|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Курсовая работа 4 | | |
| по дисциплине «Численные методы» | | |
|  | | |
| **.** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-73 |
| Вариант: | 41 |
| Студент: | Дергачев П.А. |
| Преподаватель: | Персова М.Г. |
|  |  |
|  | . |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

**1. Задание**

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции билинейные на прямоугольниках. Краевые условия всех типов. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

**2. Математическая модель**

Пусть дана эллиптическая краевая задача, определяемая дифференциальным уравнением:

*(1)*

C краевыми условиями на границах :

*-* первое краевое условие *(2)*

- второе краевое условие *(3)*

- третье краевое условие *(4)*

**2.1. Вариационная постановка в форме уравнения Галеркина**

Умножим скалярно уравнение *(1)* на пробную функцию  и проинтегрируем обе части уравнения по всей области :

,

Преобразуем первый интеграл по формуле Грина:

Интеграл по границе S разобьем на три интеграла по границам , на которых заданы краевые условия:

, функции построим такие, что на границе они равны 0.

- учет второго краевого условия

- учет третьего краевого условия

В итоге получим:

**2.2 Построение дискретного аналога**

Далее разобьем область 

Решение задачи будем искать в виде: , перебирая

получим:

где функции  есть 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.

**3. Базисные функции**

Разобьём область  на прямоугольники :

Рассмотрим один элемент, пронумеруем вершины следующим образом:

y

1

2

3

4

x

Рассмотрим билинейные базисные функции. Эти функции определяются следующим образом:

, ,

, ,

Локальные базисные функции на конечном элементе представляются в виде:

Таким образом каждая из функций равна 1 в одном из узлов и 0 в остальных. Значит вес  фактически является значением функции  в - ом узле сетки.

**4. Построение локальных матриц и локального вектора правой части.**

Рассмотрим следующие локальные матрицы:

– матрица жесткости

– матрица массы

*Матрица жёсткости:*



*Матрица массы:*



Вектор правой части:



**5.Сборка глобальной матрицы и вектора правой части**

Для занесения результатов сборки локальной матрицы в глобальную необходимо установить соответствие между локальной нумерацией узлов и глобальной.

m

m+1

k

k+1

1

2

3

4

На рисунке показано соответствие между локальной и глобальной нумерациями. Таким образом, изменения глобальной матрицы будут иметь вид:

Аналогично выполняется занесение результатов в вектор правой части.

**6. Краевые условия.**

**6.1.Краевые условия первого рода**.

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 достаточно большое число, а вместо элемента с таким номером в вектор правой части это число, умноженное на значение краевого условия.

Пусть в i-том узле задано первое краевое условие, тогда то -е уравнение СЛАУ примет вид , или .

**6.2.Краевые условия второго рода**

Учет краевого условия второго рода осуществляется за счет добавления в глобальный вектор правой части слагаемого , где ребро  принадлежит конечному элементу  и содержит узлы с номерами ,.

Вклад в вектор правой части будем вычислять по формуле:



После вычисления вектора его компоненты заносятся в глобальный вектор на позиции  и .

**6.3.Краевые условия третьего рода.**

При учете третьих краевых условий формируются локальная матрица и вектор правой части, которые заносятся в СЛАУ аналогично локальной матрицы конечного элемента и локального вектора правой части конечного элемента.

Вклад в матрицу:



Вклад в вектор правой части, при представлении на Г в виде разложения по базисным функциям :



**7. Формирование портрета глобальной матрицы.**

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.

**8. Код программы.**

#include <iostream>

#include <fstream>

#include <iomanip>

#include <locale.h>

#include <memory.h>

#include "math.h"

using namespace std;

#define STR 15 // количество выводимых знаков после запятой

#define EPS 1e-15 // точность решения СЛАУ

#define MAX\_ITER 1000 // количество итераций

#define MEMORY 200000 // объем памяти

typedef double real;

real\* global; // указатель области памяти

real\* GridX; // сетка для Х

real\* GridY; // сетка для У

real\* di; // диагональ исходной матрицы

real\* ggl; // нижний треугольник исходной матрицы

real\* ggu; // верхний треугольник исходной матрицы

real\* r; // вспомогательный вектор

real\* x; // вектор решения

real\* f; // вектор правой части

real\* p; // вспомогательный вектор

real\* z; // вспомогательный вектор

real\* q; // вспомогательный вектор

real\* L; // нижний треугольник обусловленной матрицы

real\* U; // верхний треугольник обусловленной матрицы

real\* diag; // диагональ обусловленной матрицы

real\* s; // вспомогательный вектор

real\* sout; // вспомогательный вектор

real eps; // точность решения

int\* ig; // портрет матрицы

int\* jg; // позиции элементов матрицы

int N; // размерность СЛАУ

int Nx; // количество по Х

int Ny; // количество по Y

int LU(); // функция факторизации

void assembling(); // cборка глобальной матрицы

void get\_info(); // получение параметов

void read\_grid(); // чтение сетки

void AddToMatrix(int, int, real); // добаление в матрицу

real GetLambda(real, real); // коэффициент лямбда

real GetGamma(real, real); // коэффициент гамма

real GetF(real, real); // правая часть

real GetU\_(real, real); // точное решение

void GaussL(real\*, real\*); // Решение СЛАУ

void GaussU(real\*, real\*); // Решение СЛАУ

void MultMatrixOnVector(real\*, real\*); // Умножение матрицы на вектор

real ScalarMult(real\*, real\*); // Скалярное произведение векторов

real sum(int, int); // Скалярное произведение факторизации

void method(); // Сборка метода

void run(); // Выполнение метода

void result(); // Вывод результата в файл

// Задание парамметров и функции

real GetLambda(real x, real y)

{

return 1.0;

}

real GetGamma(real x, real y)

{

return 0.;

}

real GetF(real x, real y) // F

{

return 1.;

}

real GetU\_(real x, real y) // U

{

return x+y;

}

void method()

{

int i, k;

// Получение информации о задаче

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

memset(global, 0, MEMORY \* sizeof(double)); // заполнение нулями

get\_info();

// Настройка указателей

ig = (int\*)global;

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

// Генерация количества элементов матрицы

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;

sout = s + N;

GridX = sout + N;

GridY = GridX + Nx;

// Чтение сетки

read\_grid();

// Сборка глобальной матрицы

assembling();

}

void get\_info()

{

ifstream file("Area.txt");

file >> Nx >> Ny;

N = Nx \* Ny; // размер матрицу СЛАУ

file.close();

}

void read\_grid()

{

// Чтение сетки по оси Х

ifstream flie\_x("GridX.txt");

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

flie\_x >> GridX[i];

flie\_x.close();

// Чтение сетки по оси У

ifstream flie\_y("GridY.txt");

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

flie\_y >> GridY[i];

flie\_y.close();

}

void assembling()

{

int i, k, i1, k1;

int Index[4]; // номера узлов

real lambda, gamma, px, y, xp, yp, hx, hy; // параметры КЭ

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

real fv[4]; // значения правой части

real B[4][4]; // матрица жёсткости

real C[4][4]; // матрица масс

real F[4]; // вектор правой части

// Процедура ассемблирования глобальной матрицы

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

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

{

px = GridX[i]; // опорная точка

y = GridY[k];

// верхние точки

xp = GridX[i + 1];

yp = GridY[k + 1];

// шаги

hx = xp - px;

hy = yp - y;

hx2 = hx \* hx;

hy2 = hy \* hy;

// коэффициенты

lambda = GetLambda(px + hx / 2.0, y + hy / 2.0);

gamma = GetGamma(px + hx / 2.0, y + hy / 2.0);

// значения правой части

fv[0] = GetF(px, y);

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

fv[2] = GetF(px, yp);

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

// Задаём значения матрицы жёсткости

tmp = 1.0 / (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] = ud;

B[0][1] = u1;

B[0][2] = u2;

B[0][3] = u3;

B[1][0] = u1;

B[1][1] = ud;

B[1][2] = u3;

B[1][3] = u2;

B[2][0] = u2;

B[2][1] = u3;

B[2][2] = ud;

B[2][3] = u1;

B[3][0] = u3;

B[3][1] = u2;

B[3][2] = u1;

B[3][3] = ud;

// Задаём значения матрицы масс

ud = hx \* hy / 9.0;

u1 = hx \* hy / 18.0;

u2 = u1;

u3 = hx \* hy / 36.0;

C[0][0] = ud;

C[0][1] = u1;

C[0][2] = u2;

C[0][3] = u3;

C[1][0] = u1;

C[1][1] = ud;

C[1][2] = u3;

C[1][3] = u2;

C[2][0] = u2;

C[2][1] = u3;

C[2][2] = ud;

C[2][3] = u1;

C[3][0] = u3;

C[3][1] = u2;

C[3][2] = u1;

C[3][3] = ud;

//Задаём значения вектора правой части

for (int j = 0; j++; j < 4)

F[j] = 0;

for (int j = 0; j < 4; j++)

for (int k = 0; k < 4; k++)

F[j] += C[j][k] \* fv[k];

//Сборка глобальной матрицы

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];

}

}

// Учёт краевых условий первого рода

for (i = 0; i < Nx; i++) // параллельно оси Х

{

px = GridX[i];

y = GridY[0];

di[i] = 1.0e+50;

f[i] = 1.0e+50 \* GetU\_(px, y);

y = GridY[Ny - 1];

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

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

}

for (k = 0; k < Ny; k++) // параллельно оси Y

{

y = GridY[k];

px = GridX[0];

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

f[k \* Nx] = 1.0e+50 \* GetU\_(px, y);

px = GridX[Nx - 1];

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

f[(k + 1) \* Nx - 1] = 1.0e+50 \* GetU\_(px, y);

}

}

// Добавление элементов в матрицу

void AddToMatrix(int i, int j, real 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;

}

}

}

//-----------------LOC-------------------

// Факторизация

int LU()

{

int i, j;

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

{

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

{

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

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

}

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

}

return 0;

}

// Решение нижнего треугольника

void GaussL(real\* in, real\* out)

{

int i, j;

real 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];

}

}

// Решение верхнего треугольника

void GaussU(real\* in, real\* 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];

}

}

}

// Умножение матрицы на вектор

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

{

int i, j;

real\* out1;

out1 = new real[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;

}

// Скалярное произведение

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

{

int i;

real result;

result = 0;

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

{

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

}

return result;

}

// Скалярное произведение для факторизации

real sum(int i, int j)

{

int k, l, find;

real result;

result = 0.0;

if (i == j)

{

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

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

}

else

{

// верхний треугольник

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;

}

}

}

}

// нижний треугольник

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;

}

// Основная функция метода

void run()

{

int iter;

int i, check, stop;

real alpha, alphazn, alphach, beta, betach, betazn, CheckExit;

// Факторизация

check = LU();

if (check != 0) cout << "Нельзя выполнить факторизацию" << check + 1 << endl;

// Инициализация

stop = 0;

for (i = 0; i < N; i++) x[i] = 0;

GaussL(f, r);

GaussU(r, z);

MultMatrixOnVector(z, q);

GaussL(q, p);

// Процесс решения СЛАУ

for (iter = 0; iter < MAX\_ITER && stop == 0; iter++)

{

alphach = ScalarMult(p, r);

alphazn = ScalarMult(p, p);

alpha = alphach / alphazn;

for (i = 0; i < N; i++) x[i] += alpha \* z[i];

for (i = 0; i < N; i++) r[i] -= alpha \* p[i];

GaussU(r, s);

MultMatrixOnVector(s, sout);

GaussL(sout, q);

betazn = ScalarMult(p, p);

betach = ScalarMult(p, q);

beta = -betach / betazn;

//GaussU(r,s);

for (i = 0; i < N; i++) z[i] = beta \* z[i] + s[i];

for (i = 0; i < N; i++) p[i] = beta \* p[i] + q[i];

CheckExit = ScalarMult(r, r);

if (CheckExit < EPS) stop = 1;

}

}

//---------------------------------------------------------

void result()

{

int i, k, num = 0;

real px, y, func, res = 0.0, norm = 0.0, tmp;

ofstream output("result.txt");

output << "X" << setw(STR + 8) << "Y" << setw(STR + 8)

<< "U" << setw(STR + 8) << "U\*" << setw(STR + 8) << "U\* - U" << endl;

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

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

{

px = GridX[i];

y = GridY[k];

func = GetU\_(px, y);

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

res += tmp \* tmp;

norm += func \* func;

output << setprecision(STR) << px

<< setw(STR + 8) << y

<< setw(STR + 8) << x[num]

<< setw(STR + 8) << func;

output << setw(STR + 8) << tmp << endl;

num++;

}

output << "\n ||U-U\*|| / ||U\*|| = " << scientific << sqrt(res / norm) << endl;

output.close();

}

// Главная функция

int main()

{

setlocale(LC\_CTYPE, "russian");

method();

run();

result();

return 0;

}

**9. Тестирование.**

*Тест 1*

Условия:

  

Исходная функция:



Сетка: Y={0;1;2} X={0;2;4;6}

Цель теста:

Проверка правильного составления матрицы жесткости с постоянным коэффициентом.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | 2,00E-50 | 0 | 2,00E-50 | 5,02E-17 |
| 2 | 0 | 4 | 4 | 0 |
| 4 | 0 | 16 | 16 | 0 |
| 6 | 0 | 36 | 36 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 5 | 5 | 8,88E-16 |
| 4 | 1 | 17 | 17 | 3,55E-15 |
| 6 | 1 | 37 | 37 | 0 |
| 0 | 2 | 4 | 4 | 0 |
| 2 | 2 | 8 | 8 | 0 |
| 4 | 2 | 20 | 20 | 0 |
| 6 | 2 | 40 | 40 | 0 |

*Тест 2*

Условия:

 

Исходная функция:



Сетка: Y={0;1;2} X={0;2;4;6}

Цель теста:

Проверка совместного составления матрицы жесткости и матрицы масс.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | 1,17E-50 | 0 | 1,17E-50 | 1,60E-52 |
| 2 | 0 | 4 | 4 | 0 |
| 4 | 0 | 16 | 16 | 0 |
| 6 | 0 | 36 | 36 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 5 | 5 | 0 |
| 4 | 1 | 17 | 17 | 0 |
| 6 | 1 | 37 | 37 | 0 |
| 0 | 2 | 4 | 4 | 0 |
| 2 | 2 | 8 | 8 | 0 |
| 4 | 2 | 20 | 20 | 0 |
| 6 | 2 | 40 | 40 | 0 |

*Тест 3*

Условия:

 

Исходная функция:



Сетка: Y={0;1;2} X={0;2;4;6}

Цель теста:

Проверка точного приближения базисными функциями полинома третьего порядка.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | 1,50E-50 | 0 | 1,50E-50 | 7,17E-17 |
| 2 | 0 | 8 | 8 | 0 |
| 4 | 0 | 64 | 64 | 0 |
| 6 | 0 | 216 | 216 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 9 | 9 | 1,78E-15 |
| 4 | 1 | 65 | 65 | 2,84E-14 |
| 6 | 1 | 217 | 217 | 0 |
| 0 | 2 | 8 | 8 | 0 |
| 2 | 2 | 16 | 16 | 0 |
| 4 | 2 | 72 | 72 | 0 |
| 6 | 2 | 224 | 224 | 0 |

Вывод:

Действительно, полиномы третьего порядка приближаются с точностью до 13 знаков базисными функциями на любой сетке.

*Тест 4*

Условия:

 

Исходная функция:



Цель теста:

Проверка приближения бикубическими функциями полиномы четвертого порядка.

Сетка 1: Y={0;1;2} X={0;2;4;6}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | 2,73E-50 | 0 | 2,73E-50 | 3,35E-03 |
| 2 | 0 | 16 | 16 | 0 |
| 4 | 0 | 256 | 256 | 0 |
| 6 | 0 | 1296 | 1296 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 11,545455 | 17 | 5,45E+00 |
| 4 | 1 | 251,54545 | 257 | 5,45E+00 |
| 6 | 1 | 1297 | 1297 | 0 |
| 0 | 2 | 16 | 16 | 0 |
| 2 | 2 | 32 | 32 | 0 |
| 4 | 2 | 272 | 272 | 0 |
| 6 | 2 | 1312 | 1312 | 0 |

Сетка 2: Y={0; 0.5; 1; 1.5 ;2} X={0;1;2;3;4;5;6}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | -2,12E-52 | 0 | 2,12E-52 | 2,09E-04 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 16 | 16 | 0 |
| 3 | 0 | 81 | 81 | 0 |
| 4 | 0 | 256 | 256 | 0 |
| 5 | 0 | 625 | 625 | 5,45E+00 |
| 6 | 0 | 1296 | 1296 | 5,45E+00 |
| 0 | 0,5 | 0,0625 | 0,0625 | 0 |
| 1 | 0,5 | 0,261734 | 1,0625 | 8,01E-01 |
| 2 | 0,5 | 15,14864 | 16,0625 | 9,14E-01 |
| 3 | 0,5 | 80,13150 | 81,0625 | 9,31E-01 |
| 4 | 0,5 | 255,1486 | 256,0625 | 9,14E-01 |
| 5 | 0,5 | 624,2617 | 625,0625 | 8,01E-01 |
| 6 | 0,5 | 1296,063 | 1296,063 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0,957945 | 2 | 1,04E+00 |
| 2 | 1 | 15,78054 | 17 | 1,22E+00 |
| 3 | 1 | 80,76026 | 82 | 1,24E+00 |
| 4 | 1 | 255,7805 | 257,0000 | 1,22E+00 |
| 5 | 1 | 624,9579 | 626 | 1,04E+00 |
| 6 | 1 | 1297 | 1297 | 0 |
| 0 | 1,5 | 5,0625 | 5,0625 | 0 |
| 1 | 1,5 | 5,2617 | 6,0625 | 8,01E-01 |
| 2 | 1,5 | 20,14864 | 21,0625 | 9,14E-01 |
| 3 | 1,5 | 85,13150 | 86,06250 | 9,31E-01 |
| 4 | 1,5 | 260,1486 | 261,0625 | 9,14E-01 |
| 5 | 1,5 | 629,2617 | 630,0625 | 8,01E-01 |
| 6 | 1,5 | 1301,063 | 1301,063 | 0 |
| 0 | 2 | 16 | 16 | 0 |
| 1 | 2 | 17 | 17 | 0 |
| 2 | 2 | 32 | 32 | 0 |
| 3 | 2 | 97 | 97 | 0 |
| 4 | 2 | 272 | 272 | 0 |
| 5 | 2 | 641 | 641 | 0 |
| 6 | 2 | 1312 | 1312 | 0 |

Сетка 3: Y={0; 0.25; 0.5; 0.75; 1; 1.25; 1.5; 1.75; 2}

X={0; 0.5; 1; 1.5; 2; 2.5; 3; 3.5; 4; 4.5; 5; 5.5; 6}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***X*** | ***Y*** | ***U*** | ***U\**** | ***U\* - U*** | **ll*U\*-U*ll / ll*U*\*ll** |
| 0 | 0 | -1,48E-52 | 0 | 1,48E-52 | 1,30E-05 |
| 0,5 | 0 | 0,0625 | 0,0625 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1,5 | 0 | 5,0625 | 5,0625 | 0 |
| 2 | 0 | 16 | 16 | 0 |
| 2,5 | 0 | 39 | 39 | 0 |
| 3 | 0 | 81 | 81 | 0 |
| 3,5 | 0 | 150,063 | 150,063 | 0 |
| 4 | 0 | 256 | 256 | 0 |
| 4,5 | 0 | 410,063 | 410,063 | 0 |
| 5 | 0 | 625 | 625,0000 | 0 |
| 5,5 | 0 | 915,063 | 915,063 | 0 |
| 6 | 0 | 1296,000 | 1296,000 | 0 |
| 0 | 0,25 | 0,003906 | 0,003906 | 0 |
| 0,5 | 0,25 | -0,015992 | 0,066406 | 0,082398 |
| 1 | 0,25 | 0,891582 | 1,003910 | 1,12E-01 |
| 1,5 | 0,25 | 4,940540 | 5,066410 | 1,26E-01 |
| 2 | 0,25 | 15,87220 | 16 | 1,32E-01 |
| 2,5 | 0,25 | 38,93220 | 39,06640 | 1,34E-01 |
| 3 | 0,25 | 80,86910 | 81,00390 | 1,35E-01 |
| 3,5 | 0,25 | 149,9320 | 150,0660 | 0,134179 |
| 4 | 0,25 | 255,8720 | 256,0040 | 0,131749 |
| 4,5 | 0,25 | 409,9410 | 410,0660 | 1,26E-01 |
| 5 | 0,25 | 624,8920 | 625,0040 | 1,12E-01 |
| 5,5 | 0,25 | 914,9840 | 915,0660 | 8,24E-02 |
| 6 | 0,25 | 1296 | 1296 | 0 |
| 0 | 0,5 | 0,0625 | 0,062500 | 0 |
| 0,5 | 0,5 | -0,007883 | 0,125000 | 0,132883 |
| 1 | 0,5 | 0,873026 | 1,062500 | 0,189474 |
| 1,5 | 0,5 | 4,910710 | 5,125000 | 0,214295 |
| 2 | 0,5 | 15,83730 | 16,06250 | 0,225196 |
| 2,5 | 0,5 | 38,89530 | 39,12500 | 0,229681 |
| 3 | 0,5 | 80,83160 | 81,06250 | 0,230895 |
| 3,5 | 0,5 | 149,8950 | 150,1250 | 0,229681 |
| 4 | 0,5 | 255,8370 | 256,0630 | 0,225196 |
| 4,5 | 0,5 | 409,9110 | 410,1250 | 0,214295 |
| 5 | 0,5 | 624,8730 | 625,0630 | 0,189474 |
| 5,5 | 0,5 | 914,9920 | 915,1250 | 0,132883 |
| 6 | 0,5 | 1296,060 | 1296,060 | 0 |
| 0 | 0,75 | 0,316406 | 0,316406 | 0 |
| 0,5 | 0,75 | 0,218237 | 0,378906 | 0,16067 |
| 1 | 0,75 | 1,082180 | 1,316410 | 0,23423 |
| 1,5 | 0,75 | 5,112160 | 5,378910 | 0,266746 |
| 2 | 0,75 | 16,03540 | 16,31640 | 0,280974 |
| 2,5 | 0,75 | 39,09210 | 39,37890 | 0,286837 |
| 3 | 0,75 | 81,02800 | 81,31640 | 0,288422 |
| 3,5 | 0,75 | 150,0920 | 150,3790 | 0,286837 |
| 4 | 0,75 | 256,0350 | 256,3160 | 0,280974 |
| 4,5 | 0,75 | 410,1120 | 410,3790 | 0,266746 |
| 5 | 0,75 | 625,0820 | 625,3160 | 0,23423 |
| 5,5 | 0,75 | 915,2180 | 915,3790 | 0,16067 |
| 6 | 0,75 | 1296,320 | 1296,320 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0,5 | 1 | 0,892925 | 1,0625 | 0,169575 |
| 1 | 1 | 1,751120 | 2 | 0,248883 |
| 1,5 | 1 | 5,778380 | 6,0625 | 0,284121 |
| 2 | 1 | 16,70050 | 17 | 0,299516 |
| 2,5 | 1 | 39,75660 | 40,0625 | 0,305862 |
| 3 | 1 | 81,69240 | 82 | 0,307579 |
| 3,5 | 1 | 150,7570 | 151,063 | 0,305862 |
| 4 | 1 | 256,7000 | 257 | 0,299516 |
| 4,5 | 1 | 410,7780 | 411,063 | 0,284121 |
| 5 | 1 | 625,7510 | 626 | 0,248883 |
| 5,5 | 1 | 915,8930 | 916,063 | 0,169575 |
| 6 | 1 | 1297 | 1297 | 0 |
| 0 | 1,25 | 2,441410 | 2,441410 | 0 |
| 0,5 | 1,25 | 2,343240 | 2,503910 | 0,16067 |
| 1 | 1,25 | 3,207180 | 3,441410 | 0,23423 |
| 1,5 | 1,25 | 7,237160 | 7,503910 | 0,266746 |
| 2 | 1,25 | 18,16040 | 18,44140 | 0,280974 |
| 2,5 | 1,25 | 41,21710 | 41,50390 | 0,286837 |
| 3 | 1,25 | 83,15300 | 83,44140 | 0,288422 |
| 3,5 | 1,25 | 152,2170 | 152,5040 | 0,286837 |
| 4 | 1,25 | 258,1600 | 258,4410 | 0,280974 |
| 4,5 | 1,25 | 412,2370 | 412,5040 | 0,266746 |
| 5 | 1,25 | 627,2070 | 627,4410 | 0,23423 |
| 5,5 | 1,25 | 917,3430 | 917,5040 | 0,16067 |
| 6 | 1,25 | 1298,440 | 1298,440 | 0 |
| 0 | 1,5 | 5,062500 | 5,062500 | 0 |
| 0,5 | 1,5 | 4,992120 | 5,125000 | 0,132883 |
| 1 | 1,5 | 5,873030 | 6,062500 | 0,189474 |
| 1,5 | 1,5 | 9,910710 | 10,12500 | 0,214295 |
| 2 | 1,5 | 20,83730 | 21,06250 | 0,225196 |
| 2,5 | 1,5 | 43,89530 | 44,12500 | 0,229681 |
| 3 | 1,5 | 85,83160 | 86,06250 | 0,230895 |
| 3,5 | 1,5 | 154,8950 | 155,1250 | 0,229681 |
| 4 | 1,5 | 260,8370 | 261,0630 | 0,225196 |
| 4,5 | 1,5 | 414,9110 | 415,1250 | 0,214295 |
| 5 | 1,5 | 629,8730 | 630,0630 | 0,189474 |
| 5,5 | 1,5 | 919,9920 | 920,1250 | 0,132883 |
| 6 | 1,5 | 1301,060 | 1301,060 | 0 |
| 0 | 1,75 | 9,378910 | 9,378910 | 0 |
| 0,5 | 1,75 | 9,359010 | 9,441410 | 0,082398 |
| 1 | 1,75 | 10,26660 | 10,37890 | 0,112324 |
| 1,5 | 1,75 | 14,31550 | 14,44140 | 0,125867 |
| 2 | 1,75 | 25,24720 | 25,37890 | 0,131749 |
| 2,5 | 1,75 | 48,30720 | 48,44140 | 0,134179 |
| 3 | 1,75 | 90,24410 | 90,37890 | 0,134835 |
| 3,5 | 1,75 | 159,3070 | 159,4410 | 0,134179 |
| 4 | 1,75 | 265,2470 | 265,3790 | 0,131749 |
| 4,5 | 1,75 | 419,3160 | 419,4410 | 0,125867 |
| 5 | 1,75 | 634,2670 | 634,3790 | 0,112324 |
| 5,5 | 1,75 | 924,3590 | 924,4410 | 0,082398 |
| 6 | 1,75 | 1305,380 | 1305,380 | 0 |
| 0 | 2 | 16 | 16 | 0 |
| 0,5 | 2 | 16,0625 | 16,0625 | 0 |
| 1 | 2 | 17 | 17 | 0 |
| 1,5 | 2 | 21,0625 | 21,0625 | 0 |
| 2 | 2 | 32 | 32 | 0 |
| 2,5 | 2 | 55,0625 | 55,0625 | 0 |
| 3 | 2 | 97 | 97 | 0 |
| 3,5 | 2 | 166,063 | 166,063 | 0 |
| 4 | 2 | 272 | 272 | 0 |
| 4,5 | 2 | 426,063 | 426,063 | 0 |
| 5 | 2 | 641 | 641 | 0 |
| 5,5 | 2 | 931,063 | 931,063 | 0 |
| 6 | 2 | 1312 | 1312 | 0 |