|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
| C:\Users\kimev\AppData\Local\Temp\7zO0E6F35D7\4 ЦВЕТ (1).png | | |
| Кафедра прикладной математики | | |
|  | | |
| Курсовая работа | | |
| по дисциплине «Численные методы» | | |
|  | | |
| Место для ввода текста. | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-71 |
|  | Место для ввода текста. |
| Студент: | Функ Софья |
|  |  |
|  |  |
| Преподаватели: | Патрушев Илья Игоревич |
|  | Персова Марина Геннадьевна |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. Задача

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции билинейные на прямоугольниках. Краевые условия всех типов.

1. Постановка задачи

* Решаемое уравнение

Эллиптическая краевая задача, определяемая дифференциальным уравнением:

− 𝑑𝑖𝑣 (𝜆 𝑔𝑟𝑎𝑑 𝑢)+𝛾𝑢=𝑓

C краевыми условиями:

𝑢|𝑠1=𝑢𝑔- первое краевое условие

𝜆=𝜃 - второе краевое условие

(𝜆+ 𝛽(𝑢−𝑢𝛽))|𝑠3=0 - третье краевое условие

* Краевые условия

Краевые условия первого рода будем учитывать с поправкой на то, что на границе, где указаны краевые условия функции ψ равные нулю. Данные краевые условия реализованы внесения в глобальную матрицу на главную диагональ в соответствующий номер этой диагонали ставится относительно большое число, то есть число намного большее, чем элементы глобальной матрицы. В вектор правой части ставится число, которое соответствует произведению точного значения функции в данном узле на относительно большое число. Такие действия позволяют нам решать обычное линейное уравнение, решением которого является значение функции в данном узле.

Краевые условия второго и третьего родов не учитывались

* Расчётная область

Для того, чтобы реализовать МКЭ для данной задачи Разобьём область на прямоугольные конечные элементы . Базис - билинейные функции. На каждом конечном элементе определим локальные базисные функции так, чтобы каждая из них была равна единице в одном узле и нулю во всех остальных.



Локальные базисные функции имеют вид



1. Теоретическая часть

* Вариационная постановка

Решим уравнение методом Галеркина.

Потребуем невязку:



и чтобы она была ортогональна , в смысле скалярного произведения пространства H0  , некоторому пространству пробных функций, т.е.:



Воспользуемся формулой Грина, которая является обобщением формулы интегрирования по частям для многомерного случая в виде



Интегралы по границам S3 и S2 можно преобразовать, воспользовавшись краевыми условиями

Поскольку на границе S1 краевыми условиями не определяется значение, слагаемое



следует исключить из уравнения, потребовав, чтобы пространство пробных функций содержало только функции, которые бы принимали бы нулевые значения на границе S1 .Поэтому в качестве пространства пробных функций возьмем

Представим в виде линейной комбинации базисных функций , функции  и 



Подставив в уравнение, получим строку СЛАУ для qj

.

* Конечноэлементная дискретизация и переход к локальным матрицам

Далее разобьем область 

Решение задачи будем искать в виде: , перебирая

получим:

где функции  - базисные функции, которыми аппроксимируются функция u и f. После суммирования по всем функциям  получим СЛАУ относительно весов.

в которых символ – символ Кронекера (=1, если j=i, иначе =0)

где матрица А представлена в виде суммы некоторых матриц.

 - матрица жесткости

 - матрица массы

. – вектор правой части.

* Аналитические выражения для вычисления локальных матриц либо схемы численного интегрирования в случае, если интегралы для вычисления локальных матриц предполагается считать численно

Локальные матрицы

– матрица жесткости

– матрица массы

, где

𝜇(𝑖)=((𝑖−1) 𝑚𝑜𝑑 2)+1

𝑣(𝑖)=[(𝑖−1)/2]+1

Матрица жёсткости:



Матрица массы:



Вектор правой части:



1. Описание разработанных программ

* Структуры данных, используемые для задания расчётной области и конечно элементной сетки

Сетку храним в виде списков узлов и списка координат в отдельных массивах.

* Структура основных модулей программы, в том числе генерация портрета СЛАУ, вычисление локальных матриц, генерация глобальных матриц, решение СЛАУ

Портрет глобальной матрицы формируется при помощи дополнительного массива, в который заносятся все связи между узлами (назовем узлы связанными, если существует конечный элемент, содержащий каждый из них), затем под каждую пару связанных узлов выделяется память в глобальной матрице, построенной в разреженном строчном формате.

Для решения СЛАУ используется ЛОС с неполной факторизацией.

1. Описание тестирования программ

* Цель теста

Проверить работоспособность программы на константе с 1 коэффициентами и неравномерной сеткой

, f = 7, u = 7

X Y U U\* U\* - U

1 1 7 7 0

9 1 7 7 0

1 2 7 7 0

9 2 7 7 0

1 3 7 7 0

9 3 7 7 0

1 4 7 7 0

9 4 7 7 0

Вывод

Линейными функциями функция константы приближается без погрешности.

* Цель теста

Проверить работоспособность программы на линейной функции с разными коэффициентами и равномерной сеткой.

, f = 18x+ 9y, u = 2x + y

X Y U U\* U\* - U

2 0 4 4 0

4 0 8 8 0

2 2 6 6 0

4 2 10 10 0

2 4 8 8 0

4 4 12 12 0

Вывод

Линейная функция идеально приближается линейным базисом.

* Цель теста

Посмотреть, как линейный базис приближает полином степени 4

, f = 18x+ 9y, u = 2x + y

X Y U U\* U\* - U

1 1 10 10 0

5 1 1882 1882 0

9 1 19690 19690 0

1 5 134 134 0

5 5 1848.90909090909 2006 157.09090909091

9 5 19814 19814 0

1 9 738 738 0

5 9 2610 2610 0

9 9 20418 20418 0

Видно, что линейными функциями приблизиться к полиному 4 степени не получилось.

1. Приведённые исследования и выводы

Будем делать исследование на полиноме 2 порядка(на 1 порядок выше, чем базисные функции).

, f = 9(y2 +3x2+y)+24 , u = y2 +3x2+y+8

X{1, 13}

Y{1, 4, 16}

X Y U U\* U\* - U

1 1 13 13 0

13 1 517 517 0

1 4 31 31 0

13 4 535 535 0

1 16 283 283 0

13 16 787 787 0

X{1, 7, 13}

Y{1, 2.5, 4, 10, 16}

X Y U U\* U\* - U

1 1 13 13 0

7 1 157 157 0

13 1 517 517 0

1 2.5 19.75 19.75 0

7 2.5 163.75 163.75 0

13 2.5 523.75 523.75 0

1 4 31 31 0

7 4 175 175 0

13 4 535 535 0

1 10 121 121 1.4210854715202e-14

7 10 265 265 0

13 10 625 625 0

1 16 283 283 0

7 16 427 427 0

13 16 787 787 0

X{1, 4, 7, 10, 13}

Y{1, 1.25, 2.5, 3.25, 4, 7, 10, 13, 16}

X Y U U\* U\* - U

1 1 13 13 0

4 1 58 58 0

7 1 157 157 0

10 1 310 310 0

13 1 517 517 0

1 1.25 13.8125 13.8125 0

4 1.25 58.8125 58.8125 2.1316282072803e-14

7 1.25 157.8125 157.8125 0

10 1.25 310.8125 310.8125 0

13 1.25 517.8125 517.8125 0

1 2.5 19.75 19.75 0

4 2.5 64.75 64.75 0

7 2.5 163.75 163.75 5.6843418860808e-14

10 2.5 316.75 316.75 0

13 2.5 523.75 523.75 0

1 3.25 24.8125 24.8125 0

4 3.25 69.8125 69.8125 0

7 3.25 168.8125 168.8125 0

10 3.25 321.8125 321.8125 1.13686837721616e-13

13 3.25 528.8125 528.8125 0

1 4 31 31 0

4 4 76 76 2.8421709430404e-14

7 4 175 175 0

10 4 328 328 0

13 4 535 535 0

1 7 67 67 0

4 7 112 112 2.8421709430404e-14

7 7 211 211 0

10 7 364 364 1.13686837721616e-13

13 7 571 571 0

1 10 121 121 1.4210854715202e-14

4 10 166 166 2.8421709430404e-14

7 10 265 265 5.6843418860808e-14

10 10 418 418 0

13 10 625 625 0

1 13 193 193 0

4 13 238 238 8.5265128291212e-14

7 13 337 337 0

10 13 490 490 0

13 13 697 697 0

1 16 283 283 0

4 16 328 328 0

7 16 427 427 0

10 16 580 580 0

13 16 787 787 0

Относительные погрешности

Е0 = 3,34E-03

Е1 = 2,08E-04

Е2 = 1,31E-05

Погрешность аппроксимации есть О(h2), то есть при дроблении сетки в два раза погрешность должна уменьшаться в 4 раза.

Что и видно из полученных результатов.

1. Текст программы

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <iomanip>

#include <locale.h>

#include "math.h"

using namespace std;

double\* global;

double\* hX, \*hY, \*di;

double\* ggl; // нижний треугольник исходной матрицы

double\* ggu; // верхний треугольник исходной матрицы

double\* x; // вектор решения

double\* f; // вектор правой части

double\* p, \*z, \*q, \*r, \*s, \*S;

double\* L; // нижний треугольник обусловленной матрицы

double\* U; // верхний треугольник обусловленной матрицы

double\* diag; // диагональ обусловленной матрицы

double eps;

int\* ig, \* jg;

int N, Nx, Ny, maxiter;

double Lambda(double x, double y)

{

return 1.0;

}

double Gamma(double x, double y)

{

return 1.0;

}

double FF(double x, double y)

{

return 7;

//return (-18 \* x - 6 \* y);

}

double Real(double x, double y)

{

return 7;

//return (3 \* x \* x \* x + y \* y \* y + 6);

}

void AddToMatrix(int i, int j, double el)

{

int k;

if (i == j)

di[i] += el;

else

{

if (i > j)

{

for (k = ig[i]; k < ig[i + 1]; k++)

if (jg[k] == j)

ggl[k] += el;

}

else

{

for (k = ig[j]; k < ig[j + 1]; k++)

if (jg[k] == i)

ggu[k] += el;

}

}

}

void Matrix()

{

ifstream Reg("Region.txt");

Reg >> Nx >> Ny;

N = Nx \* Ny;

Reg.close();

ifstream inf("Info.txt");

inf >> maxiter >> eps;

inf.close();

}

void ReadH()

{

int i;

ifstream X("hX.txt");

for (i = 0; i < Nx; i++)

X >> hX[i];

X.close();

ifstream Y("hY.txt");

for (i = 0; i < Ny; i++)

Y >> hY[i];

Y.close();

}

void GlobalMatrix()

{

int i, k, i1, k1;

int Index[4];

double lambda, gamma, x, y, xp, yp, hx, hy;

double tmp, hx2, hy2, ud, u1, u2, u3;

double fv[4];

double B[4][4];

double C[4][4];

double F[4];

//Сборка глобальной матрицы

for (k = 0; k < Ny - 1; k++)

{

for (i = 0; i < Nx - 1; i++)

{

x = hX[i];

y = hY[k];

xp = hX[i + 1];

yp = hY[k + 1];

hx = xp - x;

hy = yp - y;

hx2 = hx \* hx;

hy2 = hy \* hy;

lambda = Lambda(x, y);

gamma = Gamma(x, y);

fv[0] = FF(x, y);

fv[1] = FF(xp, y);

fv[2] = FF(x, yp);

fv[3] = FF(xp, yp);

//Задаём значения матрицы жёсткости

tmp = 1. / (hx \* hy);

ud = (hx2 + hy2) \* tmp / 3;

u1 = (hx2 - 2 \* hy2) \* tmp / 6;

u2 = -(2 \* hx2 - hy2) \* tmp / 6;

u3 = -(hx2 + hy2) \* tmp / 6;

B[0][0] = B[1][1] = B[2][2] = B[3][3] = ud;

B[0][1] = B[1][0] = B[2][3] = B[3][2] = u1;

B[0][2] = B[1][3] = B[2][0] = B[3][1] = u2;

B[0][3] = B[1][2] = B[2][1] = B[3][0] = u3;

// Задаём значения матрицы масс

ud = hx \* hy;

C[0][0] = C[1][1] = C[2][2] = C[3][3] = ud / 9.0;

C[0][3] = C[1][2] = C[2][1] = C[3][0] = ud / 36.0;

C[1][0] = C[1][3] = C[2][0] = C[2][3] = C[3][1] = C[3][2] = C[0][1] = C[0][2] = ud / 18.0;

//Задаём значения вектора правой части

F[0] = C[0][0] \* fv[0] + C[0][1] \* fv[1] + C[0][2] \* fv[2] + C[0][3] \* fv[3];

F[1] = C[1][0] \* fv[0] + C[1][1] \* fv[1] + C[1][2] \* fv[2] + C[1][3] \* fv[3];

F[2] = C[2][0] \* fv[0] + C[2][1] \* fv[1] + C[2][2] \* fv[2] + C[2][3] \* fv[3];

F[3] = C[3][0] \* fv[0] + C[3][1] \* fv[1] + C[3][2] \* fv[2] + C[3][3] \* fv[3];

Index[0] = Nx \* k + i;

Index[1] = Nx \* k + i + 1;

Index[2] = Nx \* (k + 1) + i;

Index[3] = Nx \* (k + 1) + i + 1;

for (i1 = 0; i1 < 4; i1++)

{

for (k1 = 0; k1 < 4; k1++)

AddToMatrix(Index[i1], Index[k1], lambda \* B[i1][k1] + gamma \* C[i1][k1]);

f[Index[i1]] += F[i1];

}

}

}

//Краевые условия первого рода

for (i = 0; i < Nx; i++)

{

x = hX[i];

y = hY[0];

di[i] = 1.0e+50;

f[i] = 1.0e+50 \* Real(x, y);

y = hY[Ny - 1];

di[Nx \* (Ny - 1) + i] = 1.0e+50;

f[Nx \* (Ny - 1) + i] = 1.0e+50 \* Real(x, y);

}

for (k = 0; k < Ny; k++)

{

y = hY[k];

x = hX[0];

di[k \* Nx] = 1.0e+50;

f[k \* Nx] = 1.0e+50 \* Real(x, y);

x = hX[Nx - 1];

di[(k + 1) \* Nx - 1] = 1.0e+50;

f[(k + 1) \* Nx - 1] = 1.0e+50 \* Real(x, y);

}

}

double Mult(int i, int j)

{

int k, l, find;

double result;

result = 0.0;

if (i == j)

{

for (k = ig[i]; k < ig[i + 1]; k++)

result += U[k] \* L[k];

}

else

{

if (i > j)

{

for (k = ig[j]; k < ig[j + 1]; k++)

{

find = 0;

for (l = ig[i]; l < ig[i + 1] && find == 0; l++)

{

if (jg[l] == jg[k])

{

result += U[k] \* L[l];

find = 1;

}

}

}

}

else

{

for (l = ig[i]; l < ig[i + 1]; l++)

{

find = 0;

for (k = ig[j]; k < ig[j + 1] && find == 0; k++)

{

if (jg[l] == jg[k])

{

result += U[k] \* L[l];

find = 1;

}

}

}

}

}

return result;

}

void LU()

{

int i, j;

for (i = 0; i < N; i++)

{

for (j = ig[i]; j < ig[i + 1]; j++)

{

L[j] = (ggl[j] - Mult(i, jg[j]));

U[j] = (ggu[j] - Mult(jg[j], i)) / diag[jg[j]];

}

diag[i] = di[i] - Mult(i, i);

}

}

void CalcL(double\* in, double\* out)

{

int i, j;

double result;

for (i = 0; i < N; i++)

{

result = 0;

for (j = ig[i]; j < ig[i + 1]; j++)

{

result += L[j] \* out[jg[j]];

}

out[i] = (in[i] - result) / diag[i];

}

}

void CalcU(double\* in, double\* out)

{

int i, j;

for (i = 0; i < N; i++)

out[i] = in[i];

for (i = N - 1; i >= 0; i--)

{

for (j = ig[i]; j < ig[i + 1]; j++)

{

out[jg[j]] -= U[j] \* out[i];

}

}

}

void MultMatrixOnVector(double\* in, double\* out)

{

int i, j;

double\* out1;

out1 = new double[N];

for (i = 0; i < N; i++)

{

out1[i] = di[i] \* in[i];

for (j = ig[i]; j < ig[i + 1]; j++)

{

out1[i] += ggl[j] \* in[jg[j]];

out1[jg[j]] += ggu[j] \* in[i];

}

}

for (i = 0; i < N; i++)

out[i] = out1[i];

delete[] out1;

}

double ScalarMult(double\* v1, double\* v2)

{

int i;

double result;

result = 0;

for (i = 0; i < N; i++)

{

result += v1[i] \* v2[i];

}

return result;

}

void Go()

{

int;

int i, check, stop = 0, iter;

double alpha, zn, ch, bet, bch, bzn, out;

LU();

for (i = 0; i < N; i++)

x[i] = 0;

CalcL(f, r);

CalcU(r, z);

MultMatrixOnVector(z, q);

CalcL(q, p);

//ЛОС c неполной факторизацией

for (iter = 0; iter < maxiter && stop == 0; iter++)

{

ch = ScalarMult(p, r);

zn = ScalarMult(p, p);

alpha = ch / zn;

for (i = 0; i < N; i++)

x[i] += alpha \* z[i];

for (i = 0; i < N; i++)

r[i] -= alpha \* p[i];

CalcU(r, s);

MultMatrixOnVector(s, S);

CalcL(S, q);

bzn = ScalarMult(p, p);

bch = ScalarMult(p, q);

bet = -bch / bzn;

for (i = 0; i < N; i++)

z[i] = bet \* z[i] + s[i];

for (i = 0; i < N; i++)

p[i] = bet \* p[i] + q[i];

out = ScalarMult(r, r);

if (out < eps)

stop = 1;

}

}

void Output()

{

int i, k, num = 0;

double px, y, norm = 0.0;

ofstream output("result.txt");

output << setw(5) << "X" << setw(10) << "Y" << setw(10) << "U" << setw(10) << "U\*" << setw(22) << "U\* - U" << endl;

for (k = 0; k < Ny; k++)

for (i = 0; i < Nx; i++)

{

px = hX[i];

y = hY[k];

func = Real(px, y);

tmp = fabs(x[num] - func);

output << setw(5) << setprecision(15) << px << setw(10) << y << setw(10) << x[num] << setw(10) << func << setw(22) << tmp << endl;

num++;

}

output.close();

}

void method()

{

int i, k;

// Получение информации о задаче

global = new double[10000];// выделение памяти

memset(global, 0, 10000 \* sizeof(double)); // еe зануление

Matrix();

// Настройка указателей

ig = (int\*)global;

jg = (int\*)(global + N + 1);

//Количество элементов

int istep = 0;

for (i = 0; i < N + 1; i++)

{

ig[i] = istep;

istep += i;

}

// Генерация позиций элементов матрицы

istep = 0;

for (i = 0; i < N; i++)

for (k = 0; k < i; k++)

{

jg[istep] = k;

istep++;

}

// Настройка указателей

ggl = global + N + 1 + ig[N];

ggu = global + 2 \* (ig[N]) + N + 1;

di = global + 3 \* (ig[N]) + N + 1;

f = di + N;

r = f + N;

z = r + N;

p = z + N;

q = p + N;

diag = q + N;

L = diag + N;

U = L + ig[N];

x = U + ig[N];

s = x + N;

S = s + N;

hX = S + N + 5;

hY = hX + Nx;

ReadH();

GlobalMatrix();

}

void main()

{

setlocale(LC\_CTYPE, "Russian");

method();

Go();

Output();

}