2} + \frac{h_k}{3}$ и $x_{3k} = x_{3k-2} + \frac{2h_k}{3}$. Каждая базисная функция равна 1 только в одном узле и 0 в остальных.

$$\psi_1(\xi) = -\frac{9}{2}\left(\xi - \frac{1}{3}\right)\left(\xi - \frac{2}{3}\right)(\xi - 1)$$

$$\psi_2(\xi) = \frac{27}{2}\xi\left(\xi - \frac{2}{3}\right)(\xi - 1)$$

$$\psi_3(\xi) = -\frac{27}{2}\xi\left(\xi - \frac{1}{3}\right)(\xi - 1)$$

$$\psi_4(\xi) = \frac{9}{2}\xi\left(\xi - \frac{1}{3}\right)\left(\xi - \frac{2}{3}\right)$$

Соответственно, получаем локальные матрицы жесткости, массы и локальный вектор правой части:

$$G_{ij} = G_{1_{ij}} + G_{2_{ij}} + G_{3_{ij}} =$$

$$= \int_0^1 \frac{\lambda_1}{h_k}\psi_1 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi + \int_0^1 \frac{\lambda_2}{h_k}\psi_2 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi + \int_0^1 \frac{\lambda_3}{h_k}\psi_3 \frac{d\psi_i}{d\xi}\frac{d\psi_j}{d\xi}d\xi$$

$$M_{ij} = \int_0^1 \gamma h_k \psi_i(\xi)\psi_j(\xi)d\xi$$

$$G_{1_{ij}} = \begin{pmatrix} \frac{2237}{840} & -\frac{1947}{560} & \frac{291}{280} & -\frac{379}{1680} \\ -\frac{1947}{560} & \frac{1377}{280} & -\frac{1053}{560} & \frac{123}{280} \\ \frac{291}{280} & -\frac{1053}{560} & \frac{243}{280} & -\frac{3}{112} \\ -\frac{379}{1680} & \frac{123}{280} & -\frac{3}{112} & -\frac{157}{840} \end{pmatrix}$$

$$G_{2_{ij}} = \begin{pmatrix} \frac{257}{210} & -\frac{171}{140} & -\frac{9}{70} & \frac{53}{420} \\ -\frac{171}{140} & \frac{351}{70} & -\frac{513}{140} & -\frac{9}{70} \\ -\frac{9}{70} & -\frac{513}{140} & \frac{351}{70} & -\frac{171}{140} \\ \frac{53}{420} & -\frac{9}{70} & -\frac{171}{140} & -\frac{257}{210} \end{pmatrix} G_{3_{ij}} = \begin{pmatrix} -\frac{157}{840} & -\frac{3}{112} & \frac{123}{280} & -\frac{379}{1680} \\ -\frac{3}{112} & \frac{243}{280} & -\frac{1053}{560} & \frac{291}{280} \\ \frac{123}{280} & -\frac{1053}{560} & \frac{1377}{280} & -\frac{1947}{560} \\ -\frac{379}{1680} & \frac{291}{280} & -\frac{1947}{560} & -\frac{2237}{840} \end{pmatrix}$$

$$M_{ij} = \begin{pmatrix} \frac{8}{105} & \frac{33}{560} & -\frac{3}{140} & \frac{19}{168} \\ \frac{33}{560} & \frac{27}{70} & -\frac{27}{560} & -\frac{27}{560} \\ -\frac{3}{140} & -\frac{27}{560} & -\frac{3}{140} & -\frac{27}{560} \\ \frac{19}{168} & -\frac{3}{140} & -\frac{27}{560} & \frac{8}{105} \end{pmatrix}$$

$$F_i = (f_1 M_{i1} + f_2 M_{i2} + f_3 M_{i3} + f_4 M_{i4})h_k$$

4. Краевые условия

Краевые условия первого рода:

Для учета краевых условий первого рода $u|_{S_1} = u_g$ в некотором граничном узле будем записывать в соответствующий элемент диагонали матрицы СЛАУ единицу, умноженную на большое число – 1e+30. В соответствующую компоненту вектора правой части – значение функции u_g в этом узле, умноженное на то же число.

Краевые условия второго рода:

Данное краевое условие вносит вклад только в правую часть СЛАУ, т.е. в соответствии с тем, в каком узле задано второе краевое условие, в правую часть к этому номеру добавляется θ_a или θ_b.

Краевые условия третьего рода:

Для учета краевых условий третьего рода $\lambda \left.\frac{\partial u}{\partial n}\right|_{S_3} + \beta\left(u|_{S_3} - u_\beta\right) = 0$ в некотором граничном узле добавим значение βu_β к соответствующему диагональному элементу матрицы СЛАУ и $\beta u|_{S_3}$ к соответствующему элементу вектора правой части.

5. Структура входных данных

На входе программа считывает информацию о количестве элементов, значении коэффициента γ и координатах узлов - файл in.txt.

Функции f, λ и тестируемая u задаются в самой программе. Также задаются все параметры для всех типов краевых условий: tetta1, tetta2, betta1, betta2,ubetta1,ubetta2 и соответственно наличие краевых условий.

6. Текст программы

main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>

#include "utils.h"

using namespace std;

int main()
{
   ifstream input;
   input.open("in.txt");
   int n;
   int m;
   double gamma;
   input >> n >> gamma;
   m = 2 * n + 1;

   vector<double> di(m);
   vector<int> ig(m + 1);
   vector<double> gg;
   vector<double> G(m);
   vector<double> F(m);

   vector<vector<double>> B(3, vector<double>(3));
   vector<vector<double>> C(3, vector<double>(3));

   vector<double> x;
   x.reserve(3 * n - 1);
   for (int i = 0; i <= n; i++)
   {
      double value;
      input >> value;
      x.push_back(value);
   }
   assembly(n, m, gamma, ig, di, gg, B, C, F, G, x);
   boundary(n, m, di, G, x);
   slau_llt(n, m, ig, di, gg, G);
   output(n, x, G);
   return 0;
       }
```

utils.h

```
#pragma once
#include <vector>
#include <fstream>

using namespace std;

double func(double x);

double lambda(double x);

void assembly(
   int n, int m,
   double gamma,
   vector<int>& ig,
   vector<double>& di,
   vector<double>& gg,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& G,
   vector<double>& x
);

void local_build(
   int n,
   int k,
   double gamma,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& x
);

void assembly2(
   int n, int m,
   double gamma,
   vector<int>& ig,
   vector<double>& di,
   vector<double>& gg,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& G,
   vector<double>& x
);

void local_build2(
   int n,
   int k,
   double gamma,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& x
);

void llt_decompose(
   int n,
   vector<double>& G,
   vector<double>& gg,
   vector<int>& ig,
   vector<double>& di
```

```
);

void forward_prop(
    int n,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
);

void backward_prop(
    int n,
    int m,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
);

void slau_llt(
    int n, int m,
    vector<int>& ig,
    vector<double>& di,
    vector<double>& gg,
    vector<double>& G
);

void llt_decompose2(
    int n,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
);

void gauss(
    int n,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
);

void slau_llt2(
    int n, int m,
    vector<int>& ig,
    vector<double>& di,
    vector<double>& gg,
    vector<double>& G
);

void boundary(
    int n,
    int m,
    vector<double>& di,
    vector<double>& G,
    vector<double>& x
    );

void output(
    int n,
    vector<double>& x,
    vector<double>& G
    );
```

```
void output2(
   int n,
   vector<double>& x,
   vector<double>& G
       );
```

utils.cpp

```
#include "utils.h"

double func(double x)
{
   //return -4;
   //return x * x;
   //return -6 * x + x * x;
   //return -2 + x;
   //return -12 * x * x + x * x * x;
   //return -8 * x * x + x * x;
   //return -10 * x * x * x + x * x;
}

double lambda(double x)
{
   //return 1;
   //return 0;
   //return x;
   //return x * x;
   //return x * x * x;
}

double u(double x)
{
   //return x * x;
   //return x;
   //return x * x * x;
}

void assembly(
   int n, int m,
   double gamma,
   vector<int>& ig,
   vector<double>& di,
   vector<double>& gg,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& G,
   vector<double>& x
)
{
   //профиль
   ig[0] = 1;
   ig[1] = 1;
   for (int i = 2; i < m + 1; i++)
      ig[i] = ig[i - 1] + i - 1;
   gg.resize(ig[m] - 1);
   for (int i = 0; i < m; i++)
   {
      G[i] = 0.0;
      di[i] = 0.0;
   }

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

```
   {
      local_build(n, i - 1, gamma, B, C, F, x);
      for (int j = 0; j < 3; j++)
      {
         di[2 * (i - 1) + j] += B[j][j] + gamma * C[j][j];
         G[2 * (i - 1) + j] += F[j];
      }
      gg[ig[2 * i - 1] - 1] = B[1][0] + gamma * C[1][0];
      gg[ig[2 * i] - 1] = B[2][0] + gamma * C[2][0];
      gg[ig[2 * i]] = B[2][1] + gamma * C[2][1];
   }
}

void local_build(
   int n,
   int k,
   double gamma,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& x
)
{
   vector<vector<double>> G1 = {{37. / 30, -22. / 15, 7. / 30}, {-22. / 15, 8. / 5, -2. /
15}, {7. / 30, -2. / 15, -1. / 10} };
   vector<vector<double>> G2 = { {6. / 5, -16. / 15, -2. / 15}, {-16. / 15, 32. / 15, -16. /
15}, {-2. / 15, -16. / 15, 6. / 5} };
   vector<vector<double>> G3 = { {-1. / 10, -2. / 15, 7. / 30}, {-2. / 15, 8. / 5, -22. /
15}, {7. / 30, -22. / 15, 37. / 30} };
   vector<vector<double>> M = { {2. / 15, 1. / 15, -1. / 30}, {1. / 15, 8. / 15, 1. / 15},
{-1. / 30, 1. / 15, 2. / 15} };
   double h = x[k + 1] - x[k];
   double xk = x[k];
   for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
      {
         B[i][j] = (lambda(x[k]) * G1[i][j] + lambda(x[k] + x[k + 1] / 2.) * G2[i][j] +
lambda(x[k + 1]) * G3[i][j]) / h;
         C[i][j] = M[i][j] * h;
      }
   vector<double> tmp(3);
   tmp[0] = func(x[k]);
   tmp[1] = func(((x[k] + x[k + 1]) / 2.));
   tmp[2] = func(x[k + 1]);
   for (int i = 0; i < 3; i++)
   {
      F[i] = 0;
      for (int j = 0; j < 3; j++)
         F[i] += C[i][j] * tmp[j];
   }
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void assembly2(
   int n, int m,
   double gamma,
   vector<int>& ig,
   vector<double>& di,
   vector<double>& gg,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& G,
   vector<double>& x
```

```
)
{
   //профиль
   ig[0] = 1;
   ig[1] = 1;
   for (int i = 2; i < m + 1; i++)
      ig[i] = ig[i - 1] + i - 1;
   gg.resize(ig[m] - 1);
   for (int i = 0; i < m; i++)
   {
      G[i] = 0.0;
      di[i] = 0.0;
   }

   for (int i = 1; i <= n; i++)
   {
      local_build2(n, i - 1, gamma, B, C, F, x);
      for (int j = 0; j < 4; j++)
      {
         di[2 * (i - 1) + j] += B[j][j] + gamma * C[j][j];
         G[2 * (i - 1) + j] += F[j];
      }
      gg[ig[3 * (i - 1) + 2] - 2] = B[1][0] + gamma * C[1][0];
      gg[ig[3 * (i - 1) + 2] - 1] = B[2][0] + gamma * C[2][0];
      gg[ig[3 * (i - 1) + 2]] = B[2][1] + gamma * C[2][1];
      gg[ig[3 * (i - 1) + 3] - 1] = B[3][0] + gamma * C[3][0];
      gg[ig[3 * (i - 1) + 3]] = B[3][1] + gamma * C[3][1];
      gg[ig[3 * (i - 1) + 3] + 1] = B[3][2] + gamma * C[3][2];
   }
}

void local_build2(
   int n,
   int k,
   double gamma,
   vector<vector<double>>& B,
   vector<vector<double>>& C,
   vector<double>& F,
   vector<double>& x
)
{
   vector<vector<double>> G1 = { {2237. / 840, -1947. / 560, 291. / 280, -379. / 1680}, {-
1947. / 560, 1377. / 280, -1053. / 560, 123. / 280}, {291. / 280, -1053. / 560, 243. / 280,
-3. / 112}, {-379. / 1680, 123. / 280, -3. / 112, -157. / 840} };
   vector<vector<double>> G2 = { {257. / 210, -171. / 140, -9. / 70, 53. / 420}, {-171. /
140, 351. / 70, -513. / 140, -9. / 70}, {-9. / 70, -513. / 140, 351. / 70, -171. / 140},
{53. / 420, -9. / 70, -171. / 140, 257. / 210} };
   vector<vector<double>> G3 = { {-157. / 840, -3. / 112, 123. / 280, -379. / 1680}, {-3. /
112, 243. / 280, -1053. / 560, 291. / 280}, {123. / 280, -1053. / 560, 1377. / 280, -1947. /
560}, {-379. / 1680, 291. / 280, -1947. / 560, 2237. / 840} };
   vector<vector<double>> M = { {8. / 105, 33. / 560, -3. / 140, 19. / 1680}, {33. / 560,
27. / 70, -27. / 560, -3. / 140}, {-3. / 140, -27. / 560, 27. / 70, 33. / 560}, {19. / 1680,
-3. / 140, 33. / 560, 8. / 105} };
   double h = x[k + 1] - x[k];
   double xk = x[k];
   for (int i = 0; i < 4; i++)
      for (int j = 0; j < 4; j++)
      {
         B[i][j] = (lambda(x[k]) * G1[i][j] + lambda(x[k] + x[k + 1] / 2.) * G2[i][j] +
lambda(x[k + 1]) * G3[i][j]) / h;
         C[i][j] = M[i][j] * h;
      }
   vector<double> tmp(4);
   tmp[0] = func(x[k]);
   tmp[1] = func(((x[k] + x[k + 1])/ 3));
```

```
    tmp[2] = func(2 * ((x[k] + x[k + 1]) / 3));
    tmp[3] = func(x[k + 1]);
    for (int i = 0; i < 4; i++)
    {
        F[i] = 0;
        for (int j = 0; j < 4; j++)
            F[i] += C[i][j] * tmp[j];
    }
}

void llt_decompose2(
    int n,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
)
{
    int i, j, k;
    double per;
    int a;
    int b;
    di[0] = sqrt(di[0]);
    for (i = 1; i < n; i++)
    {
        a = i - ig[i + 1] + ig[i];
        for (j = 0; j < ig[i + 1] - ig[i]; j++)
        {
            b = a + j - ig[a + j + 1] + ig[a + j];
            per = gg[ig[i] + j - 1];

            if (a < b)
            {
            for (k = ig[a + j + 1] - ig[a + j] - 1; k >= 0; k--)
            {
            per -= gg[ig[a + j] + k - 1] * gg[ig[i] + b - a + k - 1];
            }
            }
            else
            {
            for (k = a - b; k < ig[a + j + 1] - ig[a + j]; k++)
            {
            per -= gg[ig[i] + k - 1 - (a - b)] * gg[ig[a + j] + k - 1];
            }
            }
            gg[ig[i] + j - 1] = per / di[a + j];
        }
        per = di[i];
        for (k = 0; k < ig[i + 1] - ig[i]; k++)
            per -= gg[ig[i] + k - 1] * gg[ig[i] + k - 1];
        di[i] = sqrt(per);
    }
}

void gauss(
    int n,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di
)
{
    vector<double>q(n*4);
    int i, j;
    //прямой ход
```

```
   q[0] = G[0] / di[0];
   for (i = 1; i < n; i++)
   {
      q[i] = G[i];
      for (j = 0; j < ig[i + 1] - ig[i]; j++)
      {
         q[i] -= gg[ig[i] + j - 1] * q[i - ig[i + 1] + ig[i] + j];
      }
      q[i] = q[i] / di[i];
   }

   for (i = 0; i < n; i++)
      G[i] = q[i];
   //обратный ход
   for (i = n - 1; i >= 0; i--)
   {
      q[i] = G[i] / di[i];
      for (j = 0; j < ig[i + 1] - ig[i]; j++)
      {
         G[i - ig[i + 1] + ig[i] + j] -= q[i] * gg[ig[i] + j - 1];
      }
   }
}

void slau_llt2(
   int n, int m,
   vector<int>& ig,
   vector<double>& di,
   vector<double>& gg,
   vector<double>& G
)
{
   llt_decompose2(n, G, gg, ig, di);
   gauss(n, G, gg, ig, di);
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void llt_decompose(
   int n,
   vector<double>& G,
   vector<double>& gg,
   vector<int>& ig,
   vector<double>& di
)
{
   di[0] = sqrt(di[0]);
   for (size_t i = 1; i <= n; i++)
   {
      gg[ig[2 * i - 1] - 1] /= di[2 * i - 2];
      di[2 * i - 1] = sqrt(di[2 * i - 1] - (gg[ig[2 * i - 1] - 1] * gg[ig[2 * i - 1] - 1]));
      gg[ig[2 * i] - 1] /= di[2 * i - 2];
      gg[ig[2 * i]] = (gg[ig[2 * i]] - gg[ig[2 * i - 1] - 1] * gg[ig[2 * i] - 1]) / di[2 * i
- 1];
      di[2 * i] = sqrt(di[2 * i] - gg[ig[2 * i] - 1] * gg[ig[2 * i] - 1] - gg[ig[2 * i]] *
gg[ig[2 * i]]);
   }
}

void forward_prop(
   int n,
   vector<double>& G,
   vector<double>& gg,
```

```
    vector<int>& ig,
    vector<double>& di
)
{
    G[0] /= di[0];
    for (int i = 1; i <= n; i++)
    {
        G[2 * i - 1] = (G[2 * i - 1] - gg[ig[2 * i - 1] - 1] * G[2 * i - 2]) / di[2 * i - 1];
        G[2 * i] = (G[2 * i] - gg[ig[2 * i] - 1] * G[2 * i - 2] - gg[ig[2 * i]] * G[2 * i -
1]) / di[2 * i];
    }
}

void backward_prop(
    int n,
    int m,
    vector<double>& G,
    vector<double>& gg,
    vector<int>& ig,
    vector<double>& di)
{
    G[m - 1] /= di[m - 1];
    for (int i = n; i > 0; i--)
    {
        G[2 * i - 1] = (G[2 * i - 1] - gg[ig[2 * i]] * G[2 * i]) / di[2 * i - 1];
        G[2 * i - 2] = (G[2 * i - 2] - gg[ig[2 * i] - 1] * G[2 * i] - gg[ig[2 * i - 1] - 1] *
G[2 * i - 1]) / di[2 * i - 2];
    }
}

void slau_llt(
    int n, int m,
    vector<int>& ig,
    vector<double>& di,
    vector<double>& gg,
    vector<double>& G
)
{
    //разложение
    llt_decompose(n, G, gg, ig, di);
    //прямой ход
    forward_prop(n, G, gg, ig, di);
    //обратный ход
    backward_prop(n, m, G, gg, ig, di);
}

void boundary(
    int n,
    int m,
    vector<double>& di,
    vector<double>& G,
    vector<double>& x
    )
{
    bool e1[2] = { 1,1 };
    bool e2[2] = { 0,0 };
    bool e3[2] = { 0,0 };
    double ubetta1 = 1.5;
    double ubetta2 = 5.0;
    double betta1 = 2.0;
    double betta2 = 2.0;
    double tetta1 = 2.0;
    double tetta2 = -1;
    if (e3[0] == 1)
    {
```

```
        di[0] += ubetta1 + betta1;
        G[0] += betta1 * u(x[0]);
    }
    if (e3[1] == 1)
    {
        di[m - 1] += ubetta2 + betta2;
        G[m - 1] += betta2 * u(x[n]);
    }
    if (e2[0] == 1)
    {
        G[0] += tetta1;
    }
    if (e2[1] == 1)
    {
        G[n - 1] += tetta2;
    }
    if (e1[0] == 1)
    {
        di[0] = 1.e+30;
        G[0] = 1.e+30 * u(x[0]);
    }
    if (e1[1] == 1)
    {
        di[m - 1] = 1.e+30;
        G[m - 1] = 1.e+30 * u(x[n]);
    }
}

void output(
    int n,
    vector<double>& x,
    vector<double>& G
    )
{
    ofstream output;
    output.open("Cou1.txt");
    output << x[0] << " " << u(x[0]) << " " << G[0] << endl;
    for (int i = 1; i <= n; i++)
    {
        double x2 = (x[i - 1] + x[i]) / 2.0;
        output << x2 << " " << u(x2) << " " << G[2 * i - 1] << endl;
        output << x[i] << " " << u(x[i]) << " " << G[2 * i] << endl;
    }
}

void output2(
    int n,
    vector<double>& x,
    vector<double>& G
)
{
    ofstream output;
    output.open("Cou1.txt");
    output << x[0] << " " << u(x[0]) << " " << G[0] << endl;
    for (int i = 1; i <= n; i++)
    {
        double x2 = (x[i - 1] + x[i]) / 3;
        double x3 = 2 * (x[i - 1] + x[i]) / 3;
        output << x2 << " " << u(x2) << " " << G[3 * i - 2] << endl;
        output << x3 << " " << u(x3) << " " << G[3 * i - 1] << endl;
        output << x[i] << " " << u(x[i]) << " " << G[3 * i] << endl;
    }
}
```

7. Тесты

- Цель: проверить вычисление матрицы жесткости
  Входные данные: $\lambda = 1\ \gamma = 0\ \mathrm{u} = \mathrm{x}^2\ \mathrm{f} = -4$
  $$\mathrm{x} = \{1,3\}$$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	4	3
3	9	9

  Вывод: Матрица жесткости вычисляется верно при λ=const

- Цель: проверить вычисление матрицы масс
  Входные данные: $\lambda = 0\ \gamma = 1\ \mathrm{u} = \mathrm{x}^2\ \mathrm{f} = \mathrm{x}^2$
  $$\mathrm{x} = \{1,3\}$$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	4	4
3	9	9

  Вывод: Матрица масс вычисляется верно

- Цель: проверить вычисление матрицы жесткости при λ≠const
  Входные данные: $\lambda = \mathrm{x}\ \gamma = 1\ \mathrm{u} = \mathrm{x}^2\ \mathrm{f} = -6\mathrm{x} + \mathrm{x}^2$
  $$\mathrm{x} = \{1,3\}$$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	4	3.30769
3	9	9

  Вывод: Матрица жесткости вычисляется верно при λ≠const

- Цель: Проверить сборку глобальной матрицы
  Входные данные: $\lambda = \mathrm{x}\ \gamma = 1\ \mathrm{u} = \mathrm{x}^2\ \mathrm{f} = -6\mathrm{x} + \mathrm{x}^2$
  $$\mathrm{x} = \{1,3,5,10\}$$

  Выходные данные:

xk	uk аналитическое	uk численное

1	1	1
2	4	2.07293
3	9	6.53047
4	16	12.5803
5	25	21.4961
7.5	56.25	51.5017
10	100	100

Вывод: глобальная матрица собирается верно

- Цель: проверить решение задачи если u – линейная функция
  Входные данные: $\lambda = x\ \gamma = 1\ u = x\ f = -2 + x$
  $x = \{1,3,5,10\}$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	2	1.56946
3	3	2.52354
4	4	3.45124
5	5	4.48364
7.5	7.5	7.06042
10	10	10

Вывод: при линейных функциях задача решается верно

- Цель: проверить учет вторых краевых условий
  Входные данные: $\lambda = x\ \gamma = 1\ u = x\ f = -2 + x$
  $x = \{1,3,5,10\}$

  Первые краевые условия слева, вторые - справа

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	2	1.21088
3	3	1.80637
4	4	2.58971
5	5	3.33432
7.5	7.5	4.78552
10	10	5.34976

Вывод: Вторые краевые условия справа считаются верно

- Цель: проверить учет вторых краевых условий слева
  Входные данные: $\lambda = x\ \gamma = 1\ u = x\ f = -2 + x$
  $x = \{1,3,5,10\}$

  Первые краевые условия справа, вторые - слева

Выходные данные:

xk	uk аналитическое	uk численное
1	1	2.55736
2	2	2.20654
3	3	2.83932
4	4	3.6327
5	5	4.59395
7.5	7.5	7.09363
10	10	10

Вывод: Вторые краевые условия слева считаются верно

- Цель: проверить учет третьих краевых условий
  Входные данные: $\lambda = x\ \gamma = 1\ u = x\ f = -2 + x$
  $x = \{1,3,5,10\}$
  Третьи краевые условия слева, вторые – справа
  Выходные данные:

xk	uk аналитическое	uk численное
1	1	0.611684
2	2	1.05149
3	3	1.72657
4	4	2.54283
5	5	3.30441
7.5	7.5	4.77216
10	10	5.33926

Вывод: третьи краевые условия учитываются верно

- Цель: проверить решение задачи если u – кубическая функция
  Входные данные: $\lambda = x\ \gamma = 1\ u = x^3\ f = -12x^2 + x^3$
  $x = \{1,3,5,10\}$

xk	uk аналитическое	uk численное
1	1	1
2	8	1.84471
3	27	18.9202
4	64	50.1938
5	125	110.852
7.5	421.875	379.972
10	1000	1000

Вывод: погрешность для полиномов третей степени больше, чем для второй.

- Цель: проверить решение задачи при дроблении сетки
  Входные данные: $\lambda = x\ \ \gamma = 1\ u = x^3\ f = -12x^2 + x^3$
  $x = \{1,2,3,4,5,7.5,10\}$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
1.5	3.375	1.43782
2	8	5.06553
2.5	15.625	11.231
3	27	22.0765
3.5	42.875	36.397
4	64	57.189
4.5	91.125	82.5281
5	125	116.177
6.25	244.141	229.422
7.5	421.875	415.809
8.75	669.922	656.281
10	1000	1000

  Вывод: при дроблении сетки результаты получаются точнее
- Цель: проверить решение задачи если λ квадратичная функция
  Входные данные: $\lambda = x^2\ \ \gamma = 1\ u = x^2\ f = -8x^2 + x^2$
  $x = \{1,3,5,10\}$

  Выходные данные:

xk	uk аналитическое	uk численное
1	1	1
2	4	7.98202
3	9	16.2416
4	16	22.4959
5	25	32.0273
7.5	56.25	55.6779
10	100	100

  Вывод: Задача решается верно

- Цель: проверить решение если λ кубическая функция
  Входные данные: $\lambda = x^3\ \ \gamma = 1\ u = x^2\ f = -10x^3 + x^3$
  $x = \{1,3,5,10\}$

  Выходные данные:

xk	uk аналитическое	uk численное

1	1	1
2	4	23.1403
3	9	36.8776
4	16	41.7498
5	25	49.9235
7.5	56.25	63.2861
10	100	100

Вывод: так как λ раскладывается по квадратичным базисным функциям решение имеет слишком большую погрешность.