**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4**

по курсу «Численные методы».

Тема: «Методы решения обыкновенных дифференциальных уравнений».

Студент: Якимович А.И.

Группа: 80-308Б

Вариант: 1

Оценка:

Москва, 2017

Постановка задачи.

Реализовать методы для

решения задачи Коши:

1. метод Эйлера
2. метод Рунге-Кутты четвёртого порядка
3. метод Адамса четвёртого порядка

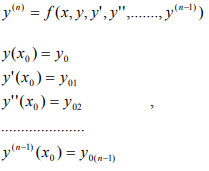
решения краевой задачи ОДУ:

1. метод стрельбы
2. метод конечных разностей

Описание методов.

**Задача Коши.**

Заданы обыкновенное дифференциально уравнение второго порядка *y’’ = f(x, y, y’)* и значения *y(a)* и *y’(a)* на границе *a* отрезка *[a,b].* Требуется найти сеточную функцию *y* с шагом *h* на отрезке *[a, b].* Сначала сведём ОДУ второго порядка к системе ОДУ первого порядка с помощью введения новых переменных



Далее мы можем представить систему в векторном виде и применять заданные методы, подставляя векторы в качестве переменных

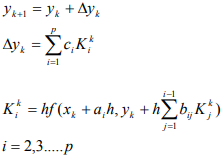


1. Редко используется на практике из-за невысокой точности. В случае небольшого шага разностной сетки h график функции и график касательной не успевают сильно разойтись друг от друга и можно в качестве значения решения в узле принять значение касательной , вместо значения неизвестного точного решения . При этом допускается погрешность. Повторяя все предыдущие рассуждения, получим значения в остальных точках.

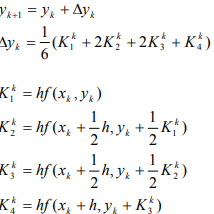
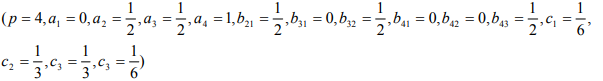
Итерационная формула . Погрешность на каждом шаге . Также контроль точности можно осуществлять методом Румбе-Ромберга или путём сравнения с точным результатом.



1. Часто используется на практике. Семейство явных методов Рунге-Кутты *р*-го порядка записывается в виде совокупности формул:



Для четвёртого порядка:



Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

1. Решение дифференциального уравнения *y’ =f(x, y)* удовлетворяет интегральному соотношению



Если решение задачи Коши получено в узлах вплоть до *k*-го, то можно аппроксимировать подынтегральную функцию, например: интерполяционным многочленом какой-либо степени. Вычислив интеграл от построенного многочлена на отрезке, получим ту или иную формулу Адамса. Чтобы получить метод четвёртого порядка, будем использовать многочлен третей степени, получим итерационную формулу

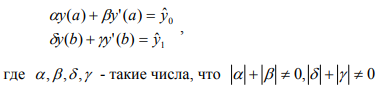


Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

**Краевая задача.**

Заданы обыкновенное дифференциально уравнение второго порядка *y’’ = f(x, y, y’)* и условия на концах отрезка *[a, b]*

*.*



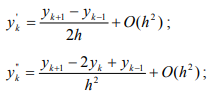
Сведём ОДУ второго порядка к системе ОДУ первого порядка так же, как сделали это для задачи Коши.

1. Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи. Вместо исходной задачи формулируется задача Коши, условие на правом конце заменяется на ещё одним условием на левом (y’(a) = η). Решается задача Коши, полученное решение на правом конце сравнивается с отброшенным краевым условием. По результатам сравнения делается корректировка η.

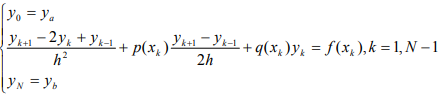
решение исходной задачи эквивалентно нахождению корня уравнения *Φ(η) = 0* , где *.* Так как невозможно вычислить производную функции Φ(η), будем пользоваться методом секущих. Следующее значение корня определяется соотношением . Условие окончания . Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.



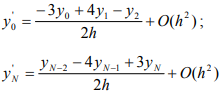
1. Введем разностную аппроксимацию производных следующим образом:



Подставим в исходное уравнение, получим:



Если мы используем аппроксимацию производных на краях с помощью односторонних разностей второго порядка, мы теряем трехдиагональную структуру матрицы коэффициентов, но сохраняем второй порядок аппроксимации:



Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

Общая информация.

Данная работа состоит из девяти модулей, которые позволяют решать различные задачи решения СЛАУ и нахождения собственных векторов и собственных значений. Полученные в ходе расчетов результаты сохраняются в отдельный файл. Что касается технических деталей реализации, все программы написаны на языке C++.

Запуск программы.

Чтобы воспользоваться программой, необходимо собрать проект с помощью утилиты make (или nmake на windows) и запустить полученный файл prog, например, на Windows:

*$nmake*

*$prog*

Исходные данные берутся из файлов с префиксом in папке data, выходные данные помещаются в файлы с префиксом out в папке data.

Входные данные и результаты.

Вариант 1.

|  |  |
| --- | --- |
| **in1.txt**  0  1  -sin(3\*x)  cos(x)+sin(x)\*11/8-sin(3\*x)/8  1.0  1.0  0.0  1.0  0.1  // p(x)  // q(x)  // f(x)  // correct(x)  // y(a)  // y'(a)  // a  // b  // h | **out1.txt**  Метод 1: Метод Эйлера  Решение:  X: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1  Y: 1 1.1 1.19 1.27196 1.34766 1.41847 1.48513 1.54758 1.60492 1.65541 1.69661  Точное решение:  Y: 1 1.09534 1.18266 1.26376 1.34001 1.41211 1.47999 1.54274 1.59864 1.64526 1.67968  Погрешность относительно точного решения:  0.0775537  Погрешность методом Рунге-Ромберга:  X[0]: -0  X[1]: -0.00333333  X[2]: -0.00534368  X[3]: -0.00609061  X[4]: -0.005823  X[5]: -0.00494817  X[6]: -0.00398561  X[7]: -0.00350928  X[8]: -0.00408408  X[9]: -0.00620223  X[10]: -0.0102254  Метод 2: Метод Рунге-Кутты  Решение:X: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1  Y: 1 1.09534 1.18266 1.26376 1.34001 1.41211 1.47999 1.54274 1.59864 1.64526 1.67969  Точное решение:Y: 1 1.09534 1.18266 1.26376 1.34001 1.41211 1.47999 1.54274 1.59864 1.64526 1.67968  Погрешность относительно точного решения:  1.48697e-005  Погрешность методом Рунге-Ромберга:  X[0]: -0  X[1]: -5.39739e-007  X[2]: -1.05352e-006  X[3]: -1.49463e-006  X[4]: -1.82415e-006  X[5]: -2.0146e-006  X[6]: -2.05248e-006  X[7]: -1.9396e-006  X[8]: -1.69295e-006  X[9]: -1.34318e-006  X[10]: -9.31888e-007  Метод 3: Метод Адамса  Решение:  X: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1  Y: 1 1.09534 1.18266 1.26376 1.34009 1.41221 1.48008 1.54275 1.59851 1.64493 1.67911  Точное решение:  Y: 1 1.09534 1.18266 1.26376 1.34001 1.41211 1.47999 1.54274 1.59864 1.64526 1.67968  Погрешность относительно точного решения: 0.00133145  Погрешность методом Рунге-Ромберга:  X[0]: -0  X[1]: -5.39739e-007  X[2]: 2.15197e-006  X[3]: 6.65434e-006  X[4]: -7.96602e-005  X[5]: -0.000106403  X[6]: -9.05516e-005  X[7]: -1.58885e-005  X[8]: 0.000123767  X[9]: 0.00031791  X[10]: 0.000557793 |
| **in2.txt**  x  2  -x  0  e^x/x  1.0  2.0  0.1  0.0  1.0  1.5  1.0  0.0  7.39  0.001  // r(x)  // p(x)  // q(x)  // f(x)  // correct(x)  // a  // b  // h  // alpha  // beta  // delta  // gamma  // y0  // y1  // eps | **out2.txt**  Метод 1: Метод стрельбы  Решение:  X: 1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2  Y: 2.71863 2.73141 2.76712 2.8229 2.89694 2.98818 3.09604 3.22038 3.36135 3.51934 3.695  Точное решение:  Y: 2.71828 2.73106 2.76676 2.82254 2.89657 2.98779 3.09565 3.21997 3.36092 3.51889 3.69453  Погрешность относительно точного решения:  0.00433988  Погрешность методом Рунге-Ромберга:  X[0]: 2.08284e-006  X[1]: -1.2861e-006  X[2]: -2.84422e-006  X[3]: -3.47277e-006  X[4]: -3.61499e-006  X[5]: -3.50319e-006  X[6]: -3.2627e-006  X[7]: -2.96216e-006  X[8]: -2.63923e-006  X[9]: -2.31411e-006  X[10]: -1.99698e-006  Метод 2: Метод конечных разностей  Решение:  X: 1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2  Y: 2.72742 2.73901 2.77377 2.82879 2.90222 2.99295 3.10039 3.22437 3.36503 3.52276 3.69819  Точное решение:  Y: 2.71828 2.73106 2.76676 2.82254 2.89657 2.98779 3.09565 3.21997 3.36092 3.51889 3.69453  Погрешность относительно точного решения:  0.0619573  Погрешность методом Рунге-Ромберга:  X[0]: -0.00841984  X[1]: -0.00728848  X[2]: -0.00639161  X[3]: -0.00567042  X[4]: -0.00508305  X[5]: -0.0045989  X[6]: -0.00419503  X[7]: -0.00385386  X[8]: -0.00356164  X[9]: -0.00330738  X[10]: -0.00308209 |

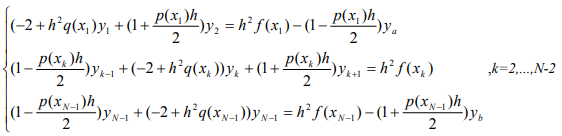
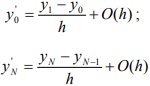
Выводы.

Выполнив лабораторную работу, я научился реализовывать методы решения задач Коши и краевых задач: метод Эйлера (*O(h)*), метод Рунге-Кутты четвёртого порядка, метод Адамса четвёртого порядка, метод стрельбы, метод конечных разностей (*O(h)* или *O(h/2)*). Существует несколько модифицированных методов Эйлера со вторым порядком точности.

Для решения жёстких систем ОДУ существуют неявные методы, например неявный метод Эйлера, использующий производную на правой границе: .

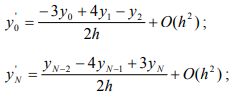


В методе Адамса можно использовать односторонние разности первого порядка, тогда мы получаем трёхдиагональную матрицу коэффициентов .



Или односторонние разности второго порядка,

, тогда второй порядок аппроксимации сохраняется везде, но матрица линейной системы не трехдиагональная. Первый подход лучше использовать, когда требуется высокая скорость вычислений, второй – когда более важна высокая точность решения.



Исходный код.

//lup.hpp

#ifndef \_\_LUP\_\_

#define \_\_LUP\_\_

#include <vector>

#include <cmath>

#include <fstream>

#include <iostream>

#include "matrix.hpp"

#include "vector.hpp"

using namespace std;

int m\_detSign = 1;

void m\_frontSub(Matrix &mat, vector <double> &vec, vector <double> &vecP, vector <double> &vecZ)

{

int n = mat.getSize();

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

double sum = 0.0;

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

sum += mat.get(i, j) \* vecZ[j];

}

vecZ[i] = vec[(int)vecP[i]] - sum;

}

}

void m\_backSub(Matrix &mat, vector <double> &vec, vector <double> &vecX)

{

int n = mat.getSize();

for (int i = n - 1; i >= 0; --i) {

double sum = 0.0;

for (int j = i + 1; j < n; ++j) {

sum += mat.get(i, j) \* vecX[j];

}

vecX[i] = (vec[i] - sum) / mat.get(i, i);

}

}

bool m\_lup(Matrix &mat, Matrix &matL, Matrix &matU, vector <double> &vecP)

{

int n = mat.getSize();

m\_detSign = 1;

matU.copy(mat);

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

vecP[i] = i;

}

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

int row = -1;

double max = 0.0;

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

double element = abs(matU.get(i, j));

if (element > max) {

max = element;

row = i;

}

}

if (row == -1) {

return false;

}

if (row != j) {

m\_detSign \*= -1;

}

matU.swapRows(j, row);

matL.swapRows(j, row);

matL.set(j, j, 1);

swap(vecP[j], vecP[row]);

for (int i = j + 1; i < n; ++i) {

double ratio = matU.get(i, j) / matU.get(j, j);

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

matU.set(i, k, matU.get(i, k) - matU.get(j, k) \* ratio);

}

matL.set(i, j, ratio);

}

}

return true;

}

void matInverse(Matrix &mat, Matrix &matInv)

{

int n = mat.getSize();

vector <double> vec1(n);

vector <double> vec2(n);

vector <double> vecP(n);

vector <double> vecX(n);

Matrix matL(n);

Matrix matU(n);

Matrix matE(n);

m\_lup(mat, matL, matU, vecP);

matE.identity();

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

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

vec1[i] = matE.get(i, j);

}

m\_frontSub(matL, vec1, vecP, vec2);

m\_backSub(matU, vec2, vecX);

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

matInv.set(i, j, vecX[i]);

}

}

}

double matDet(Matrix &mat)

{

int n = mat.getSize();

double res = 1.0;

Matrix matL(n);

Matrix matU(n);

vector <double> vecP(n);

m\_lup(mat, matL, matU, vecP);

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

res \*= matU.get(i, i);

}

return res \* m\_detSign;

}

bool lup(Matrix &mat, vector <double> &vec, vector <double> &vecX)

{

int n = mat.getSize();

Matrix matL(n);

Matrix matU(n);

vector <double> vecP(n);

vector <double> vecZ(n);

if (!m\_lup(mat, matL, matU, vecP)) {

return false;

}

m\_frontSub(matL, vec, vecP, vecZ);

m\_backSub(matU, vecZ, vecX);

return true;

}

void method1()

{

ifstream fin("../data/in1.txt");

ofstream fout("../data/out1.txt");

int n, tmp;

fin >> n;//int n;

Matrix mat(n, n);

vector <double> vec(n);

vector <double> vecX(n);

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

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

fin >> tmp;

mat.set(i, j, tmp);

}

}

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

fin >> vec[i];

}

Matrix matInv(n);

if (!lup(mat, vec, vecX)) {

fout << "Degenerate matrix\n";

} else {

matInverse(mat, matInv);

fout << "Answer (vector):\n";

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

fout << vecX[i] << " ";

}

fout << "\nInverse matrix:\n";

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

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

fout << matInv.get(i, j) << " ";

}

fout << "\n";

}

fout << "Determinant: " << matDet(mat) << "\n";

}

}

#endif

//error.hpp

#ifndef \_\_ERROR\_\_

#define \_\_ERROR\_\_

#include <vector>

#include <cmath>

#include <fstream>

#include <iostream>

#include <sstream>

#include "matrix.hpp"

#include "vector.hpp"

using namespace std;

double func1(double x)

{

return cos(x) + sin(x) \* 11 / 8 - sin(3 \* x) / 8;

}

double calcError1(vector <double> vecX, vector <double> vecY)

{

double res = 0.0;

for (int i = 0; i < vecX.size(); ++i) {

res += abs(func1(vecX[i]) - vecY[i]);

}

return res;

}

double func2(double x)

{

return exp(x) / x;

}

double calcError2(vector <double> vecX, vector <double> vecY)

{

double res = 0.0;

for (int i = 0; i < vecX.size(); ++i) {

res += abs(func2(vecX[i]) - vecY[i]);

}

return res;

}

double rungeRomberg(double h1, double h2, double y1, double y2, double p)

{

double r = h2 / h1;

return (y1 - y2) / (pow(r, p) - 1.0);

}

#endif

//matrix.hpp

#ifndef \_\_matrix\_\_

#define \_\_matrix\_\_

#include <string>

#include <vector>

#include <utility>

#include <algorithm>

#include <fstream>

using namespace std;

class Matrix

{

public:

friend ofstream &operator<<(ofstream &ofstr, Matrix &m);

friend ifstream &operator>>(ifstream &ifstr, Matrix &m);

Matrix();

Matrix(int size);

Matrix(int rows, int cols);

void resize(int rows, int cols);

double get(int row, int col);

void set(int row, int col, double value);

int getSize();

int getM();

int getN();

Matrix sub(Matrix other);

Matrix mul(double value);

Matrix mul(Matrix other);

vector <double> mul(vector <double> other);

Matrix transpose();

void identity();

void copy(Matrix other);

void swapRows(int index1, int index2);

string toString();

private:

vector < vector<double> > m\_mat;

};

#endif

//matrix.cpp

#include <string>

#include <vector>

#include <utility>

#include <algorithm>

#include <fstream>

#include "matrix.hpp"

using namespace std;

Matrix::Matrix() {}

Matrix::Matrix(int size)

{

m\_mat.resize(size);

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

m\_mat[i].resize(size);

}

}

Matrix::Matrix(int rows, int cols)

{

m\_mat.resize(rows);

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

m\_mat[i].resize(cols);

}

}

void Matrix::resize(int rows, int cols)

{

m\_mat.resize(rows);

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

m\_mat[i].resize(cols);

}

}

double Matrix::get(int row, int col)

{

return m\_mat[row][col];

}

void Matrix::set(int row, int col, double value)

{

m\_mat[row][col] = value;

}

int Matrix::getSize()

{

return m\_mat.size();

}

int Matrix::getM()

{

return getSize();

}

int Matrix::getN()

{

return m\_mat[0].size();

}

Matrix Matrix::sub(Matrix other)

{

int n = getSize();

Matrix res(n);

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

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

res.set(i, j, get(i, j) - other.get(i, j));

}

}

return res;

}

Matrix Matrix::mul(double value)

{

int n = getSize();

Matrix res(n);

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

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

res.set(i, j, get(i, j) \* value);

}

}

return res;

}

Matrix Matrix::mul(Matrix other)

{

int n = getSize();

Matrix res(n);

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

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

double sum = 0.0;

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

sum += get(i, k) \* other.get(k, j);

}

res.set(i, j, sum);

}

}

return res;

}

vector <double> Matrix::mul(vector <double> other)

{

int n = getSize();

vector <double> res(n);

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

double sum = 0.0;

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

sum += get(i, j) \* other[j];

}

res[i] = sum;

}

return res;

}

Matrix Matrix::transpose()

{

int n = getSize();

Matrix res(n);

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

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

res.set(i, j, get(j, i));

}

}

return res;

}

void Matrix::identity()

{

int n = getSize();

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

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

if (i == j) {

set(i, j, 1.0);

} else {

set(i, j, 0.0);

}

}

}

}

void Matrix::copy(Matrix other)

{

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

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

set(i, j, other.get(i, j));

}

}

}

void Matrix::swapRows(int index1, int index2)

{

swap(m\_mat[index1], m\_mat[index2]);

//double tmp = m\_mat[index1];

//m\_mat[index1] = m\_mat[index2];

//m\_mat[index2] = tmp;

}

string Matrix::toString()

{

// string res = Arrays.toString(m\_mat[0]);

// for (int i = 1; i < getM(); ++i) {

// res += '\n' + Arrays.toString(m\_mat[i]);

// }

string res = "To string doesnt work\n";

return res;

}

ifstream &operator >> (ifstream &ifstr, Matrix &m)

{

for (int i = 0; i < m.m\_mat.size(); ++i) {

for (int j = 0; j < m.m\_mat[0].size(); ++j) {

ifstr >> m.m\_mat[i][j];

}

}

return ifstr;

}

ofstream &operator << (ofstream &ofstr, Matrix &m)

{

for (int i = 0; i < m.m\_mat.size(); ++i) {

for (int j = 0; j < m.m\_mat[0].size(); ++j) {

ofstr.width(10);

ofstr << m.get(i, j) << " ";

}

ofstr << "\n";

}

return ofstr;

}

//vector.hpp

#ifndef \_\_VECTOR\_\_

#define \_\_VECTOR\_\_

#include <vector>

#include <cmath>

#include <fstream>

#define MAX\_ITERATIONS 1000

using namespace std;

vector <double> vecAdd(vector <double> a, vector <double> b);

vector <double> vecSub(vector <double> a, vector <double> b);

double vecNormC(vector <double> v);

void inputVec(vector <double> &vec, ifstream &ifs);

void printVec(vector <double> &vec, ofstream &ofs);

#endif

//vector.cpp

#include <vector>

#include <cmath>

#include <fstream>

#include "vector.hpp"

vector <double> vecAdd(vector <double> a, vector <double> b)

{

vector <double> c(a.size());

for (int i = 0; i < a.size(); ++i) {

c[i] = a[i] + b[i];

}

return c;

}

vector <double> vecSub(vector <double> a, vector <double> b)

{

vector <double> c(a.size());

for (int i = 0; i < a.size(); ++i) {

c[i] = a[i] - b[i];

}

return c;

}

double vecNormC(vector <double> v)

{

double res = 0.0;

for (int i = 0; i < v.size(); ++i) {

res = max(res, abs(v[i]));

}

return res;

}

void inputVec(vector <double> &vec, ifstream &ifs)

{

for (int i = 0; i < vec.size(); ++i) {

ifs >> vec[i];

}

}

void printVec(vector <double> &vec, ofstream &ofs)

{

for (int i = 0; i < vec.size(); ++i) {

ofs.width(10);

ofs << vec[i] << " ";

}

ofs << "\n";

}

//diffEqBoundary.hpp

#ifndef \_\_DIFF\_EQ\_BOUNDARY\_\_

#define \_\_DIFF\_EQ\_BOUNDARY\_\_

#include <vector>

#include <cmath>

#include <fstream>

#include <iostream>

#include <string>

#include "matrix.hpp"

#include "lup.hpp"

#include "vector.hpp"

#include "diffEqCauchy.hpp"

using namespace std;

class MethodDiffEqBoundary

{

public:

MethodDiffEqBoundary(string exprR, string exprP, string exprQ, string exprF, double a, double b, double h, double alpha, double beta, double delta, double gamma, double y0, double y1)

{

m\_exprR = exprR;

m\_exprP = exprP;

m\_exprQ = exprQ;

m\_exprF = exprF;

m\_a = a;

m\_b = b;

m\_h = h;

m\_alpha = alpha;

m\_beta = beta;

m\_delta = delta;

m\_gamma = gamma;

m\_y0 = y0;

m\_y1 = y1;

}

void shooting(vector <double> &vecX, vector <double> &vecY, double eps)

{

int n = getN();

double yEta1;

double yEta2;

double eta1 = 1000000.0;

double eta2 = -1000000.0;

vector <double> vecZ(n);

string strExprR = "x";

string exprP = "2 / x";

string exprQ = " - x / x";

string exprF = "0 / x";

yEta1 = fi(eta1, vecX, vecY, vecZ, exprP, exprQ, exprF);

yEta2 = fi(eta2, vecX, vecY, vecZ, exprP, exprQ, exprF);

double error = abs(eta2 - eta1);

int iter = 0;

while (error > eps) {

iter++;

double etaN = eta2 - yEta2 \* (eta2 - eta1) / (yEta2 - yEta1);

yEta1 = yEta2;

yEta2 = fi(etaN, vecX, vecY, vecZ, exprP, exprQ, exprF);

eta1 = eta2;

eta2 = etaN;

error = abs(eta2 - eta1);

}

cout << iter << "\n";

}

void finiteDifference(vector <double> &vecX, vector <double> &vecY)

{

int n = getN();

Matrix mat(n);

vector <double> vec(n);

vecX[0] = m\_a;

for (int i = 1; i < n; ++i) {

vecX[i] = vecX[i - 1] + m\_h;

}

mat.set(0, 0, m\_alpha - 3.0 \* m\_beta / (2.0 \* m\_h));

mat.set(0, 1, 4.0 \* m\_beta / (2.0 \* m\_h));

mat.set(0, 2, -m\_beta / (2.0 \* m\_h));

vec[0] = m\_y0;

for (int i = 1; i < n - 1; ++i) {

double x = vecX[i];

double rx = x;

double px = 2 / rx;

double qx = -x / rx;

double fx = 0 / rx;

double a = 1.0 / (m\_h \* m\_h) - px / (2.0 \* m\_h);

double b = -2.0 / (m\_h \* m\_h) + qx;

double c = 1.0 / (m\_h \* m\_h) + px / (2.0 \* m\_h);

mat.set(i, i - 1, a);

mat.set(i, i, b);

mat.set(i, i + 1, c);

vec[i] = -fx;

}

mat.set(n - 1, n - 3, m\_gamma / (2.0 \* m\_h));

mat.set(n - 1, n - 2, -4.0 \* m\_gamma / (2.0 \* m\_h));

mat.set(n - 1, n - 1, m\_delta + 3.0 \* m\_gamma / (2.0 \* m\_h));

vec[n - 1] = m\_y1;

lup(mat, vec, vecY);

}

int getN()

{

return (int)((m\_b - m\_a) / m\_h) + 1;

}

void setH(double h)

{

m\_h = h;

}

private:

double fi(double eta, vector <double> &vecX, vector <double> &vecY, vector <double> &vecZ, string exprP, string exprQ, string exprF)

{

int n = getN();

double y0;

double z0;

if (m\_beta != 0.0) {

y0 = eta;

z0 = (m\_y0 - m\_alpha \* eta) / m\_beta;

} else {

y0 = m\_y0 / m\_alpha;

z0 = eta;

}

MethodDiffEqCauchy method(exprP, exprQ, exprF, y0, z0, m\_a, m\_b, m\_h);

method.rungeKutta2(vecX, vecY, vecZ);

return m\_delta \* vecY[n - 1] + m\_gamma \* vecZ[n - 1] - m\_y1;

}

string m\_exprR;

string m\_exprP;

string m\_exprQ;

string m\_exprF;

double m\_a;

double m\_b;

double m\_alpha;

double m\_beta;

double m\_delta;

double m\_gamma;

double m\_y0;

double m\_y1;

double m\_h;

};

#endif

//diffEqCauchy.hpp

#ifndef \_\_DIFF\_EQ\_CAUCHY\_\_

#define \_\_DIFF\_EQ\_CAUCHY\_\_

#include <vector>

#include <cmath>

#include <fstream>

#include <iostream>

#include <string>

#include "matrix.hpp"

#include "vector.hpp"

using namespace std;

class MethodDiffEqCauchy

{

public:

MethodDiffEqCauchy(string exprP, string exprQ, string exprF, double y0, double z0, double a, double b, double h)

{

m\_exprP = exprP;

m\_exprQ = exprQ;

m\_exprF = exprF;

m\_y0 = y0;

m\_z0 = z0;

m\_a = a;

m\_b = b;

m\_h = h;

}

void euler(vector <double> &vecX, vector <double> &vecY)

{

int n = getN();

vector <double> vecZ(n);

vecX[0] = m\_a;

vecY[0] = m\_y0;

vecZ[0] = m\_z0;

for (int i = 1; i < n; ++i) {

vecX[i] = vecX[i - 1] + m\_h;

vecY[i] = vecY[i - 1] + m\_h \* vecZ[i - 1];

vecZ[i] = vecZ[i - 1] + m\_h \* g(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

}

}

void rungeKutta(vector <double> &vecX, vector <double> &vecY, vector <double> &vecZ)

{

int n = getN();

vecX[0] = m\_a;

vecY[0] = m\_y0;

vecZ[0] = m\_z0;

for (int i = 1; i < n; ++i) {

vecX[i] = vecX[i - 1] + m\_h;

vecY[i] = vecY[i - 1] + dy(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

vecZ[i] = vecZ[i - 1] + dz(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

}

}

void rungeKutta2(vector <double> &vecX, vector <double> &vecY, vector <double> &vecZ)

{

int n = getN();

vecX[0] = m\_a;

vecY[0] = m\_y0;

vecZ[0] = m\_z0;

for (int i = 1; i < n; ++i) {

vecX[i] = vecX[i - 1] + m\_h;

vecY[i] = vecY[i - 1] + dy2(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

vecZ[i] = vecZ[i - 1] + dz2(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

}

}

void adams(vector <double> &vecX, vector <double> &vecY)

{

int n = getN();

double b = m\_b;

vector <double> vecZ(n);

setB(m\_a + 3.0 \* m\_h);

rungeKutta(vecX, vecY, vecZ);

setB(b);

for (int i = 4; i < n; ++i) {

double deltaY = 0.0;

double deltaZ = 0.0;

deltaY += 55.0 \* vecZ[i - 1];

deltaY -= 59.0 \* vecZ[i - 2];

deltaY += 37.0 \* vecZ[i - 3];

deltaY -= 9.0 \* vecZ[i - 4];

deltaY /= 24.0;

deltaZ += 55.0 \* g(vecX[i - 1], vecY[i - 1], vecZ[i - 1]);

deltaZ -= 59.0 \* g(vecX[i - 2], vecY[i - 2], vecZ[i - 2]);

deltaZ += 37.0 \* g(vecX[i - 3], vecY[i - 3], vecZ[i - 3]);

deltaZ -= 9.0 \* g(vecX[i - 4], vecY[i - 4], vecZ[i - 4]);

deltaZ /= 24.0;

vecX[i] = vecX[i - 1] + m\_h;

vecY[i] = vecY[i - 1] + m\_h \* deltaY;

vecZ[i] = vecZ[i - 1] + m\_h \* deltaZ;

}

}

int getN()

{

return (int)((m\_b - m\_a) / m\_h) + 1;

}

void setH(double h)

{

m\_h = h;

}

private:

void setB(double b)

{

m\_b = b;

}

double g(double x, double y, double z)

{

return -(0 \* z + 1 \* y + (-sin(3 \* x)));

}

double g2(double x, double y, double z)

{

return -(2 / x \* z + (-x / x) \* y + 0);

}

double k1(double z)

{

return m\_h \* z;

}

double l1(double x, double y, double z)

{

return m\_h \* g(x, y, z);

}

double k2(double x, double y, double z)

{

return m\_h \* (z + 0.5 \* l1(x, y, z));

}

double l2(double x, double y, double z)

{

return m\_h \* g(x + 0.5 \* m\_h, y + 0.5 \* k1(z), z + 0.5 \* l1(x, y, z));

}

double k3(double x, double y, double z)

{

return m\_h \* (z + 0.5 \* l2(x, y, z));

}

double l3(double x, double y, double z)

{

return m\_h \* g(x + 0.5 \* m\_h, y + 0.5 \* k2(x, y, z), z + 0.5 \* l2(x, y, z));

}

double k4(double x, double y, double z)

{

return m\_h \* (z + l3(x, y, z));

}

double l4(double x, double y, double z)

{

return m\_h \* g(x + m\_h, y + k3(x, y, z), z + l3(x, y, z));

}

double k1\_2(double z)

{

return m\_h \* z;

}

double l1\_2(double x, double y, double z)

{

return m\_h \* g2(x, y, z);

}

double k2\_2(double x, double y, double z)

{

return m\_h \* (z + 0.5 \* l1\_2(x, y, z));

}

double l2\_2(double x, double y, double z)

{

return m\_h \* g2(x + 0.5 \* m\_h, y + 0.5 \* k1\_2(z), z + 0.5 \* l1\_2(x, y, z));

}

double k3\_2(double x, double y, double z)

{

return m\_h \* (z + 0.5 \* l2\_2(x, y, z));

}

double l3\_2(double x, double y, double z)

{

return m\_h \* g2(x + 0.5 \* m\_h, y + 0.5 \* k2\_2(x, y, z), z + 0.5 \* l2\_2(x, y, z));

}

double k4\_2(double x, double y, double z)

{

return m\_h \* (z + l3\_2(x, y, z));

}

double l4\_2(double x, double y, double z)

{

return m\_h \* g2(x + m\_h, y + k3\_2(x, y, z), z + l3\_2(x, y, z));

}

double dy(double x, double y, double z)

{

return (k1(z) + 2.0 \* k2(x, y, z) + 2.0 \* k3(x, y, z) + k4(x, y, z)) / 6.0;

}

double dz(double x, double y, double z)

{

return (l1(x, y, z) + 2.0 \* l2(x, y, z) + 2.0 \* l3(x, y, z) + l4(x, y, z)) / 6.0;

}

double dy2(double x, double y, double z)

{

return (k1\_2(z) + 2.0 \* k2\_2(x, y, z) + 2.0 \* k3\_2(x, y, z) + k4\_2(x, y, z)) / 6.0;

}

double dz2(double x, double y, double z)

{

return (l1\_2(x, y, z) + 2.0 \* l2\_2(x, y, z) + 2.0 \* l3\_2(x, y, z) + l4\_2(x, y, z)) / 6.0;

}

string m\_exprP;

string m\_exprQ;

string m\_exprF;

double m\_y0;

double m\_z0;

double m\_a;

double m\_b;

double m\_h;

};

#endif

//main.cpp

#include <vector>

#include <cmath>

#include <fstream>

#include <iostream>

#include <string>

#include "matrix.hpp"

#include "vector.hpp"

#include "error.hpp"

#include "diffEqBoundary.hpp"

using namespace std;

inline double correctX1(double x)

{

return cos(x) + sin(x) \* 11 / 8 - sin(3 \* x) / 8;

}

inline double correctX2(double x)

{

return x != 0 ? exp(x) / x : 0;

}

void part1()

{

ifstream fin("../data/in1.txt");

ofstream fout("../data/out1.txt");

string exprP;

fin >> exprP;

string exprQ;

fin >> exprQ;

string exprF;

fin >> exprF;

string exprA;

fin >> exprA;

double y0;

fin >> y0;

double z0;

fin >> z0;

double a;

fin >> a;

double b;

fin >> b;

double h;

fin >> h;

double h2 = h / 2.0;

MethodDiffEqCauchy method(exprP, exprQ, exprF, y0, z0, a, b, h);

int n = method.getN();

vector <double> vecX1(n);

vector <double> vecY1(n);

vector <double> vecZ1(n);

vector <double> vecX2(n \* 2);

vector <double> vecY2(n \* 2);

vector <double> vecZ2(n \* 2);

vector <double> vecY(n);

method.euler(vecX1, vecY1);

method.setH(h2);

method.euler(vecX2, vecY2);

method.setH(h);

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

vecY[i] = correctX1(vecX1[i]);

}

fout << ("Метод 1: Метод Эйлера\n");

fout << ("Решение:\n");

fout << "X: ";

printVec(vecX1, fout);

fout << "Y: ";

printVec(vecY1, fout);

fout << ("Точное решение:\n");

fout << "Y: ";

printVec(vecY, fout);

fout << "Погрешность относительно точного решения:\n" << calcError1(vecX1, vecY1) << "\n";

fout << ("Погрешность методом Рунге-Ромберга:\n");

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

fout << "X[" << i << "]: " << rungeRomberg(h, h2, vecY1[i], vecY2[i \* 2], 2.0) << "\n";

}

method.rungeKutta(vecX1, vecY1, vecZ1);

method.setH(h2);

method.rungeKutta(vecX2, vecY2, vecZ2);

method.setH(h);

fout << ("\nМетод 2: Метод Рунге-Кутты\n");

fout << ("Решение:");

fout << "X: ";

printVec(vecX1, fout);

fout << "Y: ";

printVec(vecY1, fout);

fout << ("Точное решение:");

fout << "Y: ";

printVec(vecY, fout);

fout << "Погрешность относительно точного решения:\n";

fout << calcError1(vecX1, vecY1) << "\n";

fout << ("Погрешность методом Рунге-Ромберга:\n");

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

fout << "X[" << i << "]: ";

fout << rungeRomberg(h, h2, vecY1[i], vecY2[i \* 2], 4.0) << "\n";

}

method.adams(vecX1, vecY1);

method.setH(h2);

method.adams(vecX2, vecY2);

method.setH(h);

fout << ("\nМетод 3: Метод Адамса\n");

fout << ("Решение:\n");

fout << "X: ";

printVec(vecX1, fout);

fout << "Y: ";

printVec(vecY1, fout);

fout << ("Точное решение:\n");

fout << "Y: ";

printVec(vecY, fout);

fout << "Погрешность относительно точного решения: ";

fout << calcError1(vecX1, vecY1) << "\n";

fout << "Погрешность методом Рунге-Ромберга:\n";

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

fout << "X[" << i << "]: ";

fout << rungeRomberg(h, h2, vecY1[i], vecY2[i \* 2], 4.0) << "\n";

}

fin.close();

fout.close();

}

void part2()

{

ifstream fin("../data/in2.txt");

ofstream fout("../data/out2.txt");

string exprR;

fin >> exprR;

string exprP;

fin >> exprP;

string exprQ;

fin >> exprQ;

string exprF;

fin >> exprF;

string exprA;

fin >> exprA;

double a;

fin >> a;

double b;

fin >> b;

double h;

fin >> h;

double alpha;

fin >> alpha;

double beta;

fin >> beta;

double delta;

fin >> delta;

double gamma;

fin >> gamma;

double y0;

fin >> y0;

double y1;

fin >> y1;

double eps;

fin >> eps;

double h2 = h / 2.0;

MethodDiffEqBoundary method(exprR, exprP, exprQ, exprF, a, b, h, alpha, beta, delta, gamma, y0, y1);

int n = method.getN();

vector <double> vecX1(n);

vector <double> vecY1(n);

vector <double> vecX2(n \* 2);

vector <double> vecY2(n \* 2);

vector <double> vecY(n);

vector <double> vecError(n);

method.shooting(vecX1, vecY1, eps);

method.setH(h2);

method.shooting(vecX2, vecY2, eps);

method.setH(h);

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

vecY[i] = correctX2(vecX1[i]);

}//

fout << ("Метод 1: Метод стрельбы\n");

fout << ("Решение:\n");

fout << "X: ";

printVec(vecX1, fout);

fout << "Y: ";//

printVec(vecY1, fout);

fout << "Точное решение:\n";

fout << "Y: ";

printVec(vecY, fout);

fout << "Погрешность относительно точного решения:\n";

fout << calcError2(vecX1, vecY1) << "\n";

fout << "Погрешность методом Рунге-Ромберга:" << "\n";

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

vecError[i] = rungeRomberg(h, h2, vecY1[i], vecY2[i \* 2], 2.0);

fout << "X[" << i << "]: " << vecError[i] << "\n";

}

method.finiteDifference(vecX1, vecY1);

method.setH(h2);

method.finiteDifference(vecX2, vecY2);

method.setH(h);

fout << ("\nМетод 2: Метод конечных разностей\n");

fout << ("Решение:\n");

fout << "X: ";

printVec(vecX1, fout);

fout << "Y: ";

printVec(vecY1, fout);

fout << ("Точное решение:\n");

fout << "Y: ";

printVec(vecY, fout);

fout << "Погрешность относительно точного решения:\n" << calcError2(vecX1, vecY1) << "\n";

fout << "Погрешность методом Рунге-Ромберга:\n";

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

vecError[i] = rungeRomberg(h, h2, vecY1[i], vecY2[i \* 2], 2.0);

fout << "X[" << i << "]: " << vecError[i] << "\n";

}

fin.close();

fout.close();

}

int main()

{

part1();

part2();

return 0;

}