# Задание

Написать программу, на языке С++ в среде VisualStudio “**. Модель, реализующая матрицу Якоби с граничными условиями (состояние пространства за пределом матрицы принимается равным 0)”:**

а) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) и удерживаются в течение всего времени моделирования.

б) начальное воздействие прикладывается к выбранной точке (координаты точки и величина воздействия являются входными данными) в первом такте моделирования и затем снимается.

# Модель, реализующая матрицу Якоби с граничными условиями

Итерационные вычисления Якоби предназначены для нахождения решения дифференциальных уравнений Лапласа и представляют собой последовательность итераций вычисления значений ячеек новой матрицы равной усреднённому значения соседних ячеек предшествующей матрицы.

Итерационный процесс повторяется пока отличие новой матрицы от предшествующей станет незначительным.

В программе реализована работа с многомерными матрицами, хранящаяся в памяти в виде одномерной последовательности чисел.

Количество пространственных измерений и размерность массивов по каждому из измерений запрашивается при запуске программы.

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Модель, реализующая матрицу Якоби с граничными условиями

n - количество измерений

\*m - количество точек по каждому из измерений

\*x - вектор координат точки воздействия

value - величина воздействия

mode - режим (0 - непрерывное воздействе, 1 - однократное воздействие)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int jacobi\_field(int n,int \*m,int \*x,double value,int mode, int demo){

double epsilon=1e-7; // минимальное изменение для продолжения итераций

int nmax=1000; // максимальное число итераций

int total = 1; for(int i = 0; i < n; i++) total\*=m[i];

double \*prevMatrix = new double[total]; // Многомерные массивы хранятся в виде одномерной последовательности

double \*nextMatrix = new double[total]; // Многомерные массивы хранятся в виде одномерной последовательности

if(prevMatrix==NULL) return -1; // Ошибка аллокирования памяти

if(nextMatrix==NULL) return -1; // Ошибка аллокирования памяти

for(int i = 0; i < total; i++) prevMatrix[i]=0;

int index = indexOf(x,m,n); // индекс точки воздействия

prevMatrix[index] = value;

for(int t=0; t<nmax; t++) {

if(demo!=0) cout << "\tШаг итерации:" << t << endl;

if(demo!=0) {

cout << "\tИсходное состояние матрицы:" << endl;

for(int i = 0; i < total; i++) {

cout << prevMatrix[i] << '\t';

if(i%m[0]==m[0]-1) cout << endl;

}

}

// Шаг итераций

#pragma omp parallel for

for(int i = 0; i < total; i++) {

int \*x0 = new int[n];

int \*x1 = new int[n];

nextMatrix[i]=0;

vectorOf(i, x0, m, n);

// Вычисляем среднее соседних точек к точке x0 по каждому из измерений

for(int j=0; j<n; j++) {

if(x0[j]+1<m[j]) {

for(int j1=0; j1<n; j1++) x1[j1]=(j1!=j)?x0[j1]:(x0[j]+1);

nextMatrix[i] += prevMatrix[indexOf(x1, m, n)];

}

else

nextMatrix[i] += 0; // состояние пространства за пределом матрицы принимается равным 0

if(x0[j]-1>=0) {

for(int j1=0; j1<n; j1++) x1[j1]=(j1!=j)?x0[j1]:(x0[j]-1);

nextMatrix[i] += prevMatrix[indexOf(x1, m, n)];

}

else

nextMatrix[i] += 0; // состояние пространства за пределом матрицы принимается равным 0

}

nextMatrix[i] /= 2\*n;

delete x0;

delete x1;

}

if(demo!=0) {

cout << "\tНовое состояние матрицы:" << endl;

for(int i = 0; i < total; i++) {

cout << nextMatrix[i] << '\t';

if(i%m[0]==m[0]-1) cout << endl;

}

}

// Расчёт разницы (кроме точки воздействия при непрерывном воздействии)

double delta = 0;

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

if(i==index&&mode==0) continue;

else if(delta<abs(nextMatrix[i]-prevMatrix[i]))

delta=abs(nextMatrix[i]-prevMatrix[i]);

if(demo!=0) cout << "\tМаксимальное изменение матрицы:" << delta << endl;

if(demo!=0) {

cout << "\tНажмите ВВОД для продолжения" << endl;

getchar();

getchar();

}

// Проверка условия прекращения итераций

if(delta<epsilon) break;

double \*temp = prevMatrix; prevMatrix = nextMatrix; nextMatrix = temp;

if(mode==0) prevMatrix[index] = value;

}

delete prevMatrix;

delete nextMatrix;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Преобразование вектора координат в индекс одномерного массива

int \*x; // координаты точки

int \*m; // количество точек по каждому из измерений

int n; // количество измерений

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int indexOf(int \*x, int \*m, int n) {

int index = 0;

for(int i=n;--i>0;) {

index += x[i];

index \*= m[i-1];

}

index += x[0];

return index;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Преобразование индекса в одномерном массиве в вектор координат

int index - индекс в одномерном массиве

int \*x; // координаты точки

int \*m; // количество точек по каждому из измерений

int n; // количество измерений

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int \*vectorOf(int index, int \*x, int \*m, int n) {

for(int i=n;--i>0;) {

x[i] = index/m[i-1];

index -= x[i]\*m[i-1];

}

x[0] = index;

return x;

# }

# Контрольные примеры работы программы

Пример 1.

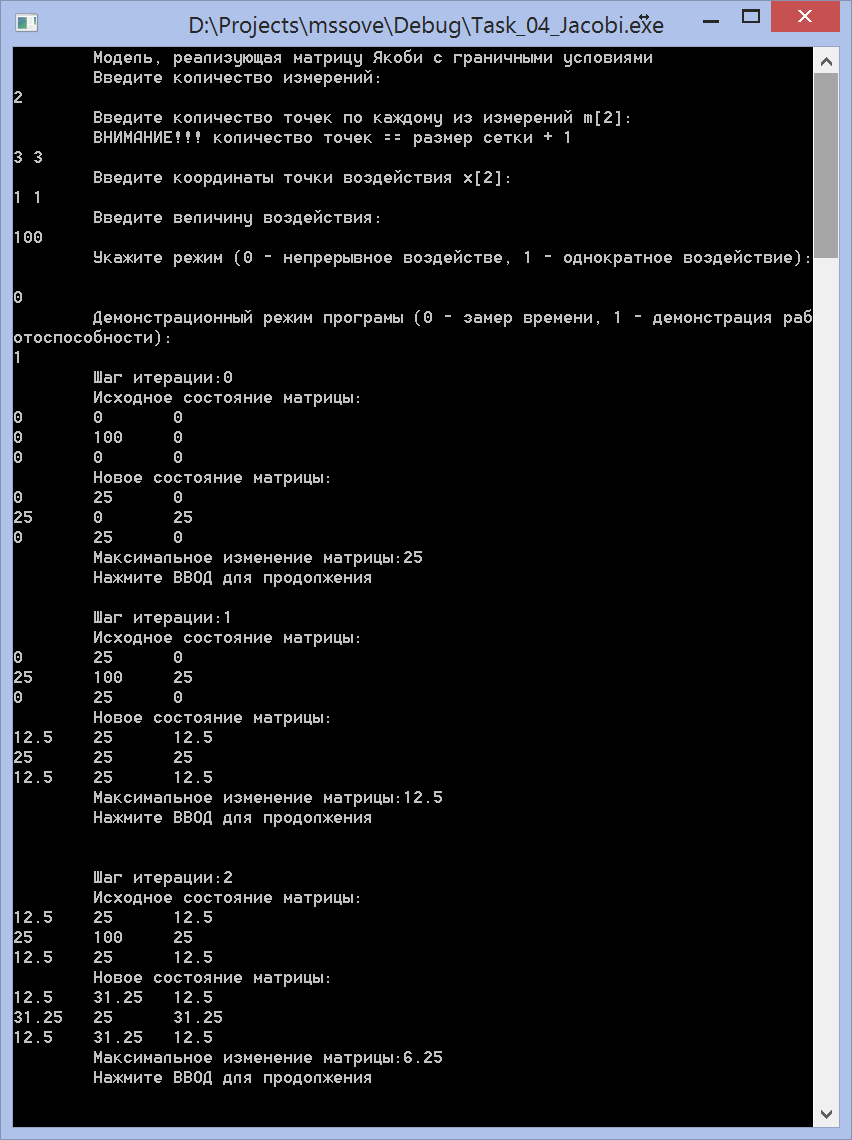


Рис.1..

# Заключение

В ходе данной лабораторной был изучены алгоритм итерационных вычислений Якоби для нахождения решения дифференциальных уравнений Лапласа. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.