**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»**

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

**ОТЧЕТ ПО РАБОТЕ НА ТЕМУ:**

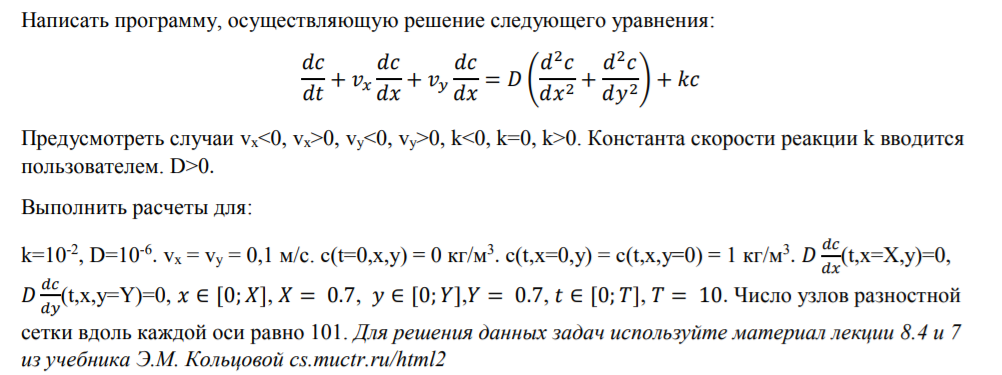
**«Решение схемой расщепления двумерного дифференциального уравнения параболического с помощью технологии OpenMP»**

**СТУДЕНТКА группы КС-40** **Терехова** **О. И.**

**Москва**

**2019**

**Задание**

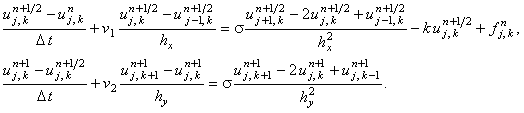




Данное уравнение относится к двумерным дифференциальным уравнениям параболического типа. В то же время оно содержит производные первого порядка по координатам х и у.



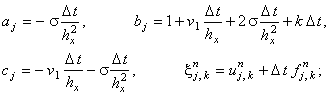
для которого запишем схему расщепления, соблюдая правило выбора конечных разностей для аппроксимации производных.



Каждая из подсхем является абсолютно устойчивой и решается с помощью метода прогонки. Коэффициенты, соответствующие уравнению:

, имеют вид:

* для первой подсхемы



* для второй подсхемы



Легко видеть, что для обеих подсхем достаточное условие сходимости прогонки выполняется:



Рекуррентное прогоночное соотношение:



1. **Определение прогоночных коэффициентов на 1-м шаге по координате**

Для определения прогоночных коэффициентов на 1-м шаге по координате х используют рекуррентное прогоночное соотношение



и левое граничное условие (1 рода):



Сравнивая эти два соотношения, получаем:



Пусть задано левое граничное условие 2-го рода:



Запишем его аппроксимацию и выразим значение :



Сравнивая данное выражение с соотношением (4.14), получаем:



1. **Определение решения на правой границе**

Значение функции u на (n + 1)-ом шаге по времени в крайней справа точке можно определить из правого граничного условия 1 рода:



Для расчёта решения на правой границе используют рекуррентное прогоночное соотношение, записанное для j = N - 1:



Пусть задано правое граничное условие 2-го рода:



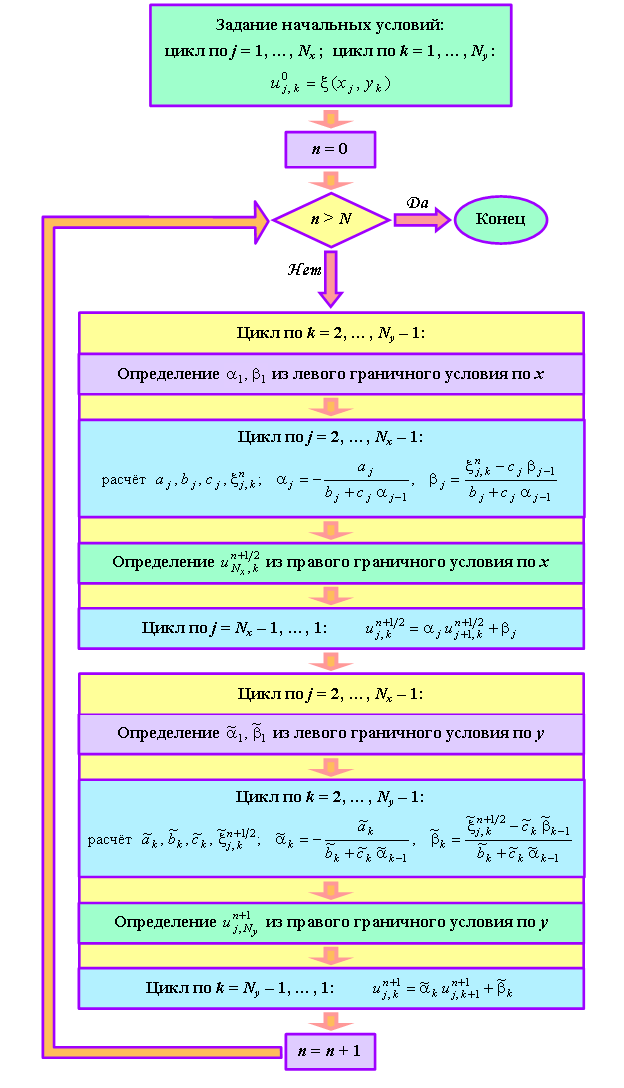
Запишем его аппроксимацию:



Подставляя соотношение (4.15) в данное равенство и выражая , получаем:



**Алгоритм решения:**



КОД ПРОГРАММЫ

Файл Header.h (используется для настройки параметров)

|  |
| --- |
| #include <iostream>  #include <omp.h>  #include <fstream>  #include <math.h>  #include <string>  using namespace std;  const int count\_parallel = 1;  const double X = 0.7; // [a;b] = [0;X]  const double Y = 0.7; // [c;d] = [0;Y]  const double T = 10; // [0;T]  const int Nx = 101; // число узлов по х  const int Ny = 101; // число узлов по у  const int Nt = 101; // число узлов по t  const double U0 = 0; // c(t=0,x,y)  const int type[4] = { 1, 2, 1, 2 }; // род граничных условий {1;2}  const double f[4] = { 1, 0, 1, 0 }; // массив граничных условий (a, b, c, d)  const double K = 0;  const double D = 0.000001;  const double Vx = 0.1;  const double Vy = 0.1; |

Файл Application.cpp (код программы)

|  |
| --- |
| #include "Header.h"  double U[Nx][Ny]; // массив решений n  double U05[Nx][Ny]; // массив решений n+0.5  double U1[Nx][Ny]; // массив решений n+1  void print(double t, double U[Nx][Ny]) {  ofstream fout;  string name = "t=" + to\_string(t);  name += ".csv";  fout.open(name);  for (int i = 0; i < Nx; i++) {  for (int j = 0; j < Ny; j++) {  fout << U[i][j] << "; ";  }  fout << endl;  }  fout.close();  }  int main()  {  double start = omp\_get\_wtime();  double x[Nx]; // массив значений х  double hx = (X / (Nx - 1)); // шаг по х  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int i = 0; i < Nx; i++) {  x[i] = i \* hx;  }  double y[Ny]; // массив значений y  double hy = (Y / (Ny - 1)); // шаг по y  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int i = 0; i < Ny; i++) {  y[i] = i \* hy;  }  double t[Nt]; // массив значений y  double ht = (T / (Nt - 1)); // шаг по t  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int i = 0; i < Nt; i++) {  t[i] = i \* ht;  }  for (int j = 0; j < Nx; j++) {  for (int k = 0; k < Ny; k++) {  U[j][k] = U0;  }  }  print(0, U);  for (int n = 0; n < Nt-1; n++) {  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int k = 0; k < Ny ; k++) {  double alpha[Nx - 1];  double beta[Nx - 1];  if (type[0] == 1) {  alpha[0] = 0;  beta[0] = f[0];  }  else {  alpha[0] = 1;  beta[0] = -hx \* f[0];  }  for (int j = 1; j < Nx - 1; j++) {  double a = -D \* ht / (hx \* hx);  double b = 1 + Vx \* ht / hx + 2 \* D \* ht / (hx \* hx) + K \* ht;  double c = -Vx \* ht / hx + a;  double E = U[j][k];  alpha[j] = -a / (b + c \* alpha[j - 1]);  beta[j] = (E - c \* beta[j - 1]) / (b + c \* alpha[j - 1]);  }  if (type[1] == 1) U05[Nx - 1][k] = f[1]; else U05[Nx - 1][k] = (hx \* f[1] + beta[Nx - 2]) / (1 - alpha[Nx - 2]);  for (int j = Nx-2; j >= 0 ; j--) {  U05[j][k] = alpha[j] \* U05[j + 1][k] + beta[j];  }  }  print((n+0.5) \* ht, U05);  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int j = 0; j < Nx; j++) {  double alpha[Ny - 1];  double beta[Ny - 1];  if (type[2] == 1) {  alpha[0] = 0;  beta[0] = f[2];  }  else {  alpha[0] = 1;  beta[0] = -hy \* f[2];  }  for (int k = 1; k < Nx - 1; k++) {  double a = - Vy \* ht / hy - D \* ht / (hy \* hy);  double b = 1 + Vy \* ht / hy + 2 \* D \* ht / (hy \* hy);  double c = -D \* ht / (hy \* hy);  double E = U05[j][k];  alpha[k] = -a / (b + c \* alpha[k - 1]);  beta[k] = (E - c \* beta[k - 1]) / (b + c \* alpha[k - 1]);  }  if (type[2] == 1) U1[j][Ny-1] = f[3]; else U1[j][Ny-1] = (hx \* f[3] + beta[Ny - 2]) / (1 - alpha[Ny - 2]);  for (int k = Ny - 2; k >= 0; k--) {  U1[j][k] = alpha[k] \* U1[j][k+1] + beta[k];  }  }    print((n + 1)\*ht, U1);  #pragma omp parallel for schedule(dynamic) num\_threads(count\_parallel)  for (int i = 0; i < Nx; i++) {  for (int j = 0; j < Ny; j++) {  U[i][j] = U1[i][j];  }  }  }  double end = omp\_get\_wtime();  cout << end - start << endl;  return 0;  } |

Результаты: