|  |  |  |
| --- | --- | --- |
|  | | |
| **Лабораторная работа 2** | | |
| по дисциплине «Разработка программного обеспечения для моделирования физических процессов» | | |
| Выполнил |  |  |
| студент гр.5130904/10101 |  | Абраамян А. М. |
| Руководитель |  | Воскобойников С. П. |
|  |  |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_202\_\_ г.

**Оглавление**

[Постановка задачи 3](#_heading=h.gjdgxs)

[**Дискретная модель 3**](#_heading=h.xtqrqbywwlgo)

[**Коэффициенты 6**](#_heading=h.5i1g1i9itkvm)

[**Решение системы ОДУ 7**](#_heading=h.lyfcb93xzy1h)

[**Жёсткость системы 8**](#_heading=h.gzcuwsg39jk2)

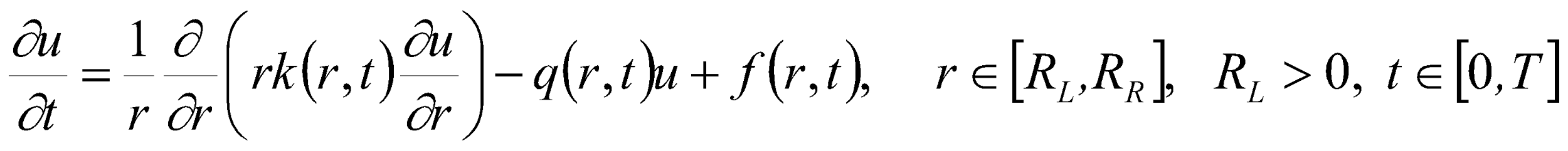
[**Тестирование 8**](#_heading=h.nqferwl8vjnt)

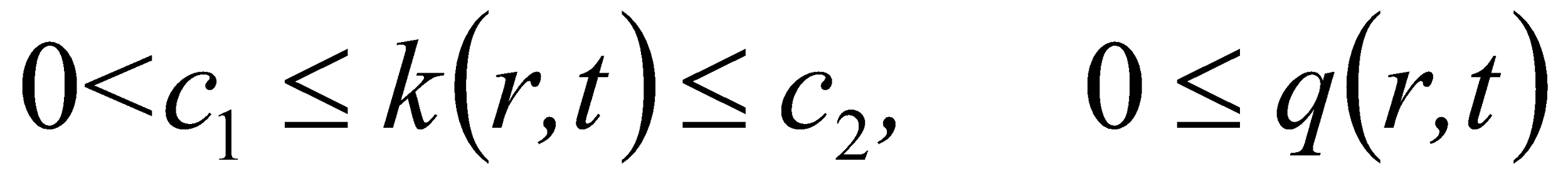
[**Вывод 10**](#_heading=h.n95xg3s13qs)

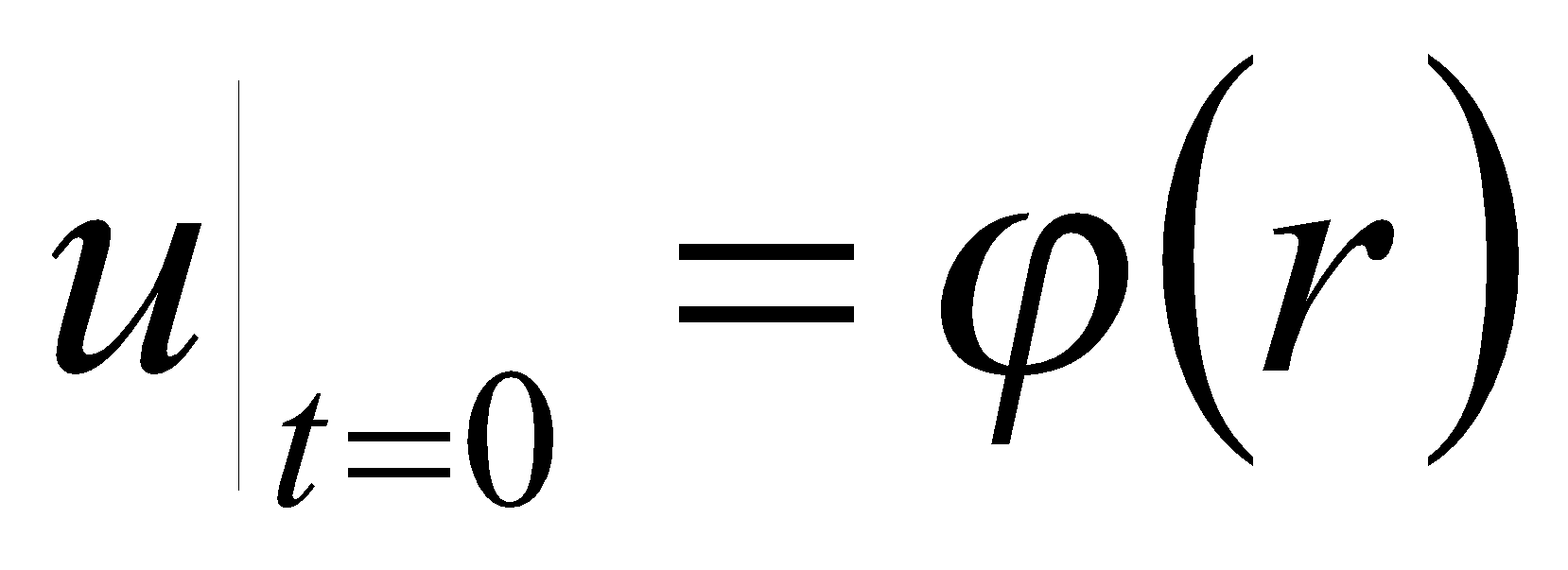
[**Код 10**](#_heading=h.jx9ef4etswmi)

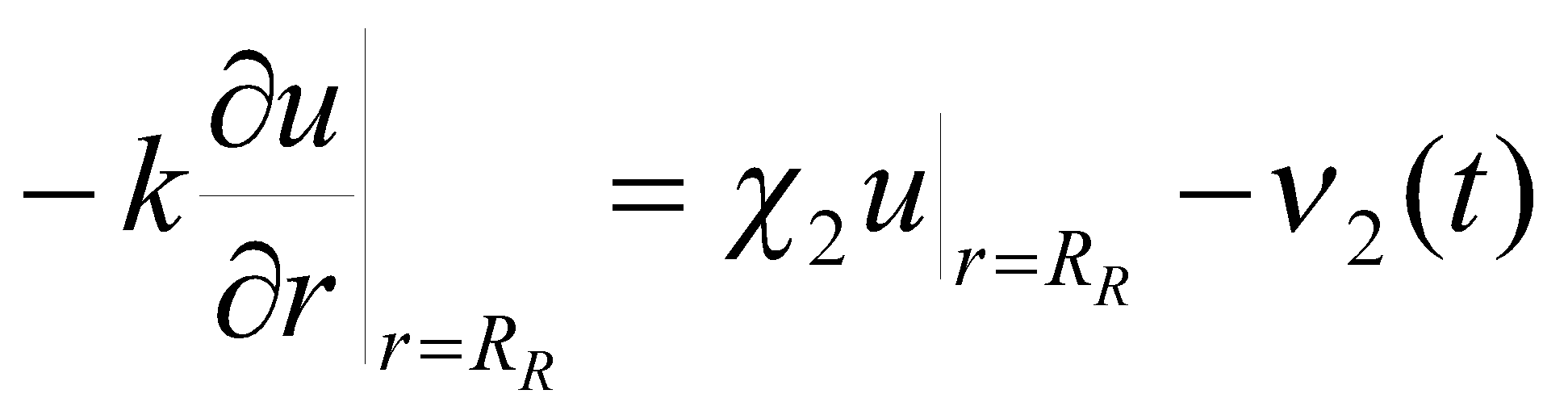
# Постановка задачи

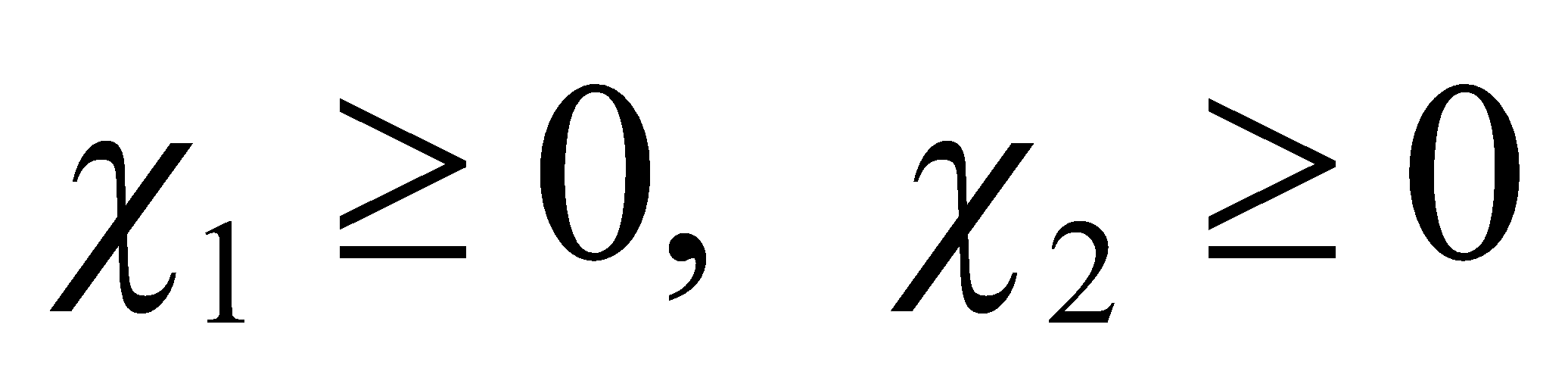
Используя интегро-интерполяционный метод (метод баланса), разработать программу для моделирования нестационарного распределения температуры в полом цилиндре, описываемого математической моделью вида:





начальным условием вида и граничными условиями вида:





Для построения и тестирования модели будет использоваться язык C++.

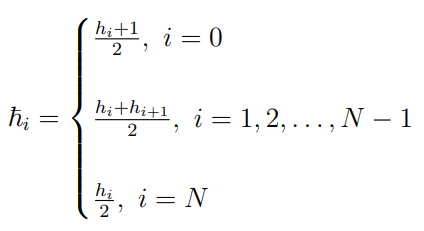
# Дискретная модель

Введём обозначения:

*N -* число разбиений интервала [Rl, Rr]

*hi* = *ri − ri−*1

*ri−1/2*=(*ri + ri−*1)*/2*



Домножим уравнение на r:

**Проинтегрируем уравнение для промежутка, не включая границы:**

*i = 1,2,...,N-1*

Интегро-дифференциальное тождество:

По формуле центральных разностей:

По формуле **средних** прямоугольников:

Разностная схема:

*i = 1,2,...,N-1*

Аппроксимация граничного условия справа:

*i = N*

Теперь используем формулу **правых** прямоугольников для аппроксимации интеграла:

Используя наше граничное условие справа:

*i =N*

Приведём подобные слагаемые для разностных схем:

*i = 1,2,...,N-1*

*i = N*

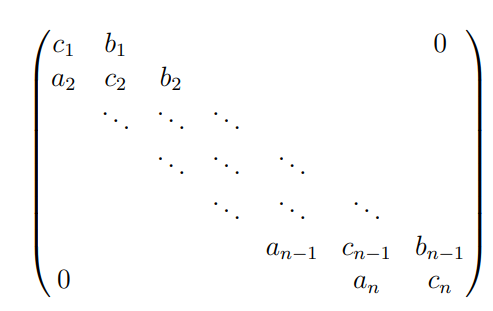
Поскольку нам дано граничное условие слева первого рода, то мы поступим следующим образом. Используем его для уравнение i = 1. Буквально возьмем и подставим вместо . Таким образом наш коэффициент просто перейдет просто перейдет в коэффициент как константа, а матрица станет на одну строчку короче чем была прежде.

# Коэффициенты

Теперь данную систему можно представить в виде:

(1)

Где *A* - трёхдиагональная матрица вида:



Коэффициенты соответственно следующие:

Для коэффициентов у нас не будет.

Решить систему численным методом - найти вектор *v* и сравнить его с точным решением *u.*

# Решение системы ОДУ

Чтобы решить систему дифференциальных уравнений, введём дискретизацию по времени и проинтегрируем:



Добавим начальное условие и получим систему:

**Явный метод ломаных Эйлера**

Аппроксимируем интеграл по формуле левых прямоугольников , получаем:

**Неявный метод ломаных Эйлера**

Аппроксимируем интеграл по формуле правых прямоугольников , получаем:



# Жёсткость системы

При решении явным методом нужно учитывать, что система может быть жёсткой. В этом случае накладывается ограничение на шаг интегрирования 𝛕.

Это ограничение зависит от обусловленности матрицы, которое зависит от шага разбиение интервала по *r.*

При превышении этого ограничения погрешность станет накапливаться с очень большой скоростью (решение неустойчиво).

# Тестирование

## Пример 1

Первый тест будет выглядеть как на рисунке. Предварительно ожидается что компьютер без проблем справится с поставленной задачей. Зададим .

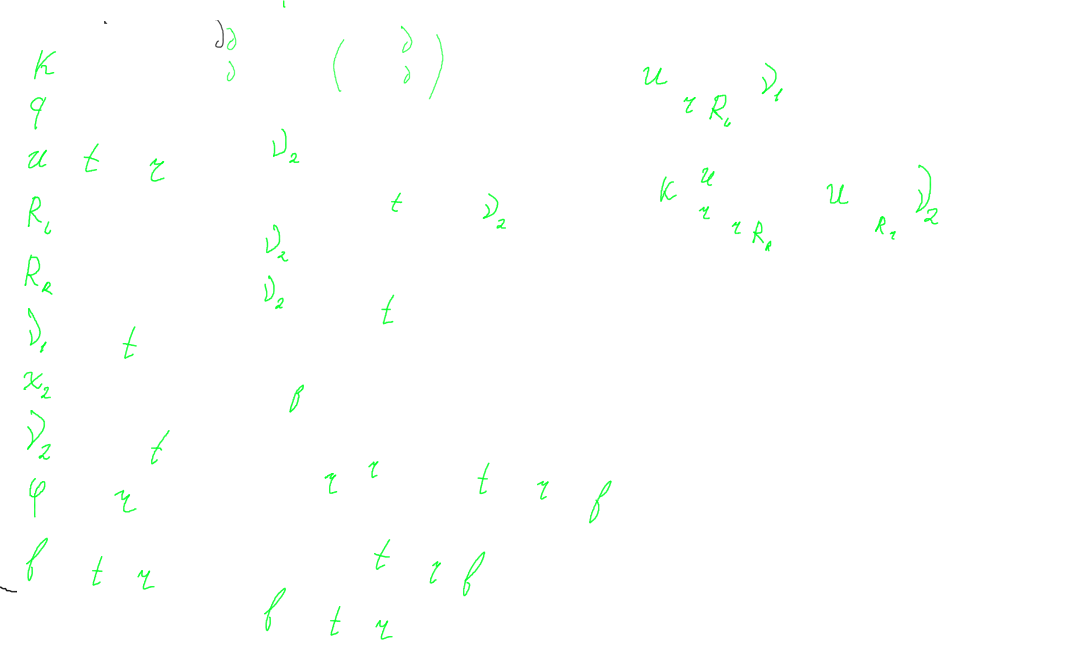


Таблица погрешностей:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Количество разбиений** | | **Размер шага** | | **Явный метод** | **Неявный метод** | **Рунге Кутты 4** |
| **R** | **T** | **R** | **T** |
| 51 | 51 | 0.18 | 0.02 | 1.82123e+251 | 1.82123e+251 | 4.68941e-09 |
| 102 | 102 | 0.0891089 | 0.0099 | inf | 0.000687675 | 6.82364e-09 |
| 202 | 202 | 0.0447761 | 0.004975 | 1.85778 | 0.00036922 | 3.72354e-09 |
| 302 | 302 | 0.0299003 | 0.003322 | 0.577381 | 0.000114426 | 1.14929e-09 |
| 402 | 402 | 0.0224439 | 0.002493 | 0.141115 | 2.81023e-05 | 2.81472e-10 |
| 502 | 502 | 0.0179641 | 0.001996 | 0.0302932 | 6.06209e-06 | 6.05029e-11 |
| 1002 | 1002 | 0.00899101 | 0.000999 | 6.05196e-06 | 1.21087e-09 | 1.20886e-14 |

## Пример 2

**Наблюдения:**

Во всех тестах решение явным методом при большом шаге интегрирования идёт с высокой погрешностью. Это связано с жесткостью системы и неустойчивости решения, которая чем больше, тем больше обусловленность матрицы A. Но при уменьшении шага, после прохождения ограничения погрешность уменьшается и достигает своего определённого предела, который зависит от точности аппроксимации уравнения, которая зависит от числа обусловленности матрицы A.

Во 2 тесте функция u не зависит от r, роль погрешности аппроксимации уменьшается, поэтому с уменьшением шага уменьшается и погрешность.

При решении неявным методом погрешность сразу невелика, поскольку он предназначен для решения жёстких систем, и не имеет ограничение на шаг интегрирования. Но при этом для его исполнения нужно находить обратную матрицу, что для больших размерностей матрицы A является очень трудоёмкой задачей.

# Вывод

Задание выполнено в полном объеме. Была сделана модель нестационарного одномерного распределения температуры. Модель была аппроксимирована с помощью интегро-интерполяционного метода и решена с помощью явного и неявного методов ломанных Эйлера. Была посчитана погрешность моделирования и выявлено, что для решения жёстких систем нужно использовать неявный метод, но нужно помнить, что при этом увеличивается трудоёмкость процесса.

# Код

#include <iostream>

#include <fstream>

#include <cmath>

#include <numbers>

const int N = 8;

const double Rl = std::numbers::pi;

const double Rr = 2 \* std::numbers::pi;

const double T0 = 0;

const double T1 = 1;

const double X1 = 1;

const double X2 = 1;

double H = (Rr - Rl) / N;

double U\_arr[N + 1];

double V\_arrTn[N + 1];

double K\_arr[N];

double Q\_arr[N + 1];

double R\_arr[N + 1];

double Rd\_arr[N];

double H\_arr[N + 1];

double Hs\_arr[N + 1];

double A[N + 1][N + 1];

double A\_inv[N + 1][N + 1];

double G[N + 1];

double U(double r, double t)

{

//return r;

//return 4 + pow(std::numbers::e, -10 \* t);

return r + t;

}

double K(double r)

{

return cos(r) / 2 + 3;

}

double Q(double r)

{

return sin(r) / 2 + 2;

}

double phi(double r)

{

//return r;

//return 5;

return r;

}

double a(int index)

{

return (Rd\_arr[index - 1] \* K\_arr[index - 1]) / (H\_arr[index] \* Hs\_arr[index] \* R\_arr[index]);

}

double b(int index)

{

return (Rd\_arr[index] \* K\_arr[index]) / (H\_arr[index] \* Hs\_arr[index] \* R\_arr[index]);

}

double c(int index)

{

if (index == 0)

{

return -b(index) - X1 / Hs\_arr[index] - Q\_arr[index];

}

else if (index == N)

{

return -a(index) - X2 / Hs\_arr[index] - Q\_arr[index];

}

else

{

return -a(index) - b(index) - Q\_arr[index];

}

}

double v1(double t)

{

//return 0.5;

//return 4 + pow(std::numbers::e, -10 \* t);

return std::numbers::pi + t - 2.5;

}

double v2(double t)

{

//return 5.5;

//return 4 + pow(std::numbers::e, -10 \* t);

return std::numbers::pi + t + 3.5;

}

double f(double r, double t)

{

//return (-cos(r)+ r \* sin(r)-6+pow(r,2)\*sin(r) + 4 \* pow(r,2))/(2\*r);

//return (-16 + 4 \* pow(std::numbers::e, 10 \* t) \* sin(r) + sin(r) + 16 \* pow(std::numbers::e, 10 \* t)) / (2 \* pow(std::numbers::e, 10 \* t));

return 1 - (cos(r)-r\*sin(r)+6)/(2\*r) + (t\*sin(r)+sin(r))/2 + 2\*t + 2\*r;

}

double g(int index, double t)

{

if (index == 0)

{

return v1(t) / Hs\_arr[index] + f(R\_arr[index], t);

}

else if (index == N)

{

return v2(t) / Hs\_arr[index] + f(R\_arr[index], t);

}

else

{

return f(R\_arr[index], t);

}

}

void fill\_K()

{

for (size\_t i = 0; i < N; i++)

{

K\_arr[i] = K(Rl + H \* i);

}

}

void fill\_Q()

{

for (size\_t i = 0; i < N + 1; i++)

{

Q\_arr[i] = Q(Rl + H \* i);

}

}

void fill\_R()

{

for (size\_t i = 0; i < N + 1; i++)

{

R\_arr[i] = Rl + H \* i;

}

}

void fill\_Rd()

{

for (size\_t i = 0; i < N; i++)

{

Rd\_arr[i] = (R\_arr[i] + R\_arr[i + 1]) / 2;

}

}

void fill\_H()

{

for (size\_t i = 0; i < N + 1; i++)

{

H\_arr[i] = H;

}

}

void fill\_Hs()

{

Hs\_arr[0] = H\_arr[1] / 2;

Hs\_arr[N] = H\_arr[N] / 2;

for (size\_t i = 1; i < N; i++)

{

Hs\_arr[i] = (H\_arr[i] + H\_arr[i + 1]) / 2;

}

}

void fill\_A(double ht)

{

A[0][0] = c(0) \* ht + 1;

A[0][1] = b(0) \* ht;

for (size\_t i = 1; i < N; i++)

{

A[i][i - 1] = a(i) \* ht;

A[i][i] = c(i) \* ht + 1;

A[i][i + 1] = b(i) \* ht;

}

A[N][N - 1] = a(N) \* ht;

A[N][N] = c(N) \* ht + 1;

}

void fill\_A\_Back(double ht)

{

A[0][0] = 1 - c(0) \* ht;

A[0][1] = -b(0) \* ht;

for (size\_t i = 1; i < N; i++)

{

A[i][i - 1] = -a(i) \* ht;

A[i][i] = 1 - c(i) \* ht;

A[i][i + 1] = -b(i) \* ht;

}

A[N][N - 1] = -a(N) \* ht;

A[N][N] = 1 - c(N) \* ht;

}

void fill\_G(double ht, double t)

{

for (size\_t i = 0; i < N + 1; i++)

{

G[i] = g(i, t) \* ht;

}

}

double eps = 0;

void sum\_vectors(double f[], double s[], int size)

{

for (size\_t i = 0; i < size; i++)

{

f[i] = f[i] + s[i];

}

}

void mul\_matrix\_vector(double matrix[N + 1][N + 1], double vector[], int size)

{

double temp\_vector[N+1];

for (size\_t i = 0; i < size; i++)

{

double sum = 0;

for (size\_t j = 0; j < size; j++)

{

sum += matrix[i][j] \* vector[j];

}

temp\_vector[i] = sum;

}

for (size\_t i = 0; i < size; i++)

{

vector[i] = temp\_vector[i];

}

}

void getCofactor(double matrix[N + 1][N + 1], double temp[N + 1][N + 1], int p, int q, int n)

{

int i = 0, j = 0;

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

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

if (row != p && col != q) {

temp[i][j++] = matrix[row][col];

if (j == n - 1) {

j = 0;

i++;

}

}

}

}

}

double determinant(double matrix[N + 1][N + 1], int n) {

double det = 0;

if (n == 1) {

return matrix[0][0];

}

double temp[N + 1][N + 1];

int sign = 1;

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

getCofactor(matrix, temp, 0, f, n);

det += sign \* matrix[0][f] \* determinant(temp, n - 1);

sign = -sign;

}

return det;

}

void getAdjugate(double matrix[N + 1][N + 1], double adj[N + 1][N + 1], int n) {

if (n == 1) {

adj[0][0] = 1;

return;

}

int sign = 1;

double temp[N + 1][N + 1];

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

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

getCofactor(matrix, temp, i, j, n);

sign = ((i + j) % 2 == 0) ? 1 : -1;

adj[j][i] = sign \* determinant(temp, n - 1); // Транспонированный адъюгат

}

}

}

bool inverse(double matrix[N + 1][N + 1], double inverseMatrix[N + 1][N + 1], int n) {

double det = determinant(matrix, n);

if (det == 0) {

std::cout << "Обратная матрица не существует!" << std::endl;

return false;

}

double adj[N + 1][N + 1]; // Адъюгат

getAdjugate(matrix, adj, n);

// Делим каждый элемент адъюгата на детерминант

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

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

inverseMatrix[i][j] = adj[i][j] / det;

}

}

return true;

}

// Функция для обращения матрицы

void check\_eps(int index, double r, double t)

{

double dif = fabs(U(r, t) - V\_arrTn[index]);

if (dif > eps)

{

eps = dif;

//std::cout << " " << eps << "\n";

}

}

void show\_array(double\* arrayv, int size)

{

std::cout.setf(std::ios::scientific);

for (size\_t i = 0; i < size; i++)

{

std::cout << arrayv[i] << " ";

}

std::cout << std::endl;

}

void show\_matrix(double matrix[N + 1][N + 1], int size)

{

for (size\_t i = 0; i < size; i++)

{

show\_array(matrix[i], size);

}

}

void reset()

{

eps = 0;

for (size\_t i = 0; i < N + 1; i++)

{

V\_arrTn[i] = phi(R\_arr[i]);

}

}

void program()

{

std::cout.setf(std::ios::scientific);

for (size\_t M = 10; M <= 1000000; M \*= 10)

{

reset();

//show\_array(V\_arrTn, N+1);

double Ht = (T1 - T0) / M;

//fill\_A(Ht);

fill\_A\_Back(Ht);

//show\_matrix(A, N+1);

inverse(A, A\_inv, N + 1);

//show\_matrix(A\_inv, N+1);

double t = 0;

for (size\_t j = 1; j <= M; j++)

{

t += Ht;

fill\_G(Ht, t);

//show\_array(G, N+1);

//Явный метод

//mul\_matrix\_vector(A, V\_arrTn, N+1);

//sum\_vectors(V\_arrTn, G, N+1);

//show\_array(V\_arrTn, N+1);

//Неявный метод

//show\_array(G, N+1);

sum\_vectors(V\_arrTn, G, N + 1);

mul\_matrix\_vector(A\_inv, V\_arrTn, N + 1);

//show\_array(V\_arrTn, N+1);

for (size\_t i = 0; i < N + 1; i++)

{

check\_eps(i, R\_arr[i], t);

}

}

std::cout << "H = " << Ht << "\tEps = " << eps << "\n";

}

}

int main()

{

fill\_K();

fill\_Q();

fill\_R();

fill\_Rd();

fill\_H();

fill\_Hs();

program();

return 0;

}