Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Численные методы

Курсовой проект

Факультет	ПМИ
Группа	ПМ-01
Студент	Осяев Д.С.
Преподаватель	Персова М.Г.
Вариант	26

Новосибирск

2013

1. Постановка задачи

1.1. Формулировка задания

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции линейные на треугольниках. Краевые условия всех типов. Коэффициент γ разложить по квадратичным базисным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

1.2. Постановка задачи

Эллиптическая краевая задача для функции u определяется уравнением
$-div(\lambda \cdot grad(u)) + \gamma u = f$,
заданным в некоторой области Ω с границей $S = S_1 \cup S_2 \cup S_3$ и краевыми условиями:

$$u|_{S_1} = u_g$$

$$\lambda \frac{\partial u}{\partial n}|_{S_2} = \theta$$

$$\lambda \frac{\partial u}{\partial n}|_{S_3} + \beta(u|_{S_3} - u_\beta) = 0$$

В декартовой системе координат {x,y} это уравнение может быть записано в виде

$$-\frac{\partial}{\partial x}\left(\lambda \frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(\lambda \frac{\partial u}{\partial y}\right) + \gamma u = f$$

2. Теоретическая часть

2.1. Вариационная постановка в форме уравнения Галеркина

В основе МКЭ лежат т.н. вариационные постановки, в которых решение краевых задач заменяется минимизацией некоторого функционала. Областью определения этого функционала является Гильбертово пространство функций, содержащее в качестве одного из своих элементов решение u данной краевой задачи.

В операторной форме исходное уравнение можно переписать в форме $Lu = f$, где L - оператор, действующий в Гильбертовом пространстве H (для данной задачи мы работаем в L_2 - пространстве функций, интегрируемых с квадратом). Нам нужно найти приближение к элементу $u \in H$, соответствующее заданному элементу $f \in H$.

Пусть $\varphi_1, \varphi_2, ..., \varphi_n, ...$ – некоторая полная замкнутая система линейно независимых элементов из H . Ее первые n элементов выделяют в H конечномерное подпространство H_n, в котором и ищется приближенное решение уравнения в виде $u_n = \sum_{i=1}^{n} q_i \psi_i$. Поскольку рассматриваемый оператор L - оператор Лапласа, то $Lu_n \in H_n$. Отсюда, исходя из того, что любой элемент из H может быть представлен в виде суммы элемента из H_n и элемента, ортогонального к этому подпространству, получаем: $f = Lu^h + (f - Lu^h) \in H$. Тогда $(f - Lu^h) \perp H_n$ и, соответственно, $(f - Lu^h) \perp \psi_i$, где ψ_i, $i = \overline{1,n}$ - финитные базисные функции.

Таким образом, приближенное решение будем искать как проекцию на конечномерное подпространство гильбертова пространства H_n, натянутого на систему базисных функций ψ_i, $i = \overline{1,n}$. Отсюда получаем, что $(Lu_n - f, \psi_i) = 0,$ или $(Lu_n, \psi_i) = (f, \psi_i),$ $i = \overline{1,n}.$ То есть, бу-

дем искать приближенное решение в виде разложения по базисным функциям конечномерного подпространства гильбертова пространства.

Поскольку мы рассматриваем Гильбертово пространство функций, интегрируемых с квадратом, то есть L_2, то скалярное произведение в двумерном случае можно переписать в виде:

$$\int_\Omega (-div(\lambda \cdot grad\, u_n) + \gamma u_n)\psi_i d\Omega = \int_\Omega f\psi_i d\Omega$$

Применяя формулу (Грина) интегрирования по частям для многомерного случая, перепишем уравнение в виде:

$$\int_\Omega \lambda \cdot grad\, u_n \cdot grad\, \psi_i d\Omega + \int_\Omega \gamma u_n \psi_i d\Omega - \int_S \lambda \frac{\partial u}{\partial n}\psi_i dS = \int_\Omega f\psi_i d\Omega$$

Учитывая, что $S = S_1 \cup S_2 \cup S_3$: $\int_S \lambda \frac{\partial u}{\partial n}\psi_i dS = \int_{S_1} \lambda \frac{\partial u}{\partial n}\psi_i dS + \int_{S_2} \lambda \frac{\partial u}{\partial n}\psi_i dS + \int_{S_3} \lambda \frac{\partial u}{\partial n}\psi_i dS$

Теперь учтем заданные краевые условия. Поскольку $\psi_i|_{S_1} = 0$, а, значит, и $\int_{S_1} \lambda \frac{\partial u}{\partial n}\psi_i dS = 0$, то интегральное соотношение принимает вид:

$$\int_\Omega \lambda \cdot grad\, u_n \cdot grad\, \psi_i d\Omega + \int_\Omega \gamma u_n \psi_i d\Omega - \int_{S_2} \theta\psi_i dS = \int_\Omega f\psi_i d\Omega - \int_{S_3} \beta(u - u_\beta)\psi_i dS$$

Исходя из того, что $u_n = \sum_{i=1}^{n} q_i\psi_i$, перепишем уравнение в виде:

$$\sum_{j=1}^{n} q_j \int_\Omega \lambda \cdot grad\, \psi_j \cdot grad\, \psi_i d\Omega + \sum_{j=1}^{n} q_j \int_\Omega \gamma\psi_j\psi_i d\Omega + \sum_{j=1}^{n} q_j \int_{S_3} \beta\psi_j\psi_i dS = \int_\Omega f\psi_i d\Omega + \int_{S_2} \theta\psi_i dS + \int_{S_3} \beta u_\beta \psi_i dS$$

Поскольку исходная задача рассматривается в декартовой системе координат, то

$grad\, u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)$ и, соответственно: $grad\, u \cdot grad\, v = \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}$

Отсюда получаем уравнение в виде:

$$\sum_{j=1}^{n} q_j \int_\Omega \lambda \cdot \left(\frac{\partial \psi_j}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y}\frac{\partial \psi_i}{\partial y}\right)dxdy + \sum_{j=1}^{n} q_j \int_\Omega \gamma\psi_j\psi_i dxdy + \sum_{j=1}^{n} q_j \int_{S_3} \beta\psi_j\psi_i dxdy = \int_\Omega f\psi_i dxdy +$$

$$+ \int_{S_2} \theta\psi_i dxdy + \int_{S_3} \beta u_\beta \psi_i dxdy$$

2.2. Конечноэлементная дискретизация

Так как для решения задачи используются линейные базисные функции, то на каждом конечном элементе Ω_k - треугольнике эти функции будут совпадать с функциями $L_1(x,y)$, $L_2(x,y)$, $L_3(x,y)$, такими, что $L_1(x,y)$ равна единице в вершине (x_1, y_1) и нулю во всех остальных вершинах, $L_2(x,y)$ равна единице в вершине (x_2, y_2) и нулю во всех остальных вершинах, $L_3(x,y)$ равна единице в вершине (x_3, y_3) и нулю во всех остальных вершинах. Любая линейная на Ω_k функция представима в виде линейной комбинации этих базисных линейных функций, коэффициентами будут значения функции в каждой из вершин треугольника Ω_k. Таким образом, на каждом конечном элементе нам понадобятся три узла – вершины треугольника.

Получаем:

$\psi_1 = L_1(x, y)$

$\psi_2 = L_2(x, y)$

$\psi_3 = L_3(x, y)$

При дальнейшем решении задачи будем использовать соотношения:

$$\int_{\Omega_k} (L_1)^{\nu_1} (L_2)^{\nu_2} (L_3)^{\nu_3} d\Omega_k = \frac{\nu_1!\nu_2!\nu_3!}{(\nu_1+\nu_2+\nu_3+2)!} 2mes\Omega_k$$

$$\int_{\Gamma} (L_i)^{\nu_i} (L_j)^{\nu_j} dS = \frac{\nu_i!\nu_j!}{(\nu_i+\nu_j+1)!} mes\Gamma, \quad i \neq j, \qquad (*)$$

где $mes\Omega_k = \frac{1}{2}|\det D|$ - это площадь треугольника, $D = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}$ - матрица координат его вершин.

Учитывая построение L - функций, получаем следующие соотношения:

$$\begin{cases} L_1 + L_2 + L_3 = 1, \\ L_1 x_1 + L_2 x_2 + L_3 x_3 = x, \\ L_1 y_1 + L_2 y_2 + L_3 y_3 = y. \end{cases}$$

Т.е. имеем систему:

$$\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} \cdot \begin{pmatrix} L_1 \\ L_2 \\ L_3 \end{pmatrix} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}$$

Отсюда находим коэффициенты линейных функций $L_1(x, y)$, $L_2(x, y)$, $L_3(x, y)$

$L_i = \alpha_0^i + \alpha_1^i x + \alpha_2^i y, \quad i = \overline{1,3}$

$$\begin{pmatrix} \alpha_0^1 & \alpha_1^1 & \alpha_2^1 \\ \alpha_0^2 & \alpha_1^2 & \alpha_2^2 \\ \alpha_0^3 & \alpha_1^3 & \alpha_2^3 \end{pmatrix} = D^{-1} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1}$$

$$D^{-1} = \frac{1}{|\det D|} \begin{pmatrix} x_2 y_3 - x_3 y_2 & y_2 - y_3 & x_3 - x_2 \\ x_3 y_1 - x_1 y_3 & y_3 - y_1 & x_1 - x_3 \\ x_1 y_2 - x_2 y_1 & y_1 - y_2 & x_2 - x_1 \end{pmatrix}$$

При вычислении интегралов от произведений вида $L_i L_j$ по треугольнику Ω_k или любому его ребру Γ можно использовать вышеуказанные формулы (*).

2.3. Переход к локальным матрицам

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

$$\int_{\Omega_k} \lambda\left(\frac{\partial \psi_j}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y}\frac{\partial \psi_i}{\partial y}\right)dxdy + \int_{\Omega_k} \gamma \psi_j \psi_i dxdy = \int_{\Omega_k} f \psi_i dxdy$$

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

2.3.1. Построение матрицы массы

Рассмотрим второй член в вышеуказанном выражении:

$$\int_{\Omega_k} \gamma \psi_i \cdot \psi_j \, dxdy$$

Учитывая, что $\psi_j = L_j, \psi_i = L_i$, получим:

$$\int_{\Omega_k} \gamma L_i \cdot L_j \, dxdy$$

В поставленной задаче требуется разложить γ по квадратичным базисным функциям: $\gamma = \sum_{p=0}^{5} \gamma_p \varphi_p$, где γ_p - значения коэффициента γ в соответствующих узлах, φ_p - квадратичные базисные функции, которые определяются следующим образом:

$$\varphi_0 = L_1\left(2L_1 - 1\right) \qquad \varphi_1 = L_2\left(2L_2 - 1\right) \qquad \varphi_2 = L_2\left(2L_2 - 1\right)$$
$$\varphi_3 = 4L_1L_2 \qquad \varphi_4 = 4L_2L_3 \qquad \varphi_5 = 4L_1L_3$$

Таким образом, $C_{i,j} = \left(\sum_{p=0}^{5} \gamma_p \int_{\Omega_m} \varphi_p \psi_i \psi_j \, d\Omega_m\right)$, $i, j = \overline{0,2}$

2.3.2. Построение матрицы жёсткости

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

$$\int_{\Omega_k} \lambda\left(\frac{\partial \psi_j}{\partial x}\frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y}\frac{\partial \psi_i}{\partial y}\right)dxdy$$

$$B_{i,j} = \left(\alpha_1^i \alpha_1^j + \alpha_2^i \alpha_2^j\right)\frac{\left|\det D\right|}{2} \quad i, j = \overline{0,2}$$

2.3.3. Построение вектора правой части

Рассмотрим правую часть выражения для k-го конечного элемента:

$$\int_{\Omega_k} f \psi_i dxdy$$

f представим в виде $f = f_1L_1 + f_2L_2 + f_3L_3$, где f_i - значения в вершинах треугольника. Получим:

$$\int_{\Omega_k} f_q L_q L_i dxdy = f_q \int_{\Omega_k} L_q L_i d\Omega_k$$

Таким образом, $G_i = \sum_{q=1}^{3} f_q \int_{\Omega_k} L_q L_i \, d\Omega_k$, $i = \overline{0,2}$.

2.3.4. Сборка глобальной матрицы и глобального вектора правой части

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

2.4. Учет краевых условий

2.4.1. Учет первых краевых условий

Для учета первых краевых условий, в глобальной матрице и глобальном векторе находим соответствующую глобальному номеру краевого узла строку, и ставим вместо диагонального элемента глобальной матрицы на этой строке «большое число», а вместо элемента с таким номером в вектор правой части - «большое число», умноженное на значение краевого условия, заданное в исходной задаче.

2.4.1. Учет вторых и третьих краевых условий

Рассмотрим краевые условия второго и третьего рода

$$\lambda \left.\frac{\partial u}{\partial n}\right|_{S_2} = \theta,$$

$$\lambda \left.\frac{\partial u}{\partial n}\right|_{S_3} + \beta(u|_{S_3} - u_\beta) = 0.$$

Отсюда получаем, что для учета краевых условий необходимо вычислить интегралы:

$$\int_{S_2} \theta\psi_j \, dxdy, \quad \int_{S_3} \beta(u_\beta)\psi_j \, dxdy, \quad \int_{S_3} \beta\psi_i\psi_j \, dxdy.$$

Краевые условия второго и третьего рода задаются на ребрах, т.е. определяются двумя узлами, лежащими на ребре.

Будем считать, что параметр β на S_3 постоянен, тогда параметр u_β будем раскладывать по двум базисным функциям, определенным на этом ребре.

$u_\beta = u_{\beta 0}\varphi_0 + u_{\beta 1}\varphi_1$, где φ_i, $i = \overline{0,1}$ - локально занумерованные линейные базисные функции, которые имеют также свои глобальные номера во всей расчетной области, а $u_{\beta i}$ - значения функции u_β в узлах ребра.

Аналогично поступаем и при учете вторых краевых условий, раскладывая по базису ребра функцию $\theta = \theta_0\varphi_0 + \theta_1\varphi_1$.

Тогда приведенные выше интегралы примут вид:

$$I_1 = \int_{S_2} (\theta_0\varphi_0 + \theta_1\varphi_1)\varphi_i \, dxdy,$$

$$I_2 = \beta \int_{S_3} (u_{\beta 1}\varphi_0 + u_{\beta 2}\varphi_1)\varphi_i \, dxdy,$$

$$I_3 = \beta \int_{S_3} \varphi_i\varphi_j \, dxdy.$$

Фактически, решая задачу учета краевых условий второго и третьего рода, мы переходим к решению одномерной задачи на ребре для того, чтобы занести соответствующие результаты в глобальную матрицу и вектор.

Базисными функциями ребра являются две ненулевые на данном ребре базисные функции из ψ_i , $i = \overline{1,3}$ конечного элемента.

Для учета вклада вторых и третьих краевых условий рассчитываются 2 матрицы 2×2.

Интегралы I_1, I_2, I_3 будем вычислять по формуле $\int_{\Gamma} (L_i)^{\nu_i} \cdot (L_j)^{\nu_j} dS = \frac{\nu_i ! \nu_j !}{(\nu_i + \nu_j + 1)!} mes\Gamma, \quad i \neq j$, где $mes\Gamma$ длина ребра. При этом независимо от того, что на каждом из ребер присутствуют свои базисные функции, интегралы, посчитанные по приведенным выше формулам, будут равны.

$$I_1 = \begin{pmatrix} \int_{S_2} L_1 L_1 dxdy & \int_{S_2} L_1 L_2 dxdy \\ \int_{S_2} L_2 L_1 dxdy & \int_{S_2} L_2 L_2 dxdy \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} = \frac{1}{6} mes\, S_2 \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$$

Этот вектор поправок в правую часть позволяет учесть не только вторые краевые условия, но и часть βu_β из третьих.

Осталось рассмотреть матрицу поправок в левую часть.

$$I_3 = \beta \int_{S_3} \varphi_i \varphi_j dxdy.$$

Очевидно, что получится та же матрица, только не умноженная на вектор констант.

$$I_3 = \frac{1}{6} mes\, S_3 \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Добавляя эту матрицу в левую часть, на места соответствующие номерам узлов, получаем учет третьих краевых условий.

При расчете θ и $\beta(u|_{S_3} - u_\beta)$ должно учитываться направление нормали $\lambda \left.\frac{\partial u}{\partial n}\right|_S = \lambda\, gradu \cdot \vec{n}$.

Если рассматривать нормаль к наклонной стороне области, то для каждой из двух точек ребра, в которых рассматриваются нормали, значения производных решения по обеим координатам будет ненулевыми, если, производная самой функции по какой-либо координате не будет нулевой.

2.5. Метод решения СЛАУ

Для решения СЛАУ с полученной матрицей в разреженно-строчном формате будем применять ЛОС. В результате получим вектор, элементами которого и будут искомые коэффициенты разложения нашего решения по базисным функциям, учитывая построение базисных функций, то фактически мы получим вектор, координатами которого будут значения функции-решения в узлах сетки.

3.Текст программы

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <math.h>

using namespace std;
```

```
double **point=NULL;
double **gamma_betta=NULL;
int **finit=NULL;
int *finit_in_area=NULL;
int **kraev=NULL;
int *point_in_area=NULL;

int *ig=NULL;
int *jg=NULL;
double *di=NULL;
double *ggl=NULL;
double *ggu=NULL;
double *F=NULL;
double *x=NULL;

double *z,*r,*p,*t,*r1,*l,*l1,*f;

int num_points=0;//количество точек
int num_finit_elements=0;//количество конечных элементов
int num_areas=0;
int num_kraev=0;

double func(double x,double y,int i)
{
	if(i == 0)
		return -20;
	return 0;
}

double func_kraev1(double *x,int k)
{
	switch(k){
		case 0:  return x[1]*x[1];
		default: return 0;
	}
}

double func_kraev2(double *x,int k)
{
	switch(k)
	{
		case 0: return 20;
		case 1: return 0;
		//case 2: return (2.);
		default: return 0;
	}
}

double func_kraev3(double *x,int k)
{
	if (k == 0)
		return (20*x[1]-27);
	return 0;
}

double resh(double x,double y,int k)
{
	switch(k)
	{
		case 0: return (y*y);
		case 1: return 20.0*y-19.0;
		default: return 0;
	}
}

int input()
{
	ifstream _file("coords.txt");
```

```
if(!_file.is_open())
{
	_file.close();
	return 1;
}

_file >> num_points;

if(num_points<=0)
{
	_file.close();
	return 2;
}

point = new double*[num_points];
for(int i=0; i < num_points; i++){
	point[i]=new double[2];
	for(int j=0; j<2; j++)
		_file >> point[i][j];
}
_file.close();

ifstream _file1("finit_elements.txt");
if(!_file1.is_open()){
	_file1.close();
	return 1;
}
_file1 >> num_finit_elements;
if(num_finit_elements<=0){
	_file.close();
	return 2;
}
finit=new int*[num_finit_elements];
for(int i=0; i < num_finit_elements; i++){
	finit[i]=new int[3];
	for(int j = 0; j < 3; j++)
		_file1 >> finit[i][j];
}
_file1.close();

ifstream _file2("areas.txt");
if(!_file2.is_open()){
	_file2.close();
	return 1;
}
_file2 >> num_areas;
if(num_areas<=0){
	_file.close();
	return 2;
}
gamma_betta=new double*[num_areas];
finit_in_area=new int[num_finit_elements];
for(int i = 0; i < num_areas; i++){
	gamma_betta[i]=new double[2];
	for(int j = 0; j < 2; j++)
		_file2 >> gamma_betta[i][j];
}
for(int i = 0; i < num_finit_elements; i++)
	_file2 >> finit_in_area[i];
_file2.close();

ifstream _file3("board.txt");
if(!_file3.is_open()){
	_file3.close();
	return 1;
}
_file3 >> num_kraev;
kraev = new int*[num_kraev];
```

```
	for(int i = 0; i < num_kraev; i++){
		kraev[i]=new int[4];
		for(int j = 0; j < 5; j++)
			_file3 >> kraev[i][j];
	}
	_file3.close();

	ifstream _file4("point_in_area.txt");
	if(!_file4.is_open()){
		_file4.close();
		return 1;
	}
	point_in_area = new int[num_points];
	for(int i = 0; i < num_points; i++)
		_file4 >> point_in_area[i];
	_file4.close();
	return 0;
}
void sort(int *mas,int k)
{
	int l=0;
	for(int i=k-1;i>=0;i--)
		for(int j=0;j<=i;j++)
			if(mas[j]>mas[j+1]) {
				l=mas[j];
				mas[j]=mas[j+1];
				mas[j+1]=l;
			}
}

void portret()
{
	int i=0;
	int j=0;
	int k=0;
	int kk=0;
	int key=0;
	int position=0;//позиция в массиве jg, в которую надо добавлять
	int *mas=new int[num_points];
	struct List{
		int num;
		List *next;
	};

	List *list=new List[num_points];
	List *p=NULL;

	ig=new int[num_points+1];

	for(i=0;i<num_points;i++)
		list[i].next=NULL;

	//составление "массива", содержащего номер точки и смежные с ней
	for(i=0;i<num_finit_elements;i++){
		for(j=0; j<3; j++){
			key=0;
			k=finit[i][(j==2)];//0 0 1
			kk=finit[i][((j!=0)+1)];// 1 2 2
			if(k<kk){
				k+=kk;
				kk=k-kk;
				k-=kk;
			}
			p=&list[k];
			while(p->next){
				if(p->next->num==kk){
					key=1;
					break;
				}
```

```
                        p=p->next;
                }
                if(!key){
                        p->next = new List;
                        p->next->num=kk;
                        p->next->next=NULL;
                }
        }
    }

    //составление массива ig
    ig[0]=0;
    for(i=0; i<num_points; i++){
        k=0;
        p=&list[i];
        while(p=p->next)
            k++;
        ig[i+1]=ig[i]+k;
    }

    jg=new int[ig[i]-1];
    //составление массива jg
    for(i=0;i<num_points;i++){
        k=0;
        key=0;
        p=&list[i];
        while(p=p->next){
            mas[k]=p->num;
            k++;
            key=1;
        }
        if(key){
            sort(mas,--k);//сортировка
            int ii=0;//добавляет в jg
            int jj=0;
            for(ii=position,jj=0;ii<=k+position;ii++,jj++)
                jg[ii]=mas[jj];

            position+=k+1;
        }
    }
}

void tochnoe()
{
    for(int i = 0; i < num_points; i++)
        x[i]=resh(point[i][0], point[i][1], point_in_area[i]);
}

double lambda(int k)
{
    if(!k) return 10;
    return 1;
}

double mes_G(double *x,double *y)
{
    return (sqrt((y[0]-x[0])*(y[0]-x[0])+(y[1]-x[1])*(y[1]-x[1])));
}

void M_matrix(double *p1,double *p2,double *p3,double gamma,double **M_matr, double *local_F,int
num_of_area)
{
    int i=0;
    int j=0;
    double det=(p2[0]-p1[0])*(p3[1]-p1[1])-(p3[0]-p1[0])*(p2[1]-p1[1]);
    double mnoz=fabs(det)/24;
    double *f=new double[3];
    double mnoz2=mnoz*gamma;
```

```
	f[0]=mnoz*func(p1[0],p1[1],num_of_area);
	f[1]=mnoz*func(p2[0],p2[1],num_of_area);
	f[2]=mnoz*func(p3[0],p3[1],num_of_area);

	local_F[0]=2*f[0]+f[1]+f[2];
	local_F[1]=f[0]+2*f[1]+f[2];
	local_F[2]=f[0]+f[1]+2*f[2];

	for(i=0;i<3;i++)
		for(j=0;j<3;j++)
			if(i==j)
				M_matr[i][j]=2*mnoz2;
			else
				M_matr[i][j]=mnoz2;

}

void G_matrix(double *p1, double *p2, double *p3,double **G_matr, int k)
{
	int i=0;
	int j=0;
	double ck=0;
	double det=(p2[0]-p1[0])*(p3[1]-p1[1])-(p3[0]-p1[0])*(p2[1]-p1[1]);
	double *p12=new double[2];
	double *p23=new double[2];
	double *p31=new double[2];
	for(i=0;i<2;i++){
		p12[i]=(p1[i]+p2[i])/2;
		p23[i]=(p2[i]+p3[i])/2;
		p31[i]=(p3[i]+p1[i])/2;
	}

	ck=lambda(k)*fabs(det)/2;
	G_matr[0][0]=ck*((p2[1]-p3[1])*(p2[1]-p3[1])/(det*det)+(p3[0]-p2[0])* (p3[0]-
p2[0])/(det*det));
	G_matr[0][1]=ck*((p2[1]-p3[1])*(p3[1]-p1[1])/(det*det)+(p3[0]-p2[0])* (p1[0]-
p3[0])/(det*det));
	G_matr[0][2]=ck*((p2[1]-p3[1])*(p1[1]-p2[1])/(det*det)+(p3[0]-p2[0])* (p2[0]-
p1[0])/(det*det));
	G_matr[1][0]=ck*((p3[1]-p1[1])*(p2[1]-p3[1])/(det*det)+(p1[0]-p3[0])* (p3[0]-
p2[0])/(det*det));
	G_matr[1][1]=ck*((p3[1]-p1[1])*(p3[1]-p1[1])/(det*det)+(p1[0]-p3[0])* (p1[0]-
p3[0])/(det*det));
	G_matr[1][2]=ck*((p3[1]-p1[1])*(p1[1]-p2[1])/(det*det)+(p1[0]-p3[0])* (p2[0]-
p1[0])/(det*det));
	G_matr[2][0]=ck*((p2[1]-p3[1])*(p1[1]-p2[1])/(det*det)+(p3[0]-p2[0])* (p2[0]-
p1[0])/(det*det));
	G_matr[2][1]=ck*((p3[1]-p1[1])*(p1[1]-p2[1])/(det*det)+(p1[0]-p3[0])* (p2[0]-
p1[0])/(det*det));
	G_matr[2][2]=ck*((p1[1]-p2[1])*(p1[1]-p2[1])/(det*det)+(p2[0]-p1[0])* (p2[0]-
p1[0])/(det*det));
}

void local_matrix(int num_of_finit_element, double **local_matr, double *local_F)
{
	int k=finit[num_of_finit_element][0];
	int l=finit[num_of_finit_element][1];
	int m=finit[num_of_finit_element][2];
	double **M_matr=new double*[3];
	double **G_matr=new double*[3];

	for(int i = 0; i < 3; i++){
		M_matr[i]=new double[3];
		G_matr[i]=new double[3];
	}


	M_matrix(	point[k], point[l], point[m],
```

```
                        gamma_betta[finit_in_area[num_of_finit_element]][0],
                        M_matr, local_F,
                        finit_in_area[num_of_finit_element]);

        G_matrix(       point[k], point[l], point[m],
                        G_matr,
                        finit_in_area[num_of_finit_element]);

        for(int i = 0; i < 3; i++)
                for(int j = 0; j < 3; j++)
                        local_matr[i][j] = M_matr[i][j] + G_matr[i][j];
}

int uchet_kraev(int *current_kraev,  double **a,  double *b, double betta)
{
        double ck=0;
        if(current_kraev[3]==1)
                return 0;
        else
                if(current_kraev[3]==2)
                {
                        ck=mes_G(point[current_kraev[1]],point[current_kraev[2]])/6.0;
                        b[0]=ck*(2*func_kraev2(point[current_kraev[1]],current_kraev[4])+
func_kraev2(point[current_kraev[2]],current_kraev[4]));
                        b[1]=ck*(func_kraev2(point[current_kraev[1]],current_kraev[4])+
2*func_kraev2(point[current_kraev[2]],current_kraev[4]));
                        return 1;
                }
                else
                {
                        ck=betta*mes_G(point[current_kraev[1]],point[current_kraev[2]])/6;
                        a[0][0]=2*ck;
                        a[0][1]=ck;
                        a[1][0]=ck;
                        a[1][1]=2*ck;
                        b[0]=ck*(2*func_kraev3(point[current_kraev[1]],current_kraev[4])+
func_kraev3(point[current_kraev[2]],current_kraev[4]));
                        b[1]=ck*(func_kraev3(point[current_kraev[1]],current_kraev[4])+
2*func_kraev3(point[current_kraev[2]],current_kraev[4]));
                        return 2;
                }
}

void pervoe_kraevoe(int *current_kraev)
{
        int kol=0,m=0;
        int lbeg;
        int lend;
        di[current_kraev[1]]=1;
        di[current_kraev[2]]=1;

        F[current_kraev[1]]=func_kraev1(point[current_kraev[1]],current_kraev[4]);
        F[current_kraev[2]]=func_kraev1(point[current_kraev[2]],current_kraev[4]);

        kol=ig[current_kraev[1]+1]-ig[current_kraev[1]];

        for(int i = 0; i < kol; i++)
                ggl[ig[current_kraev[1]]+i]=0;

        kol = ig[current_kraev[2]+1] - ig[current_kraev[2]];
        for(int i = 0; i < kol; i++)
                ggl[ig[current_kraev[2]]+i] = 0;

        for(int i = current_kraev[1] + 1; i < num_points; i++){
                lbeg = ig[i];
                lend = ig[i+1];
                for(int p = lbeg; p < lend; p++)
                        if(jg[p]==current_kraev[1]){
                                ggu[p]=0;
```

```
                        continue;
                }
        }
        for(int i = current_kraev[2]+1; i < num_points; i++)
        {
                lbeg=ig[i];
                lend=ig[i+1];
                for(int p = lbeg; p < lend; p++)
                        if(jg[p] == current_kraev[2]){
                                ggu[p]=0;
                                continue;
                        }
        }
}

void global_matrix()
{
        int i=0;
        int j=0;
        int k=0;
        int p=0;
        int ibeg=0;
        int iend=0;
        int h=0;
        int key=0;
        int kol=0;
        int *L=new int[3];
        int *L2=new int[2];
        int *K=new int[num_points/2];

        double *local_F=new double[3];
        double **local_matr=new double*[3];
        double *b=new double[2];//вектор для краевых
        double **a=new double*[2];//матрица для краевых

        for(i=0;i<2;i++){
                a[i]=new double[2];
                b[i]=0;
                for(j=0;j<2;j++)
                        a[i][j]=0;
        }

        for(i=0;i<3;i++)
                local_matr[i]=new double[3];

        for(k=0;k<num_finit_elements;k++){
                local_matrix(k,local_matr,local_F);
                memcpy(L,finit[k],3*sizeof(double));
                //локальная в глобальную//
                for(i=0; i<3; i++){
                        //h=0;
                        ibeg=L[i];
                        for(j=i+1; j<3; j++){
                                iend=L[j];
                                if(ibeg < iend){
                                        h=ig[iend];
                                        while(jg[h++]-ibeg);
                                        h--;
                                        ggl[h]+=local_matr[i][j];
                                        ggu[h]+=local_matr[j][i];
                                }
                                else{
                                        h=ig[ibeg];
                                        while(jg[h++]-iend);
                                        h--;
                                        ggl[h]+=local_matr[i][j];
                                        ggu[h]+=local_matr[j][i];
                                }
                        }
```

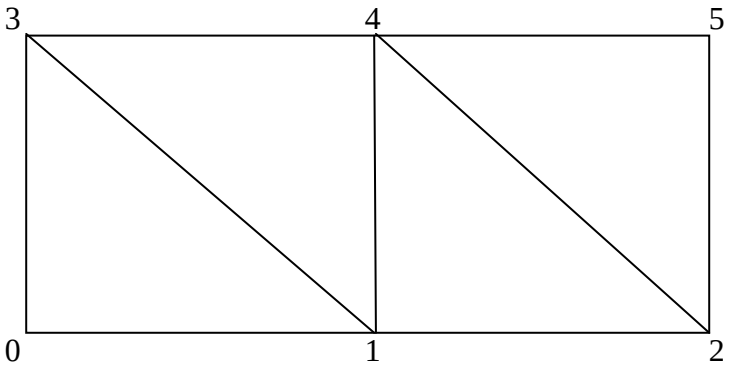
```
                        di[ibeg]+=local_matr[i][i];
                }
                //правая часть//
                for(i=0;i<3;i++)
                        F[L[i]]+=local_F[i];
        }

        for(k=0;k<num_kraev;k++){
                key=uchet_kraev(kraev[k],a,b,gamma_betta[kraev[k][0]][1]);
                if(!key){
                        K[p]=k;
                        p++;
                        continue;
                }
                L2[0]=kraev[k][1];
                L2[1]=kraev[k][2];
                if(key==2){
                        for(i=0; i<2; i++){
                                ibeg=L2[i];
                                for(j=i+1; j<2; j++){
                                        iend=L2[j];
                                        if(ibeg<iend){
                                                h=ig[iend];
                                                while(jg[h++]-ibeg);
                                                h--;
                                                ggl[h]+=a[i][j];
                                                ggu[h]+=a[j][i];
                                        }
                                        else {
                                                h=ig[ibeg];
                                                while(jg[h++]-iend);
                                                h--;
                                                ggl[h]+=a[i][j];
                                                ggu[h]+=a[j][i];
                                        }
                                }
                                di[ibeg]+=a[i][i];
                        }
                }
                for(i=0;i<2;i++)
                        F[L2[i]]+=b[i];

                for(i=0;i<2;i++){
                        b[i]=0;
                        for(j=0;j<2;j++)
                                a[i][j]=0;
                }
        }
        for(i = 0; i < p; i++)
                pervoe_kraevoe(kraev[K[i]]);
}

double norma(double *w)//НОРМА ВЕКТОРА
{
        double s=0;

        for (int i=0;i<num_points;i++)
                s+=w[i]*w[i];

        return sqrt(s);
}

double calc(int i,int j,double*gl,double* gu,int kl)//по формуле это сумма которую вычитаем
{
        double s=0;
        int k,J=jg[kl],p;
        for(k=j;k>0;k--)
                for(p=ig[J];p<ig[J+1];p++)
                        if(jg[p]==jg[kl-k])
```

```
                                s+=gl[kl-k]*gu[p];
        return s;
}

double calcD(int j,double*gl,double* gu,int kl)//аналогично только для диагонали
{
        double s=0;
        for(int k = kl-j; k < kl; k++)
                s+=gl[k]*gu[k];
        return s;
}

void lulu(double*gl,double*gu,double *gd)
{
        int i,j,kol,kl=0,ku=0;
        for(i=0;i<num_points;i++){
                kol=ig[i+1]-ig[i];
                for(j = 0; j < kol; j++,kl++)
                        gl[kl]=(ggl[kl]-calc(i,j,gl,gu,kl))/gd[jg[kl]];

                for(j=0;j<kol;j++,ku++)
                        gu[ku]=(ggu[ku]-calc(i,j,gu,gl,ku))/gd[jg[ku]];

                gd[i]=sqrt(di[i]-calcD(j,gu,gl,kl));
        }
}

void Mult_A_Vect(double *xn)//перемножение исходной матрицы A на вектор
{
        long i,j,st;
        for(i = 0; i < num_points; i++){
                f[i] = di[i] * xn[i];
                for(j = ig[i]; j < ig[i+1]; j++){
                        st = jg[j];
                        f[i] += ggl[j]*xn[st];
                        f[st] += ggu[j]*xn[i];
                }
        }
}//на выходе вектор f который является глобальным

double sk_pr(double*a,double*b)//скалярное произведение векторов.
{
        double s=0;
        for(int i = 0; i < num_points; i++)
                s+=a[i]*b[i];
        return s;
}

void LOC()
{
        double nvzk, alfa,beta,skp,eps = 9.999999682655226e-030;
        int i;

        double lastnvzk;

        Mult_A_Vect(x);

        for(i=0; i<num_points; i++)
                z[i] = r[i] = F[i]-f[i];

        Mult_A_Vect(z);

        for(i=0; i<num_points; i++)
                p[i]=f[i];

        nvzk=sqrt(sk_pr(r,r))/sqrt(sk_pr(F,F));

        for(int k=1; k<10000 && nvzk > eps; k++)
        {
```

```
                lastnvzk = nvzk;
                skp=sk_pr(p,p);
                alfa=sk_pr(p,r)/skp;
                for(i=0; i<num_points; i++){
                        x[i]+=alfa*z[i];
                        r[i]-=alfa*p[i];
                }

                Mult_A_Vect(r);
                beta=-sk_pr(p,f)/skp;
                for(i=0; i<num_points; i++){
                        z[i]=r[i]+beta*z[i];
                        p[i]=f[i]+beta*p[i];
                }
                nvzk = sqrt(sk_pr(r,r)) / sqrt(sk_pr(F,F));
        }
}

int main()
{
        int i=0;
        int key=0;

        if(input())
                cout << "Input error!" << endl;

        double *tr;

        x       = new double[num_points];
        di      = new double[num_points];
        F       = new double[num_points];
        z    = new double[num_points];
        r    = new double[num_points];
        p    = new double[num_points];
        l    = new double[num_points];
        l1   = new double[num_points];
        f    = new double[num_points];
        tr   = new double[num_points];

        portret();

        for(i=0;i<num_points;i++)
                di[i]=F[i]=x[i]=0;

        ggu=new double[ig[num_points]-1];
        ggl=new double[ig[num_points]-1];

        for(i=0;i<ig[num_points];i++)
                ggu[i]=ggl[i]=0;

        global_matrix();

        LOC();
        for(int i = 0; i < num_points; i++)
                tr[i] = x[i];

        tochnoe();

        ofstream file11("1.txt");
        for(i = 0; i < num_points; i++)
                file11 << setprecision(20) << x[i] << "    " << tr[i] << endl;
        file11.close();

        return 0;
}
```

4.Тестирование

1 тест для общей проверки программы на полиномах первой степени со всеми краевыми условиями



Узлы имеют координаты

0: (0,0)
1: (2,0)
2: (4,0)
3: (0,1)
4: (2,1)
5: (4,1)

Конечные элементы состоят из узлов

0: 0 1 3
1: 1 3 4
2: 1 4 2
3: 2 4 5

Области состоят из конечных элементов

0: 0 1
1: 2 3

Граничные условия на ребрах

0 ,1 первое нулевого типа
1,2 второе нулевого типа
0,3 третье первого типа
3,4 второе второго типа
4,5 третье нулевого типа
2,5 второе первого типа

$$\lambda(x,y)=\begin{cases}1,(x,y)\ принадлежит\ первой\ области\\2,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$\gamma(x,y)=\begin{cases}5,(x,y)\ принадлежит\ первой\ области\\4,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$\beta(x,y)=\begin{cases}5,(x,y)\ принадлежит\ первой\ области\\4,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$f(x,y)=\begin{cases}5x+10y,(x,y)\ принадлежит\ первой\ области\\-2x+8y+12,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$Точное\ решение: u(x,y)=\begin{cases}x+2y,(x,y)\ принадлежит\ первой\ области\\-\frac{x}{2}+2y+3,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$I_0 : u_g = x$

$II_0 : \theta = -4$

$II_1 : \theta = -1$

$II_2 : \theta = 2$

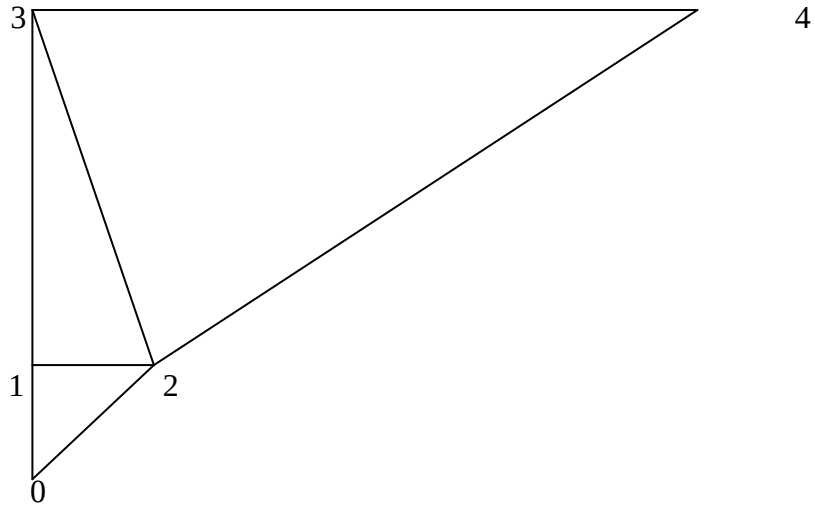
$III_0 : u_\beta = \frac{24 - 2x}{4}$

$III_1 : u_\beta = \frac{10y - 1}{5}$

Точное решение	Полученное решение	Вектор погрешности	Погрешность
0	0,000000000E+00	0,000000000E+00	
2	2,000000000E+00	0,000000000E+00	
1	1,000000130E+00	-1,301999999E-07	8,32159E-06
2	2,000000462E+00	-4,620000000E-07	
4	4,000008291E+00	-8,291000000E-06	
3	3,000000527E+00	-5,270000001E-07	

Линейные базисные функции позволяют получить достаточно точное решение полинома первой степени.

2 тест на полиномах более высокого порядка



Узлы имеют координаты

0: (2,0)
1: (2,1)
2: (3,1)
3: (2,4)
4: (7,4)

Конечные элементы состоят из узлов

0: 0 1 2
1: 1 2 3
2: 2 3 4

Области состоят из конечных элементов

0: 0
1: 1 2

Граничные условия на ребрах

0 ,1 второе первого типа
0,2 первое нулевого типа
1,3 второе первого типа
3,4 второе первого типа
2,4 третье нулевого типа

$$\lambda(x,y)=\begin{cases}10,(x,y)\ принадлежит\ первой\ области\\ 1,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$\gamma(x,y)=0$$

$$\beta(x,y)=2$$

$$f(x,y)=\begin{cases}-20,(x,y)\ принадлежит\ первой\ области\\ 0,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$Точное\ решение: u(x,y)=\begin{cases}y^2,(x,y)\ принадлежит\ первой\ области\\ 20y-19,(x,y)\ принадлежит\ второй\ области\end{cases}$$

$$I_0: u_g = y^2$$

$$II_0: \theta = 20$$

$$II_1: \theta = 0$$

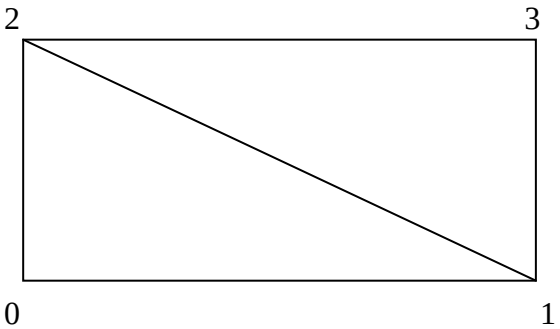
$$III_0: u_\beta = 20y - 27$$

Точное решение	Полученное решение	Погрешность	Норма погрешности
0	0	0	
1	1,1432	0,1432	
1	1	0	1,452E-1
61	61,024	0,024	
61	61,001	0,001	

Полученное решение содержит некоторую погрешность. Это связано с тем, что в качестве аналитического решения рассматриваемой краевой задачи на части расчетной области был взят полином второй степени, который не представим без погрешности в используемом линейном базисе

3,4,5 тесты - на поведение программы при разбиении конечных элементов

3 тест



Узлы имеют координаты

0: (0,0)
1: (2,0)
2: (0,1)
3: (2,1)

Конечные элементы состоят из узлов

0: 0 1 2
1: 1 2 3

Граничные условия на ребрах

0 ,1 первое нулевого типа
1,3 второе нулевого типа
2,3 третье нулевого типа
0,2 второе первого типа

$\lambda(x,y)=2 \qquad \gamma(x,y)=3 \qquad \beta(x,y)=-1$

$f(x,y)=-e^{x+y}$

$Точное\ решение: u(x,y)=e^{x+y}$

$I_0: u_g = e^x$

$II_0: \theta = 2e^{x+y}$

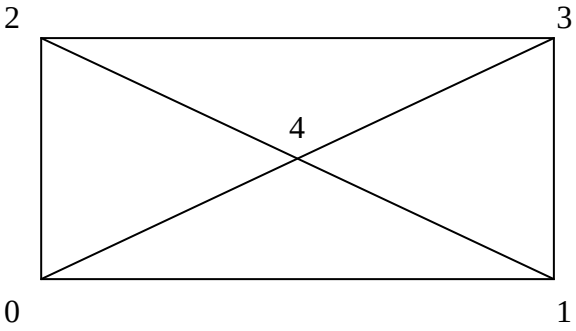
$II_1: \theta = -2e^{x+y}$

$III_0: u_\beta = e^{x+1} - 2e^{x+y}$

Точное решение	Полученное решение	Погрешность	Норма погрешности
1,00E+00	1,00E+00	0,00E+00	
7,39E+00	7,39E+00	0,00E+00	2,70E+00
2,72E+00	3,81E+00	-1,09E+00	
2,01E+01	1,76E+01	2,46E+00	

Теперь следует произвести разбиение каждого конечного элемента на 2, чтобы улучшить решение. Таким образом имеем:

4 тест



Узлы имеют координаты

0: (0,0)
1: (2,0)
2: (0,1)
3: (2,1)
4: (1, 0,5)

Конечные элементы состоят из узлов

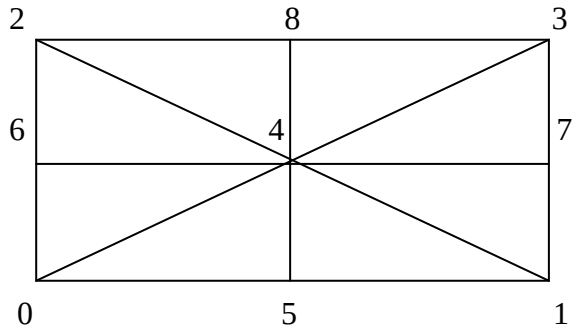
0: 0 1 4
1: 1 3 4
2: 2 3 4
3: 0 2 4

Граничные условия пересчитывать не нужно, так как узлов на ребра не добавилось, следовательно граничные условия будут такими же. Так как количество областей не увеличилось, то коэффициенты γ и β останутся прежними. Прежней останется и функция $f(x,y)=-e^{x+y}$.

Точное решение	Полученное решение	Погрешность	Норма погрешности
1,00E+00	1,00E+00	0,00E+00	
7,39E+00	7,39E+00	0,00E+00	
2,72E+00	3,36E+00	-6,40E-01	7,50E-01
2,01E+01	1,98E+01	3,00E-01	
4,48E+00	4,73E+00	-2,50E-01	

Произведем еще одно разбиение конечных элементов. Так как теперь узлы добавятся на границы области, то следует пересмотреть граничные условия:

5 тест



Узлы имеют координаты

0: (0,0)
1: (2,0)
2: (0,1)
3: (2,1)
4: (1, 0,5)
5: (1,0)
6: (0, 0,5)
7: (2, 0,5)
8: (1,1)

Конечные элементы состоят из узлов

0: 0 4 5
1: 1 4 5
2: 0 4 6
3: 1 4 7
4: 2 4 6
5: 3 4 7
6: 2 4 8
7: 3 4 8

Граничные условия на ребрах

0 ,5 первое нулевого типа
1,5 первое нулевого типа
1,7 второе нулевого типа
3,7 второе нулевого типа
2,8 третье нулевого типа
3,8 третье нулевого типа
0,6 второе первого типа
2,6 второе первого типа

Точное решение	Полученное решение	Погрешность	Норма погрешности
1,00E+00	1,00E+00	0,00E+00	
7,39E+00	7,39E+00	0,00E+00	
2,72E+00	2,47E+00	2,50E-01	
2,01E+01	2,01E+01	4,00E-02	
4,48E+00	4,40E+00	8,00E-02	3,85E-01
2,72E+00	2,72E+00	0,00E+00	
1,65E+00	1,91E+00	-2,60E-01	
1,22E+01	1,21E+01	1,00E-01	
7,39E+00	7,40E+00	-6,00E-03	

Таким образом получаем, что при разбиении сетки погрешность решения уменьшается.