Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Непрерывные математические модели

Лабораторная работа №1

Факультет	ПМИ
Группа	ПММ-42
Студенты	Александров М.Е.
	Жигалов П.С.
Преподаватели	Вагин Д.В.
	Персова М.Г.
Вариант	12

Новосибирск

2014

1. Цель работы

Разработать программу построения сглаживающего сплайна с использованием кусочно-полиномиальных эрмитовых базисных функций третьего порядка в одномерных, двумерных или трехмерных областях и опробовать ее при решении задач фильтрации для произвольных наборов зашумленных данных и при решении задач выдачи численного решения и его производных по набору весов конечноэлементного решения для определенного типа конечных элементов.

2. Задание

Решение задачи уточнения численного решения, полученного с использованием билинейных базисных функций на прямоугольных конечных элементах, в двумерной области в декартовых кординатах.

3. Теоретическая часть

3.1. Описание метода

Под задачей сглаживания понимают построение по заданному набору значений $\{\tilde{x}_k, \tilde{f}_k\}$ такой достаточно гладкой функции $f(x)$, значения которой в точках $\tilde{x}_k$ максимально близки к значениям $\tilde{f}_k$. Функцию $P(x)$, аппроксимирующую $f(x)$, находят из условия минимальности суммы квадратов отклонений значений $P(\tilde{x}_k)$ от $\tilde{f}_k$:

$$\sum_k \left(P(\tilde{x}_k) - \tilde{f}_k\right)^2 \to \min . \tag{1}$$

Записав $P(x)$ как

$$P(x) = \sum_l q_l \psi_l , \tag{2}$$

представим минимизируемый функционал в следующем виде:

$$F(\boldsymbol{q}) = \sum_k \omega_k \left(\sum_l q_l \psi_l(\tilde{x}_k) - \tilde{f}_k \right)^2 . \tag{3}$$

После преобразований получим СЛАУ вида $\boldsymbol{A}\boldsymbol{q} = \boldsymbol{b}$, определяющую коэффициенты q_i:

$$A_{i,j} = \sum_k \omega_k \psi_i(\tilde{x}_k) \psi_j(\tilde{x}_k) , \tag{4}$$

$$b_i = \sum_k \omega_k \psi_i(\tilde{x}_k) \tilde{f}_k . \tag{5}$$

Если ввести дополнительные регуляризирующие компоненты, (4) примет вид (6):

$$A_{i,j} = \sum_k \omega_k \psi_i(\tilde{x}_k) \psi_j(\tilde{x}_k) + \int_\Omega \alpha(\nabla \psi_i \cdot \nabla \psi_j) \, d\Omega + \int_\Omega \beta(\Delta \psi_i \Delta \psi_j) \, d\Omega . \tag{6}$$

3.2. Базисные функции

В качестве базиса будем использовать бикубические эрмитовы базисные функции. Рассмотрим одномерные эрмитовы локальные базисные функции. Они определяются следующим образом:

$$\begin{gathered}\hat{\varphi}_1(\xi)=1-3\xi^2+2\xi^3, \quad \hat{\varphi}_2(\xi)=\xi-2\xi^2+\xi^3, \\ \hat{\varphi}_3(\xi)=3\xi^2-2\xi^3, \qquad \hat{\varphi}_4(\xi)=-\xi^2+\xi^3\end{gathered} \tag{7}$$

где $\xi(x)=\dfrac{x-x_i}{h_i}$.

Пусть подобласть Ω_k имеет вид $\Omega_k=[x_p, x_{p+1}]\times[y_s, y_{s+1}]$. Обозначим $X_i=\hat{\varphi}_i$ - одномерные базисные функции на интервале $[x_p, x_{p+1}]$, а $Y_j=\hat{\varphi}_j$ - одномерные базисные функции на интервале $[y_s, y_{s+1}]$. Тогда бикубические базисные функции можно выразить следующим образом:

$$\begin{gathered}\hat{\psi}_i=X_{\mu(i)}Y_{\nu(i)}, \\ \mu(i)=2\left(\left[\frac{i-1}{4}\right]\mathrm{mod}\,2\right)+\left((i-1)\,\mathrm{mod}\,2\right)+1, \\ \nu(i)=2\left[\frac{i-1}{8}\right]+\left(\left[\frac{i-1}{2}\right]\mathrm{mod}\,2\right)+1.\end{gathered} \tag{8}$$

Связи и степени свободы получившихся базисных функций можно представить в виде таблицы:

	$\varphi_1, f(y_s)$	$\varphi_2, f'(y_s)$	$\varphi_3, f(y_{s+1})$	$\varphi_4, f'(y_{s+1})$
$\varphi_1, f(x_p)$	$\psi_1, f(x_p, y_s)$	$\psi_3, f'_y(x_p, y_s)$	$\psi_9, f(x_p, y_{s+1})$	$\psi_{11}, f'_y(x_p, y_{s+1})$
$\varphi_2, f'(x_p)$	$\psi_2, f'_x(x_p, y_s)$	$\psi_4, f''_{x,y}(x_p, y_s)$	$\psi_{10}, f'_x(x_p, y_{s+1})$	$\psi_{12}, f''_{x,y}(x_p, y_{s+1})$
$\varphi_3, f(x_{p+1})$	$\psi_5, f(x_{p+1}, y_s)$	$\psi_7, f'_y(x_{p+1}, y_s)$	$\psi_{13}, f(x_{p+1}, y_{s+1})$	$\psi_{15}, f'_y(x_{p+1}, y_{s+1})$
$\varphi_4, f'(x_{p+1})$	$\psi_6, f'_x(x_{p+1}, y_s)$	$\psi_8, f''_{x,y}(x_{p+1}, y_s)$	$\psi_{14}, f'_x(x_{p+1}, y_{s+1})$	$\psi_{16}, f''_{x,y}(x_{p+1}, y_{s+1})$

3.3. Численное интегрирование

Интегралы из формулы (6) будем считать численно с использованием формулы Гаусса:

$$\int_{\Omega_k} f(x, y)\, d\Omega_k=\sum_{i=1}^{m} f(x_i, y_i) w_i \tag{9}$$

где (x_i, y_i) – точки Гаусса, m – число точек Гаусса, w_i – соответствующие веса.

Выберем число точек равное 12. Точки Гаусса и соответствующие им веса для квадратного мастер-элементра приведены в таблице:

x	$-c$	c	0	0	$-a$	a	$-a$	a	$-b$	b	$-b$	b
y	0	0	$-c$	c	$-a$	$-a$	a	a	$-b$	$-b$	b	b
w	w_c	w_c	w_c	w_c	w_a	w_a	w_a	w_a	w_b	w_b	w_b	w_b

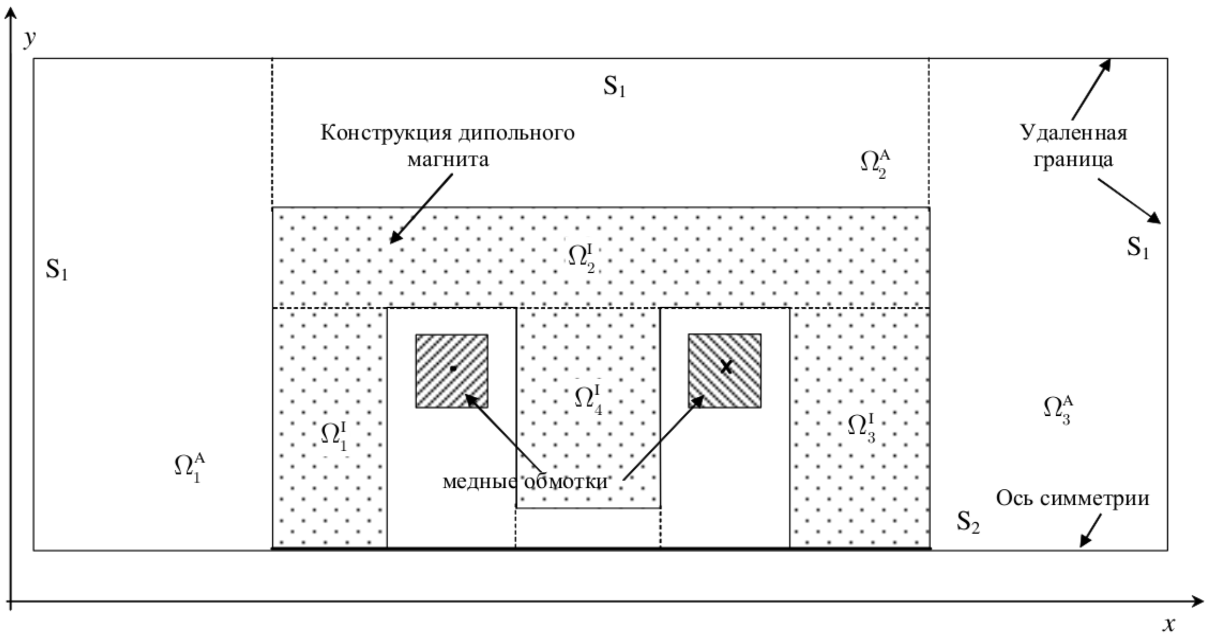
где:

$$a=\sqrt{\frac{114-3\sqrt{583}}{287}}\ ,\quad b=\sqrt{\frac{114+3\sqrt{583}}{287}}\ ,\quad c=\sqrt{\frac{6}{7}}\ , \tag{10}$$

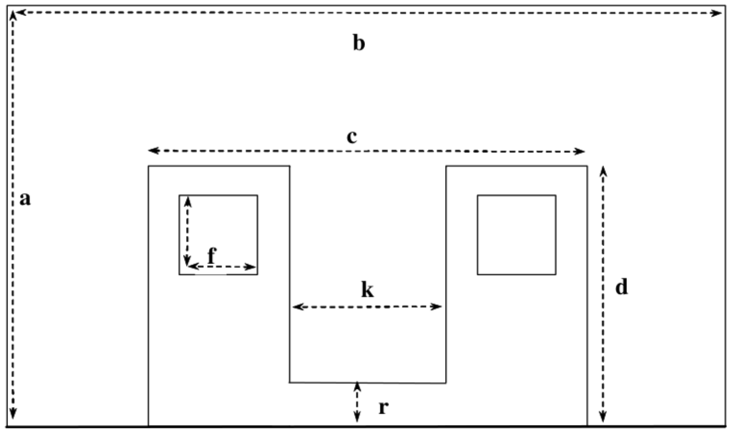
$$w_a=\frac{307}{810}+\frac{923}{270\sqrt{583}}\ ,\quad w_a=\frac{307}{810}-\frac{923}{270\sqrt{583}}\ ,\quad w_c=\frac{98}{405}\ . \tag{11}$$

4. Расчетная область

В качестве расчетной области была взята область из лабораторной работы №1 курса «Метод конечных элементов» (вариант 3).

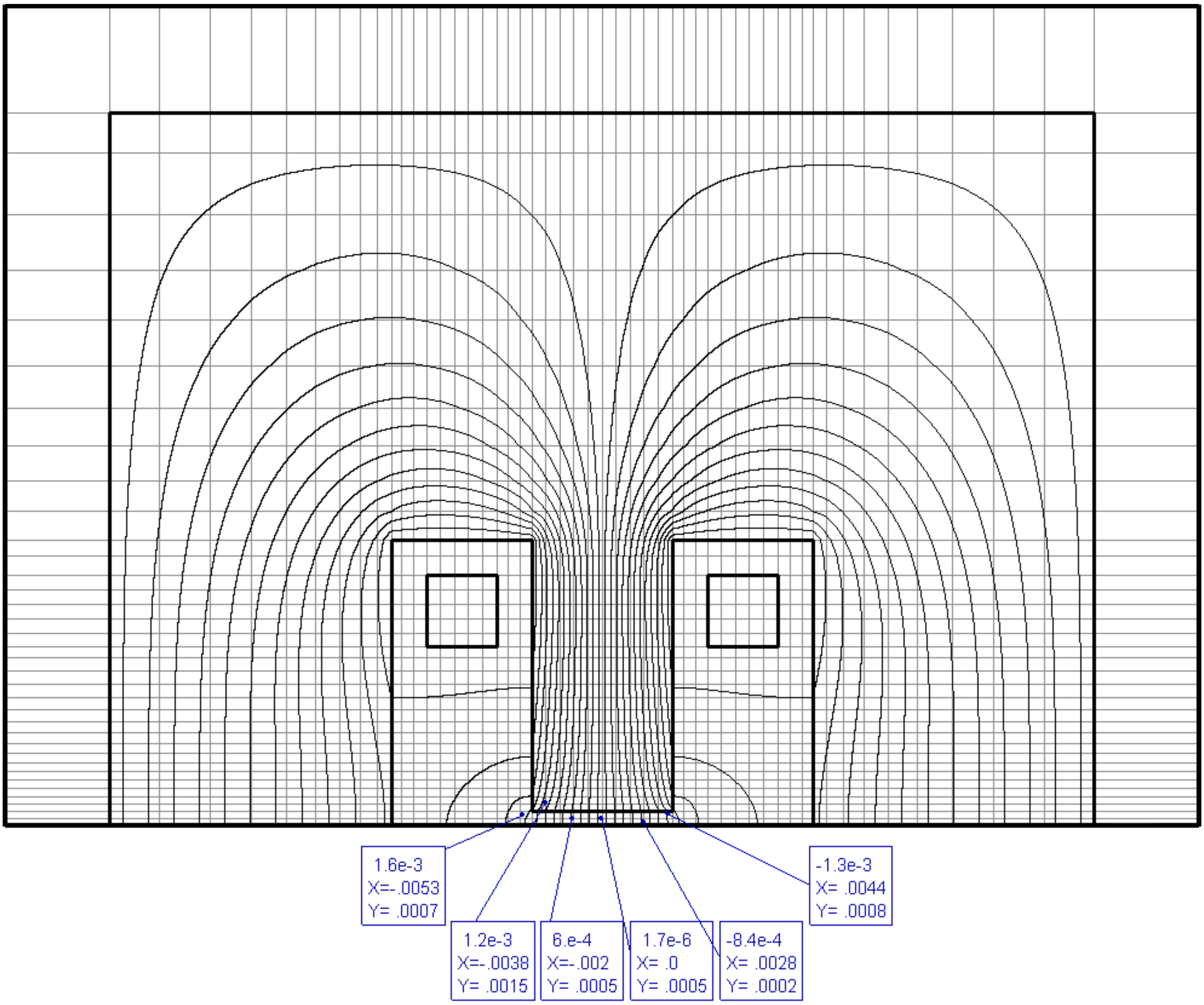


Расчетная область Ω



Конструкция магнита

Размеры, см	a	5
	b	7
	c	3
	d	2
	f	0.5
	k	1
	r	0.1
Знак тока в обмотке	лев	+
	прав	-



Конечноэлементная сетка, изолинии поля без сплайна и исследуемые точки

5. Исследования

5.1. Исследования без коэффициентов регуляризации

Оригинальная сетка:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,564715E-004	1,566888E-004	1,976702E-002	2,091769E-002
-3,80E-003	1,50E-003	1,205380E-004	1,209585E-004	3,337567E-002	3,399310E-002
-2,00E-003	5,00E-004	5,999387E-005	6,000305E-005	2,996506E-002	2,928134E-002
0,00E+000	5,00E-004	6,914272E-011	6,369216E-009	2,999259E-002	2,975191E-002
2,80E-003	2,00E-004	-8,396121E-005	-8,384162E-005	2,996507E-002	2,995843E-002
4,40E-003	8,00E-004	-1,327092E-004	-1,323876E-004	3,254244E-002	3,469488E-002

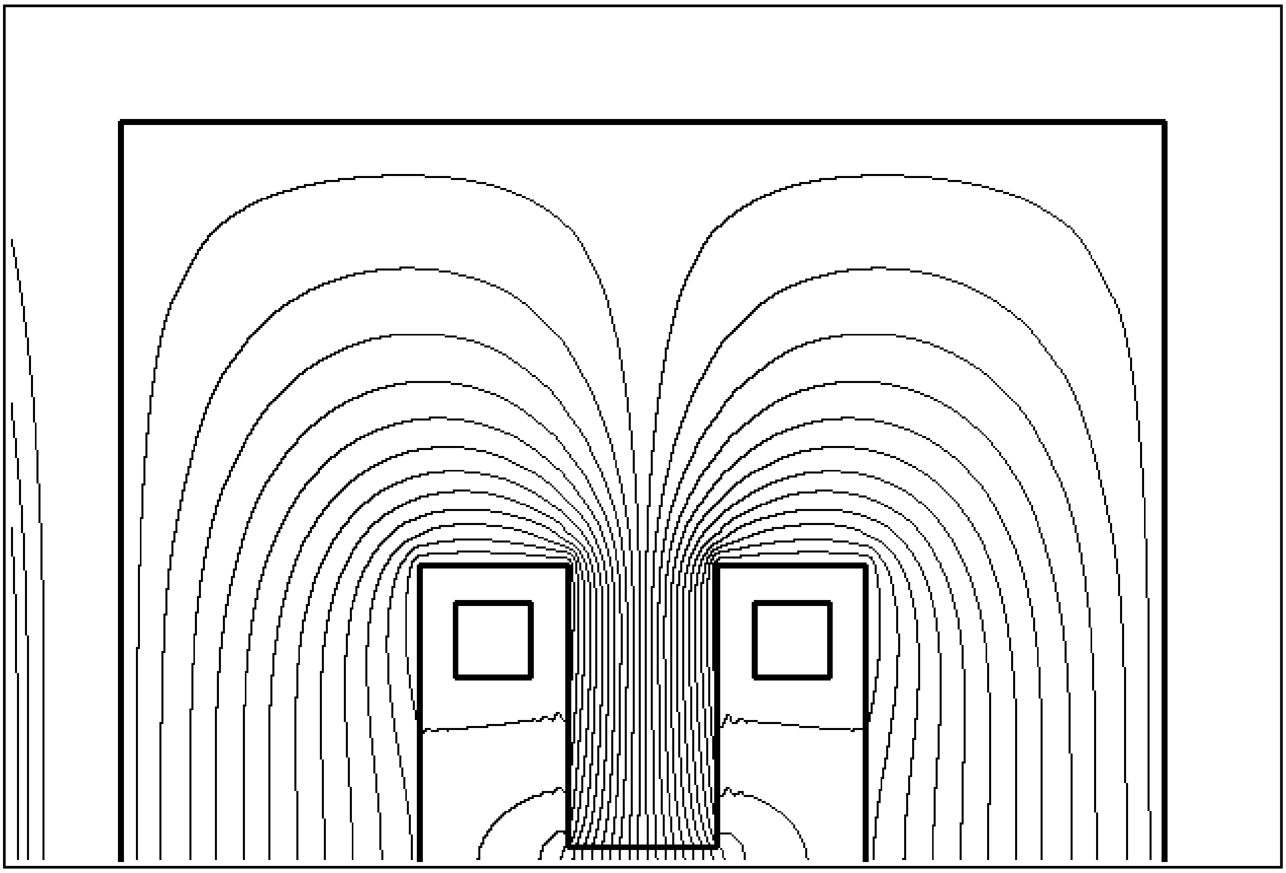
Сетка $h/2$:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,574366E-004	1,575570E-004	2,397251E-002	2,355989E-002
-3,80E-003	1,50E-003	1,210657E-004	1,210775E-004	3,570149E-002	3,623965E-002
-2,00E-003	5,00E-004	5,997765E-005	5,998550E-005	2,998323E-002	2,993730E-002
0,00E+000	5,00E-004	8,673460E-011	1,451712E-010	2,999026E-002	2,998968E-002
2,80E-003	2,00E-004	-8,396155E-005	-8,396185E-005	2,997681E-002	3,002321E-002
4,40E-003	8,00E-004	-1,322723E-004	-1,324690E-004	3,045944E-002	3,071000E-002

Сетка $h/4$:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,578879E-004	1,579095E-004	2,282396E-002	2,338520E-002
-3,80E-003	1,50E-003	1,211797E-004	1,211820E-004	3,710962E-002	3,684406E-002
-2,00E-003	5,00E-004	5,997495E-005	5,997583E-005	2,998287E-002	2,998333E-002
0,00E+000	5,00E-004	8,677125E-011	8,819823E-011	2,998908E-002	2,998935E-002
2,80E-003	2,00E-004	-8,395700E-005	-8,395647E-005	2,996952E-002	2,997369E-002
4,40E-003	8,00E-004	-1,322485E-004	-1,322602E-004	3,093034E-002	3,105699E-002

Так как построенный сплайн имеет непрерывные первые производные, в местах границ между материалами функция получилась сильно осциллирующей (заметно на рисунке невооруженным взглядом), поэтому точность вычисления производных весьма низкая.



5.2. Исследования с коэффициентами регуляризации

Экспериментальным путем были подобраны коэффициенты: $\alpha=1\cdot10^{-7}$, $\beta=1\cdot10^{-16}$.

Оригинальная сетка:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,564715E-004	1,567317E-004	1,976702E-002	2,081553E-002
-3,80E-003	1,50E-003	1,205380E-004	1,206978E-004	3,337567E-002	3,602354E-002
-2,00E-003	5,00E-004	5,999387E-005	6,001783E-005	2,996506E-002	2,977504E-002
0,00E+000	5,00E-004	6,914272E-011	-3,602770E-009	2,999259E-002	2,999662E-002
2,80E-003	2,00E-004	-8,396121E-005	-8,396947E-005	2,996507E-002	2,991535E-002
4,40E-003	8,00E-004	-1,327092E-004	-1,331619E-004	3,254244E-002	3,349946E-002

Сетка $h/2$:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,574366E-004	1,576054E-004	2,397251E-002	2,255212E-002
-3,80E-003	1,50E-003	1,210657E-004	1,210516E-004	3,570149E-002	3,665607E-002
-2,00E-003	5,00E-004	5,997765E-005	5,999785E-005	2,998323E-002	2,993390E-002
0,00E+000	5,00E-004	8,673460E-011	-3,937841E-010	2,999026E-002	3,000016E-002
2,80E-003	2,00E-004	-8,396155E-005	-8,396693E-005	2,997681E-002	2,997028E-002
4,40E-003	8,00E-004	-1,322723E-004	-1,327353E-004	3,045944E-002	3,173383E-002

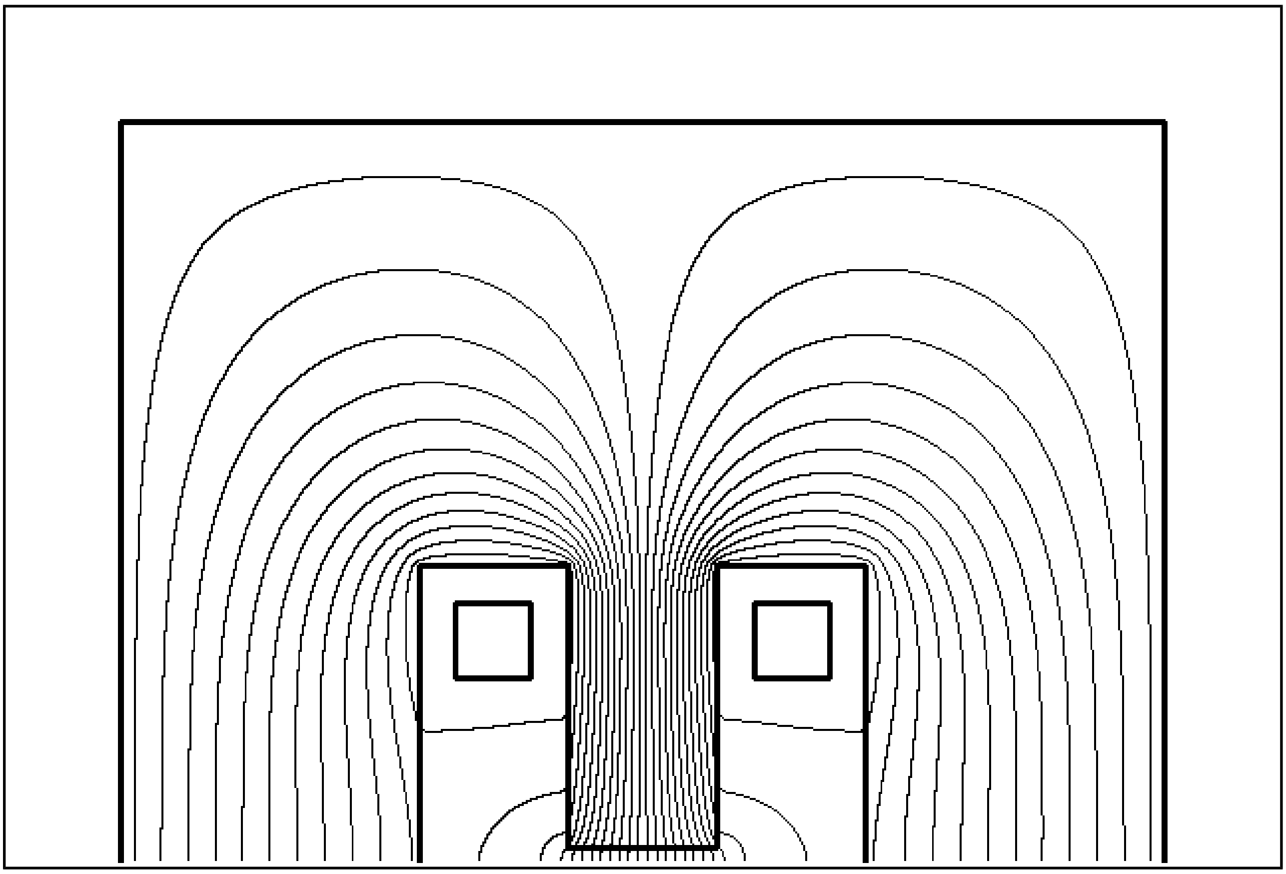
Сетка $h/4$:

x	y	A_z	A_z (spline)	$\lvert B\rvert$	$\lvert B\rvert$ (spline)
-5,30E-003	7,00E-004	1,578879E-004	1,579570E-004	2,282396E-002	2,287590E-002
-3,80E-003	1,50E-003	1,211797E-004	1,211941E-004	3,710962E-002	3,682049E-002
-2,00E-003	5,00E-004	5,997495E-005	5,999312E-005	2,998287E-002	2,997552E-002
0,00E+000	5,00E-004	8,677125E-011	5,878062E-011	2,998908E-002	2,999759E-002
2,80E-003	2,00E-004	-8,395700E-005	-8,396171E-005	2,996952E-002	2,997436E-002
4,40E-003	8,00E-004	-1,322485E-004	-1,327778E-004	3,093034E-002	3,151226E-002

При таких значениях параметров регуляризации уже наблюдается приемлемая точность производных:

$$\frac{\left\| |B|_{h/4} - |B|_h^{spline} \right\|}{\left\| |B|_{h/4} \right\|} = 1.51\cdot10^{-3}, \quad \frac{\left\| |B|_{h/4} - |B|_{h/2}^{spline} \right\|}{\left\| |B|_{h/4} \right\|} = 2.1\cdot10^{-4}.$$

Однако условие непрерывности по-прежнему соблюдается, поэтому значения производных вблизи областей влияния межматериальных границ по-прежнему не отличаются точностью (на рисунке можно обратить внимание на плавные переходы в местах разделения материалов):



6. Выводы

Применение сглаживающих сплайнов в задачах уточнения численного решения может быть оправдано в случаях, когда само решение имеет непрерывные производные, т. е. область должна быть однородной. Также однородной может быть не вся область, а только ее часть, в которой и будет строиться сплайн. Если не требуется особо высокой точности, то можно использовать сплайн и в неоднородной области, но при условии не особо сильной разницы между физическими параметрами материалов, и, возможно, придется подбирать значения регуляризирующих параметров.

7. Код программы (фрагменты, относящиеся к собственно сплайну)

Файл spline.h

```
#ifndef SPLINE_H_INCLUDED
#define SPLINE_H_INCLUDED

#include "../fem.h"

class fe_spline: public finite_element
{
public:
    node * nodes;
    // Значения двумерных эрмитовых базисных функций
    double phi(size_t func_n, double x, double y);
    double phi(size_t func_n, class node p);
    // Лапласcиан двумерных эрмитовых базисных функций
    double lap_phi(size_t func_n, double x, double y);
    double lap_phi(size_t func_n, class node p);
    // Инициализация эрмитовых КЭ из обычных
    void init(const finite_element & fe, node * n);
    // Скалярное произведение градиентов двумерных эрмитовых базисных функций
    double grad_phi(size_t bf1, size_t bf2, double x, double y);
    double grad_phi(size_t bf1, size_t bf2, node p);
    // Интеграл от скалярного произведения градиентов двумерных эрмитовых базисных функций
    double integrate_grad_phi(size_t bf1, size_t bf2);
    // Интеграл от лапласcиана двумерных эрмитовых базисных функций
    double integrate_lap_phi(size_t bf1, size_t bf2);
    // Получние номеров одномерных БФ из номера двумерной
    void two2one(size_t two, size_t & one1, size_t & one2);
    // Одномерные эрмитовы базисные функции
    double hermit_func(size_t func_n, char var, double x);
    // Первые производные одномерных эрмитовых базисных функций
    double hermit_func_first_der(size_t func_n, char var, double x);
    // Вторые производные одномерных эрмитовых базисных функций
    double hermit_func_second_der(size_t func_n, char var, double x);
    // Перевод в систему координат мастер-элемента
    node to_local(node p);
    // Перевод в глобальную систему координат
    node to_global(node p);
private:
    // Геометрия КЭ и якобиан
    double hx, hy, jacobian;
    // Веса Гаусса
    double gauss_weights[12];
    // Точки Гаусса
    node gauss_points[12];
};

// Сплайн
class FEM_spline: public FEM
{
public:
    // СЛАУ для сплайна
    class SLAE slae_spline;
    // Построение сплайна
    void make_spline();
    // Генерация портрета матрицы сплайна
    void generate_portrait_spline();
    // Коэффициенты регуляризации
    double alpha;
    double beta;
    // Получение решения (до применения сплайна)
    double get_solution_2(size_t fe_sol, node pnt);
    // Получение решения (после применения сплайна)
    double get_spline_solution(double x, double y);
    // Получение модуля градиента решения (после применения сплайна)
    double get_spline_b(double x, double y);
    // Рисовалки
    void draw(unsigned int width, unsigned int height, unsigned int num_isolines, bool need_grid);
    void draw(unsigned int width, unsigned int height, unsigned int num_isolines, bool need_grid, double x0, double y0, double x1,
double y1);
protected:
    // Эрмитовы КЭ
    fe_spline * fes_s;
    // Получение индекса в глобальной матрице
    size_t get_matrix_pos(unsigned int * nodes, size_t bf_num);
};

#endif // SPLINE_H_INCLUDED
```

Файл spline.cpp

```
#include "spline.h"

// Получение индекса в глобальной матрице
size_t FEM_spline::get_matrix_pos(unsigned int * nodes, size_t bf_num)
{
    bf_num--;
    size_t num1 = (size_t)(bf_num / 4);
    size_t num2 = bf_num - num1 * 4;
    return nodes[num1] * 4 + num2;
}
```

```
// Построение сплайна
void FEM_spline::make_spline()
{
    cout << "Making spline ..." << endl;
    fes_s = new fe_spline[fe_num];
    for(size_t i = 0; i < fe_num; i++)
        fes_s[i].init(finite_elements[i], nodes);

    generate_portrait_spline();

    // Узлы по которым берутся значения (в мастер-координатах)
    /*
    const size_t local_nodes_num = 4;
    const node local_nodes[local_nodes_num] =
    {
        node(0.0, 0.0),
        node(1.0, 0.0),
        node(0.0, 1.0),
        node(1.0, 1.0)
    };
    */
    /*
    const size_t local_nodes_num = 9;
    const node local_nodes[local_nodes_num] =
    {
        node(0.0, 0.0),
        node(0.5, 0.0),
        node(1.0, 0.0),
        node(0.0, 0.5),
        node(0.5, 0.5),
        node(1.0, 0.5),
        node(0.0, 1.0),
        node(0.5, 1.0),
        node(1.0, 1.0)
    };
    */
    const size_t local_nodes_num = 16;
    const node local_nodes[local_nodes_num] =
    {
        node(0.0, 0.0),
        node(1.0/3.0, 0.0),
        node(2.0/3.0, 0.0),
        node(1.0, 0.0),
        node(0.0, 1.0/3.0),
        node(1.0/3.0, 1.0/3.0),
        node(2.0/3.0, 1.0/3.0),
        node(1.0, 1.0/3.0),
        node(0.0, 2.0/3.0),
        node(1.0/3.0, 2.0/3.0),
        node(2.0/3.0, 2.0/3.0),
        node(1.0, 2.0/3.0),
        node(0.0, 1.0),
        node(1.0/3.0, 1.0),
        node(2.0/3.0, 1.0),
        node(1.0, 1.0)
    };

    // Цикл по КЭ
    for(size_t k = 0; k < fe_num; k++)
    {
        // Цикл по БФ 1
        for(size_t i = 1; i <= 16; i++)
        {
            size_t ii = get_matrix_pos(fes_s[k].node_n, i);
            // Цикл по БФ 2
            for(size_t j = 1; j < i; j++)
            {
                size_t jj = get_matrix_pos(fes_s[k].node_n, j);
                double sum = 0.0;
                // Цикл по точкам
                for(size_t m = 0; m < local_nodes_num; m++)
                {
                    node global_node = fes_s[k].to_global(local_nodes[m]);
                    sum += fes_s[k].phi(i, global_node) * fes_s[k].phi(j, global_node);
                }
                slae_spline.add(ii, jj, sum + alpha * fes_s[k].integrate_grad_phi(i, j) + beta * fes_s[k].integrate_lap_phi(i,
j));
            }
            double sum_di = 0.0, sum_rp = 0.0;
            // Цикл по точкам
            for(size_t m = 0; m < local_nodes_num; m++)
            {
                node global_node = fes_s[k].to_global(local_nodes[m]);
                sum_di += fes_s[k].phi(i, global_node) * fes_s[k].phi(i, global_node);
                sum_rp += fes_s[k].phi(i, global_node) * get_solution_2(k, global_node);
            }
            slae_spline.di[ii] += sum_di + alpha * fes_s[k].integrate_grad_phi(i, i) + beta * fes_s[k].integrate_lap_phi(i, i);
            slae_spline.f[ii] += sum_rp;
        }
    }

    slae_spline.solve();
}

// Генерация портрета матрицы сплайна
void FEM_spline::generate_portrait_spline()
{
    slae_spline.n = node_num * 4; // на 4 узла 16 бф
    set<size_t> * portrait = new set<size_t>[slae_spline.n];
```

```
    for(size_t k = 0; k < fe_num; k++)
    {
        for(size_t i = 0; i < 4; i++)
        {
            for(size_t j = 0; j < 4; j++)
            {
                size_t qi = fes_s[k].node_n[i];
                size_t qj = fes_s[k].node_n[j];
                if(qj > qi) swap(qi, qj);
                if(qi != qj)
                {
                    for(size_t m = 0; m < 4; m++)
                    {
                        portrait[qi * 4].insert(qj * 4 + m);
                        portrait[qi * 4 + 1].insert(qj * 4 + m);
                        portrait[qi * 4 + 2].insert(qj * 4 + m);
                        portrait[qi * 4 + 3].insert(qj * 4 + m);
                    }
                }
                else
                {
                    portrait[qi * 4 + 1].insert(qj * 4);
                    portrait[qi * 4 + 2].insert(qj * 4);
                    portrait[qi * 4 + 2].insert(qj * 4 + 1);
                    portrait[qi * 4 + 3].insert(qj * 4);
                    portrait[qi * 4 + 3].insert(qj * 4 + 1);
                    portrait[qi * 4 + 3].insert(qj * 4 + 2);
                }
            }
        }
    }

    size_t gg_size = 0;
    for(size_t i = 0; i < slae_spline.n; i++)
        gg_size += portrait[i].size();

    slae_spline.alloc_all(gg_size);

    slae_spline.ig[0] = 0;
    slae_spline.ig[1] = 0;
    size_t tmp = 0;
    for(size_t i = 0; i < slae_spline.n; i++)
    {
        for(set<size_t>::iterator j = portrait[i].begin(); j != portrait[i].end(); j++)
        {
            slae_spline.jg[tmp] = *j;
            tmp++;
        }
        slae_spline.ig[i + 1] = slae_spline.ig[i] + portrait[i].size();

        portrait[i].clear();
    }

    delete [] portrait;
}

// Получение решения (до применения сплайна)
double FEM_spline::get_solution_2(size_t fe_sol, node pnt)
{
    double x = pnt.x;
    double y = pnt.y;

    // Вычисление шага
    double hx = fabs(nodes[finite_elements[fe_sol].node_n[1]].x - nodes[finite_elements[fe_sol].node_n[0]].x);
    double hy = fabs(nodes[finite_elements[fe_sol].node_n[2]].y - nodes[finite_elements[fe_sol].node_n[0]].y);

    // Находим линейные одномерные функции
    double X1 = (nodes[finite_elements[fe_sol].node_n[1]].x - x) / hx;
    double X2 = (x - nodes[finite_elements[fe_sol].node_n[0]].x) / hx;
    double Y1 = (nodes[finite_elements[fe_sol].node_n[2]].y - y) / hy;
    double Y2 = (y - nodes[finite_elements[fe_sol].node_n[0]].y) / hy;

    // Находим значение билинейных базисных функций
    double psi[4];
    psi[0] = X1 * Y1;
    psi[1] = X2 * Y1;
    psi[2] = X1 * Y2;
    psi[3] = X2 * Y2;

    // Линейная комбинация базисных функций на веса
    double result = 0.0;
    for(unsigned int i = 0; i < 4; i++)
        result += slae.q[finite_elements[fe_sol].node_n[i]] * psi[i];

    return result;
}

// Получение решения (после применения сплайна)
double FEM_spline::get_spline_solution(double x, double y)
{
    // Определение КЭ, в который попала точка
    bool finded = false;
    unsigned int fe_sol = 0;
    for(unsigned int i = 0; i < fe_num && !finded; i++)
    {
        if(x >= nodes[finite_elements[i].node_n[0]].x && x <= nodes[finite_elements[i].node_n[1]].x &&
                y >= nodes[finite_elements[i].node_n[0]].y && y <= nodes[finite_elements[i].node_n[2]].y)
        {
            finded = true;
```

```
            fe_sol = i;
        }
    }

    // Если не нашли, значит точка за пределами области
    if(!finded)
    {
        cerr << "Error: Target point is outside area!" << endl;
        return 0.0;
    }

    // Если нашли, то решение будет линейной комбинацией базисных функций на соответствующие веса
    double result = 0.0;
    for(size_t i = 1; i <= 16; i++)
        result += fes_s[fe_sol].phi(i, node(x, y)) * slae_spline.q[get_matrix_pos(fes_s[fe_sol].node_n, i)];

    return result;
}

// Получение модуля градиента решения (после применения сплайна)
double FEM_spline::get_spline_b(double x, double y)
{
    // Определение КЭ, в который попала точка
    bool finded = false;
    unsigned int fe_sol = 0;
    for(unsigned int i = 0; i < fe_num && !finded; i++)
    {
        if(x >= nodes[finite_elements[i].node_n[0]].x && x <= nodes[finite_elements[i].node_n[1]].x &&
                y >= nodes[finite_elements[i].node_n[0]].y && y <= nodes[finite_elements[i].node_n[2]].y)
        {
            finded = true;
            fe_sol = i;
        }
    }

    // Если не нашли, значит точка за пределами области
    if(!finded)
    {
        cerr << "Error: Target point is outside area!" << endl;
        return 0.0;
    }

    // Если нашли, то решение будет линейной комбинацией базисных функций на соответствующие веса
    double grad_x = 0.0;
    double grad_y = 0.0;
    for(size_t i = 1; i <= 16; i++)
    {
        size_t ii = get_matrix_pos(fes_s[fe_sol].node_n, i);
        size_t b1, b2;
        fes_s[fe_sol].two2one(i, b1, b2);
        grad_x += fes_s[fe_sol].hermit_func_first_der(b1, 'x', x) * fes_s[fe_sol].hermit_func(b2, 'y', y) * slae_spline.q[ii];
        grad_y += fes_s[fe_sol].hermit_func_first_der(b2, 'y', y) * fes_s[fe_sol].hermit_func(b1, 'x', x) * slae_spline.q[ii];
    }

    return sqrt(grad_x * grad_x + grad_y * grad_y);
}
```

Файл hermit.cpp

```
#include "spline.h"

// Одномерные эрмитовы базисные функции
double fe_spline::hermit_func(size_t func_n, char var, double x)
{
    double x0 = nodes[node_n[0]].x;
    double y0 = nodes[node_n[0]].y;
    double hx = fabs(nodes[node_n[3]].x - nodes[node_n[0]].x);
    double hy = fabs(nodes[node_n[3]].y - nodes[node_n[0]].y);

    double ksi = 0.0;
    double h_var = 0.0;

    switch(var)
    {
    case 'x' :
        ksi = (x-x0)/hx;
        h_var = hx;
        break;
    case 'y' :
        ksi = (x-y0)/hy;
        h_var = hy;
        break;
    };

    switch(func_n)
    {
    case 1:
        return 1.0 - 3.0*ksi*ksi + 2.0*ksi*ksi*ksi;
        break;
    case 2:
        return h_var * (ksi - 2.0*ksi*ksi + ksi*ksi*ksi);
        break;
    case 3:
        return 3.0*ksi*ksi - 2.0*ksi*ksi*ksi;
        break;
    case 4:
        return h_var*(-ksi*ksi + ksi*ksi*ksi);
```

```
        break;
    };
    cerr << "Unknown number detected!" << endl;
    return 0.0;
}

// Первые производные одномерных эрмитовых базисных функций
double fe_spline::hermit_func_first_der(size_t func_n, char var, double x)
{
    double x0 = nodes[node_n[0]].x;
    double y0 = nodes[node_n[0]].y;
    double hx = fabs(nodes[node_n[3]].x - nodes[node_n[0]].x);
    double hy = fabs(nodes[node_n[3]].y - nodes[node_n[0]].y);

    double ksi = 0.0;
    double h_var = 0.0;

    switch(var)
    {
    case 'x' :
        ksi = (x-x0)/hx;
        h_var = hx;
        break;
    case 'y' :
        ksi = (x-y0)/hy;
        h_var = hy;
        break;
    };

    switch(func_n)
    {
    case 1:
        return (-6.0*ksi + 6.0*ksi*ksi)/h_var;
        break;
    case 2:
        return (1.0 - 4.0*ksi + 3.0*ksi*ksi);
        break;
    case 3:
        return (6.0*ksi - 6.0*ksi*ksi)/h_var;
        break;
    case 4:
        return (-2.0*ksi + 3.0*ksi*ksi);
        break;
    };
    cerr << "Unknown number detected!" << endl;
    return 0.0;
}

// Вторые производные одномерных эрмитовых базисных функций
double fe_spline::hermit_func_second_der(size_t func_n, char var, double x)
{
    double x0 = nodes[node_n[0]].x;
    double y0 = nodes[node_n[0]].y;
    double hx = fabs(nodes[node_n[3]].x - nodes[node_n[0]].x);
    double hy = fabs(nodes[node_n[3]].y - nodes[node_n[0]].y);

    double ksi = 0.0;
    double h_var = 0.0;

    switch(var)
    {
    case 'x' :
        ksi = (x-x0)/hx;
        h_var = hx;
        break;
    case 'y' :
        ksi = (x-y0)/hy;
        h_var = hy;
        break;
    };

    switch(func_n)
    {
    case 1:
        return (-6.0 + 12.0*ksi)/(h_var*h_var);
        break;
    case 2:
        return (-4.0 + 6.0*ksi)/h_var;
        break;
    case 3:
        return (6.0 - 12.0*ksi)/(h_var*h_var);
        break;
    case 4:
        return (-2.0 + 6.0*ksi)/h_var;
        break;
    };
    cerr << "Unknown number detected!" << endl;
    return 0.0;
}

// Значения двумерных эрмитовых базисных функций
double fe_spline::phi(size_t func_n, double x, double y)
{
    switch(func_n)
    {
    case 1 :
        return hermit_func(1, 'x', x) * hermit_func(1, 'y', y);
        break;
    case 2 :
        return hermit_func(2, 'x', x) * hermit_func(1, 'y', y);
```

```
        break;
    case 3 :
        return hermit_func(1, 'x', x) * hermit_func(2, 'y', y);
        break;
    case 4 :
        return hermit_func(2, 'x', x) * hermit_func(2, 'y', y);
        break;
    case 5 :
        return hermit_func(3, 'x', x) * hermit_func(1, 'y', y);
        break;
    case 6 :
        return hermit_func(4, 'x', x) * hermit_func(1, 'y', y);
        break;
    case 7 :
        return hermit_func(3, 'x', x) * hermit_func(2, 'y', y);
        break;
    case 8 :
        return hermit_func(4, 'x', x) * hermit_func(2, 'y', y);
        break;
    case 9 :
        return hermit_func(1, 'x', x) * hermit_func(3, 'y', y);
        break;
    case 10 :
        return hermit_func(2, 'x', x) * hermit_func(3, 'y', y);
        break;
    case 11 :
        return hermit_func(1, 'x', x) * hermit_func(4, 'y', y);
        break;
    case 12 :
        return hermit_func(2, 'x', x) * hermit_func(4, 'y', y);
        break;
    case 13 :
        return hermit_func(3, 'x', x) * hermit_func(3, 'y', y);
        break;
    case 14 :
        return hermit_func(4, 'x', x) * hermit_func(3, 'y', y);
        break;
    case 15 :
        return hermit_func(3, 'x', x) * hermit_func(4, 'y', y);
        break;
    case 16 :
        return hermit_func(4, 'x', x) * hermit_func(4, 'y', y);
        break;
    }
    cerr << "Unknown number detected!" << endl;
    return 0.0;
}

// Значения двумерных эрмитовых базисных функций
double fe_spline::phi(size_t func_n, class node p)
{
    return phi(func_n, p.x, p.y);
}

// Лапласиан двумерных эрмитовых базисных функций
double fe_spline::lap_phi(size_t func_n, double x, double y)
{
    switch(func_n)
    {
    case 1 :
        return hermit_func_second_der(1, 'x', x)
               * hermit_func(1, 'y', y)
               + hermit_func(1, 'x', x)
               * hermit_func_second_der(1, 'y', y);
        break;
    case 2 :
        return hermit_func_second_der(2, 'x', x)
               * hermit_func(1, 'y', y)
               + hermit_func(2, 'x', x)
               * hermit_func_second_der(1, 'y', y);
        break;
    case 3 :
        return hermit_func_second_der(1, 'x', x)
               * hermit_func(2, 'y', y)
               + hermit_func(1, 'x', x)
               * hermit_func_second_der(2, 'y', y);
        break;
    case 4 :
        return hermit_func_second_der(2, 'x', x)
               * hermit_func(2, 'y', y)
               + hermit_func(2, 'x', x)
               * hermit_func_second_der(2, 'y', y);
        break;
    case 5 :
        return hermit_func_second_der(3, 'x', x)
               * hermit_func(1, 'y', y)
               + hermit_func(3, 'x', x)
               * hermit_func_second_der(1, 'y', y);
        break;
    case 6 :
        return hermit_func_second_der(4, 'x', x)
               * hermit_func(1, 'y', y)
               + hermit_func(4, 'x', x)
               * hermit_func_second_der(1, 'y', y);
        break;
    case 7 :
        return hermit_func_second_der(3, 'x', x)
               * hermit_func(2, 'y', y)
               + hermit_func(3, 'x', x)
               * hermit_func_second_der(2, 'y', y);
```

```
        break;
    case 8 :
        return hermit_func_second_der(4, 'x', x)
               * hermit_func(2, 'y', y)
               + hermit_func(4, 'x', x)
               * hermit_func_second_der(2, 'y', y);
        break;
    case 9 :
        return hermit_func_second_der(1, 'x', x)
               * hermit_func(3, 'y', y)
               + hermit_func(1, 'x', x)
               * hermit_func_second_der(3, 'y', y);
        break;
    case 10 :
        return hermit_func_second_der(2, 'x', x)
               * hermit_func(3, 'y', y)
               + hermit_func(2, 'x', x)
               * hermit_func_second_der(3, 'y', y);
        break;
    case 11 :
        return hermit_func_second_der(1, 'x', x)
               * hermit_func(4, 'y', y)
               + hermit_func(1, 'x', x)
               * hermit_func_second_der(4, 'y', y);
        break;
    case 12 :
        return hermit_func_second_der(2, 'x', x)
               * hermit_func(4, 'y', y)
               + hermit_func(2, 'x', x)
               * hermit_func_second_der(4, 'y', y);
        break;
    case 13 :
        return hermit_func_second_der(3, 'x', x)
               * hermit_func(3, 'y', y)
               + hermit_func(3, 'x', x)
               * hermit_func_second_der(3, 'y', y);
        break;
    case 14 :
        return hermit_func_second_der(4, 'x', x)
               * hermit_func(3, 'y', y)
               + hermit_func(4, 'x', x)
               * hermit_func_second_der(3, 'y', y);
        break;
    case 15 :
        return hermit_func_second_der(3, 'x', x)
               * hermit_func(4, 'y', y)
               + hermit_func(3, 'x', x)
               * hermit_func_second_der(4, 'y', y);
        break;
    case 16 :
        return hermit_func_second_der(4, 'x', x)
               * hermit_func(4, 'y', y)
               + hermit_func(4, 'x', x)
               * hermit_func_second_der(4, 'y', y);
        break;
    }
    cerr << "Unknown number detected!" << endl;
    return 0.0;
}

// Лаплассиан двумерных эрмитовых базисных функций
double fe_spline::lap_phi(size_t func_n, class node p)
{
    return lap_phi(func_n, p.x, p.y);
}

// Инициализация эрмитовых КЭ из обычных
void fe_spline::init(const finite_element & fe, node * n)
{
    nodes = n;
    for(int i = 0; i < 4; i++)
    {
        node_n[i] = fe.node_n[i];
        f[i] = fe.f[i];
    }
    lambda = fe.lambda;
    gamma = fe.gamma;

    hx = nodes[node_n[3]].x - nodes[node_n[0]].x;
    hy = nodes[node_n[3]].y - nodes[node_n[0]].y;
    jacobian = hx * hy / 4.0;

    // https://ru.wikipedia.org/wiki/Список_квадратурных_формул

    static double g_a = sqrt((114.0 - 3.0 * sqrt(583.0)) / 287.0);
    static double g_b = sqrt((114.0 + 3.0 * sqrt(583.0)) / 287.0);
    static double g_c = sqrt(6.0 / 7.0);
    static double g_wa = 307.0 / 810.0 + 923.0 / (270.0 * sqrt(583.0));
    static double g_wb = 307.0 / 810.0 - 923.0 / (270.0 * sqrt(583.0));
    static double g_wc = 98.0 / 405.0;

    for(size_t i = 0; i < 4; i++)
    {
        gauss_weights[i] = g_wc;
        gauss_weights[i+4] = g_wa;
        gauss_weights[i+8] = g_wb;
    }

    double gauss_points_local[2][12] =
    {
```

```
        {-g_c, g_c, 0.0, 0.0, -g_a, g_a, -g_a, g_a, -g_b, g_b, -g_b, g_b},
        {0.0, 0.0, -g_c, g_c, -g_a, -g_a, g_a, g_a, -g_b, -g_b, g_b, g_b}
    };

    for(size_t i = 0; i < 12; i++)
        gauss_points[i] = to_global(node(gauss_points_local[0][i], gauss_points_local[1][i]));
}

// Получние номеров одномерных БФ из номера двумерной
void fe_spline::two2one(size_t two, size_t & one1, size_t & one2)
{
    one1 = 2 * ((size_t)((two - 1) / 4) % 2) + ((two - 1) % 2) + 1;
    one2 = 2 * (size_t)((two - 1) / 8) + ((size_t)((two - 1) / 2) % 2) + 1;
}

// Скалярное произведение градиентов двумерных эрмитовых базисных функций
double fe_spline::grad_phi(size_t bf1, size_t bf2, double x, double y)
{
    size_t b1, b2;
    two2one(bf1, b1, b2);
    double tmp1 = hermit_func_first_der(b1, 'x', x);
    double tmp2 = hermit_func(b1, 'x', x);
    double tmp3 = hermit_func_first_der(b2, 'y', y);
    double tmp4 = hermit_func(b2, 'y', y);
    double dx1 = tmp1 * tmp4;
    double dy1 = tmp3 * tmp2;
    two2one(bf2, b1, b2);
    tmp1 = hermit_func_first_der(b1, 'x', x);
    tmp2 = hermit_func(b1, 'x', x);
    tmp3 = hermit_func_first_der(b2, 'y', y);
    tmp4 = hermit_func(b2, 'y', y);
    double dx2 = tmp1 * tmp4;
    double dy2 = tmp3 * tmp2;
    return dx1 * dx2 + dy1 * dy2;
}

// Скалярное произведение градиентов двумерных эрмитовых базисных функций
double fe_spline::grad_phi(size_t bf1, size_t bf2, node p)
{
    return grad_phi(bf1, bf2, p.x, p.y);
}

// Перевод в систему координат мастер-элемента
node fe_spline::to_local(node p)
{
    double ksi = 2.0 * (p.x - nodes[node_n[0]].x) / hx - 1.0;
    double eta = 2.0 * (p.y - nodes[node_n[0]].y) / hy - 1.0;
    return node(ksi, eta);
}

// Перевод в глобальную систему координат
node fe_spline::to_global(node p)
{
    double x = (p.x + 1.0) * hx / 2.0 + nodes[node_n[0]].x;
    double y = (p.y + 1.0) * hy / 2.0 + nodes[node_n[0]].y;
    return node(x, y);
}

// Интеграл от скалярного произведения градиентов двумерных эрмитовых базисных функций
double fe_spline::integrate_grad_phi(size_t bf1, size_t bf2)
{
    double result = 0.0;
    for(int g = 0; g < 12; g++)
    {
        result += gauss_weights[g] * grad_phi(bf1, bf2, gauss_points[g]);
    }
    return result;
}

// Интеграл от лапласиана двумерных эрмитовых базисных функций
double fe_spline::integrate_lap_phi(size_t bf1, size_t bf2)
{
    double result = 0.0;
    for(int g = 0; g < 12; g++)
    {
        result += gauss_weights[g] * lap_phi(bf1, gauss_points[g]) * lap_phi(bf2, gauss_points[g]);
    }
    return result;
}
```