**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа № 4   
по курсу «Численные методы»

Группа: М8О-308Б-21

Студент: Белоносов К. А.

Преподаватель: Ревизников Д. Л.

Дата: 22.05.2024

Москва, 2024

**ОГЛАВЛЕНИЕ**

[1 Тема 3](#_Toc158983147)

[2 Задание 3](#_Toc158983148)

[3 Теория 4](#_Toc158983149)

[4 Ход лабораторной работы 5](#_Toc158983150)

[5 Выводы 6](#_Toc158983151)

# **Тема**

Методы решения начальных и краевых задач для обыкновенных дифференциальных уравнений (ОДУ) и систем ОДУ

# **Задание**

4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

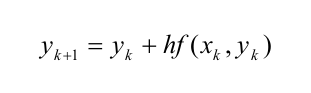
|  |  |  |
| --- | --- | --- |
| 3 | , |  |

4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

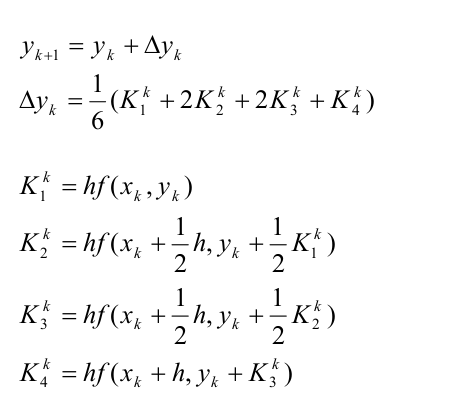
|  |  |  |
| --- | --- | --- |
| 3 | x2(x+1) y″-2y=0,  y′ (1)=-1,  2y(2) – 4 y′(2) =4 |  |

# **Теория**

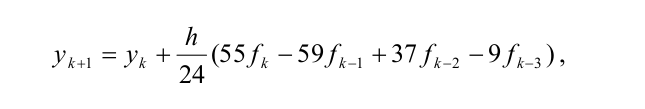
**Метод Эйлера** играет важную роль в теории численных методов решения ОДУ, хотя и не часто используется в практических расчетах из-за невысокой точности. Вывод расчетных соотношений для этого метода может быть произведен несколькими способами: с помощью геометрической интерпретации, с использованием разложения в ряд Тейлора, конечно разностным методом (с помощью разностной аппроксимации производной), квадратурным способом (использованием эквивалентного интегрального уравнения).



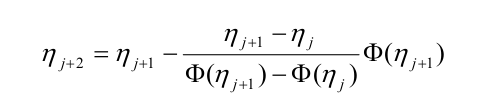
Все рассмотренные выше явные методы являются вариантами методов **Рунге-Кутты.** Семейство явных методов Рунге-Кутты 4-го порядка записывается в виде совокупности формул:



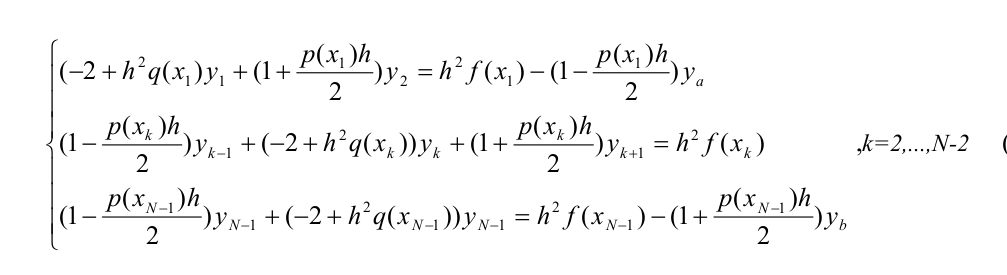
При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим **метод Адамса** четвертого порядка точности:



Суть **метода стрельбы** заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи.



**Конечно-разностный метод** решения краевой задачи получим систему линейных алгебраических уравнений с трехдиагональной матрицей коэффициентов, которую можно решить с помощью метода прогонки:



# **Х****од лабораторной работы**

Код был реализован на языке C++

#ifndef LAB4\_DERIVATIVEMETHODS\_H

#define LAB4\_DERIVATIVEMETHODS\_H

#include <iostream>

#include <vector>

#include <functional>

#include <cmath>

#include <iomanip>

#include "../lab1/Matrix.h"

using namespace std;

using namespace numeric;

vector<vector<double>>

eulerMethod(const std::function<double(double, double, double)> &f, double x0, double y1\_0, double y2\_0, double h,

int n) {

vector<double> x(n), y1(n), y2(n);

x[0] = x0;

y1[0] = y1\_0;

y2[0] = y2\_0;

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

x[i] = x[i - 1] + h;

y1[i] = y1[i - 1] + h \* y2[i - 1];

y2[i] = y2[i - 1] + h \* f(x[i - 1], y1[i - 1], y2[i - 1]);

}

return vector<vector<double>>{x, y1, y2};

}

vector<vector<double>>

rungeKutta4(const std::function<double(double, double, double)> &f, double x0, double y1\_0, double y2\_0, double h,

int n) {

vector<double> x(n), y1(n), y2(n);

x[0] = x0;

y1[0] = y1\_0;

y2[0] = y2\_0;

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

x[i] = x[i - 1] + h;

double k1\_y1 = h \* y2[i - 1];

double k1\_y2 = h \* f(x[i - 1], y1[i - 1], y2[i - 1]);

double k2\_y1 = h \* (y2[i - 1] + 0.5 \* k1\_y2);

double k2\_y2 = h \* f(x[i - 1] + 0.5 \* h, y1[i - 1] + 0.5 \* k1\_y1, y2[i - 1] + 0.5 \* k1\_y2);

double k3\_y1 = h \* (y2[i - 1] + 0.5 \* k2\_y2);

double k3\_y2 = h \* f(x[i - 1] + 0.5 \* h, y1[i - 1] + 0.5 \* k2\_y1, y2[i - 1] + 0.5 \* k2\_y2);

double k4\_y1 = h \* (y2[i - 1] + k3\_y2);

double k4\_y2 = h \* f(x[i - 1] + h, y1[i - 1] + k3\_y1, y2[i - 1] + k3\_y2);

y1[i] = y1[i - 1] + (k1\_y1 + 2 \* k2\_y1 + 2 \* k3\_y1 + k4\_y1) / 6;

y2[i] = y2[i - 1] + (k1\_y2 + 2 \* k2\_y2 + 2 \* k3\_y2 + k4\_y2) / 6;

}

return vector<vector<double>>{x, y1, y2};

}

vector<vector<double>>

adams4(const std::function<double(double, double, double)> &f, double x0, double y1\_0, double y2\_0, double h, int n) {

vector<double> x(n), y1(n), y2(n);

x[0] = x0;

y1[0] = y1\_0;

y2[0] = y2\_0;

for (int i = 1; i < 4; ++i) {

x[i] = x[i - 1] + h;

double k1\_y1 = h \* y2[i - 1];

double k1\_y2 = h \* f(x[i - 1], y1[i - 1], y2[i - 1]);

double k2\_y1 = h \* (y2[i - 1] + 0.5 \* k1\_y2);

double k2\_y2 = h \* f(x[i - 1] + 0.5 \* h, y1[i - 1] + 0.5 \* k1\_y1, y2[i - 1] + 0.5 \* k1\_y2);

double k3\_y1 = h \* (y2[i - 1] + 0.5 \* k2\_y2);

double k3\_y2 = h \* f(x[i - 1] + 0.5 \* h, y1[i - 1] + 0.5 \* k2\_y1, y2[i - 1] + 0.5 \* k2\_y2);

double k4\_y1 = h \* (y2[i - 1] + k3\_y2);

double k4\_y2 = h \* f(x[i - 1] + h, y1[i - 1] + k3\_y1, y2[i - 1] + k3\_y2);

y1[i] = y1[i - 1] + (k1\_y1 + 2 \* k2\_y1 + 2 \* k3\_y1 + k4\_y1) / 6;

y2[i] = y2[i - 1] + (k1\_y2 + 2 \* k2\_y2 + 2 \* k3\_y2 + k4\_y2) / 6;

}

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

x[i] = x[i - 1] + h;

y1[i] = y1[i - 1] + h / 24 \* (55 \* y2[i - 1] - 59 \* y2[i - 2] + 37 \* y2[i - 3] - 9 \* y2[i - 4]);

y2[i] = y2[i - 1] + h / 24 \* (55 \* f(x[i - 1], y1[i - 1], y2[i - 1]) - 59 \* f(x[i - 2], y1[i - 2], y2[i - 2]) +

37 \* f(x[i - 3], y1[i - 3], y2[i - 3]) - 9 \* f(x[i - 4], y1[i - 4], y2[i - 4]));

}

return vector<vector<double>>{x, y1, y2};

}

void rungeRomberg(

const std::function<vector<vector<double>>(const std::function<double(double, double, double)> &, double,

double, double, double, int)> &method,

const std::function<double(double, double, double)> &f, double x0, double y1\_0, double y2\_0, double h, int n) {

vector<double> y1\_h, y2\_h, y1\_h2, y2\_h2;

auto res1 = method(f, x0, y1\_0, y2\_0, h, n);

auto res2 = method(f, x0, y1\_0, y2\_0, h / 2, 2 \* n);

y1\_h = res1[1];

y2\_h = res1[2];

y1\_h2 = res2[1];

y2\_h2 = res2[2];

const int width = 15;

const int precision = 6;

cout << "+---------------+---------------+---------------+" << endl;

cout << "| x | Error y | Error y' |" << endl;

cout << "+---------------+---------------+---------------+" << endl;

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

double x\_i = x0 + i \* h;

double error\_y1 = (y1\_h2[2 \* i] - y1\_h[i]) / (pow(2, 4) - 1);

double error\_y2 = (y2\_h2[2 \* i] - y2\_h[i]) / (pow(2, 4) - 1);

cout << "| " << setw(width - 2) << setprecision(precision) << x\_i << " | "

<< setw(width - 2) << setprecision(precision) << error\_y1 << " | "

<< setw(width - 2) << setprecision(precision) << error\_y2 << " |" << endl;

}

cout << "+---------------+---------------+---------------+" << endl;

}

void shootingMethod(const std::function<double(double, double, double)>& f, double a, double b, double alpha, double beta, double eps, double h, int n) {

vector<vector<double>> result;

vector<double> y\_trial(n);

vector<double> s\_values;

double s0 = 0;

double s1 = 1.0;

double y\_b0, y\_b1, s\_new;

auto result0 = rungeKutta4(f, a, alpha, s0, h, n);

y\_b0 = result0[1].back();

auto result1 = rungeKutta4(f, a, alpha, s1, h, n);

y\_b1 = result1[1].back();

cout << "+---------------+---------------+---------------+" << endl;

cout << "| s | f(b, y, s) | |Ф(s)| |" << endl;

cout << "+---------------+---------------+---------------+" << endl;

while (abs(y\_b1 - beta) > eps) {

s\_new = s1 - (y\_b1 - beta) \* (s1 - s0) / (y\_b1 - y\_b0);

s\_values.push\_back(s\_new);

cout << "| " << setw(13) << setprecision(6) << s1 << " | "

<< setw(13) << setprecision(6) << y\_b1 << " | "

<< setw(13) << setprecision(6) << abs(y\_b1 - beta) << " |" << endl;

s0 = s1;

y\_b0 = y\_b1;

s1 = s\_new;

result = rungeKutta4(f, a, alpha, s1, h, n);

y\_b1 = result[1].back();

}

cout << "| " << setw(13) << setprecision(6) << s1 << " | "

<< setw(13) << setprecision(6) << y\_b1 << " | "

<< setw(13) << setprecision(6) << abs(y\_b1 - beta) << " |" << endl;

cout << "+---------------+---------------+---------------+" << endl;

cout << "+---------------+---------------+" << endl;

cout << "| x | y |" << endl;

cout << "+---------------+---------------+" << endl;

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

cout << "| " << setw(13) << setprecision(6) << result[0][i] << " | "

<< setw(13) << setprecision(6) << result[1][i] << " |" << endl;

}

cout << "+---------------+---------------+" << endl;

}

void finiteDifferenceMethod(double a, double b, double y0, double y1, double h,

const std::function<double(double)>& p,

const std::function<double(double)>& q,

const std::function<double(double)>& f,

vector<double>& x, vector<double>& y) {

int n = static\_cast<int>((b - a) / h) + 1;

x.resize(n);

y.resize(n);

vector<double> rhs(n);

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

x[i] = a + i \* h;

}

rhs[0] = h \* h \* f(x[0]) - (1 - p(x[0]) \* h / 2) \* y0;

rhs[n - 1] = h \* h \* f(x[n - 1]) - (1 + p(x[n - 1]) \* h / 2) \* y1;

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

rhs[i] = h \* h \* f(x[i]);

}

Matrix<double> A(n, n);

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

A[i][i] = -2 + h \* h \* q(x[i]);

if (i > 0) A[i][i - 1] = 1 - p(x[i]) \* h / 2;

if (i < n - 1) A[i][i + 1] = (1 + p(x[i]) \* h / 2);

}

y = tridiagonalSolve(A, rhs);

cout << "+---------------+---------------+" << endl;

cout << "| x | y |" << endl;

cout << "+---------------+---------------+" << endl;

cout << "| " << setw(13) << setprecision(6) << a << " | "

<< setw(13) << setprecision(6) << y0 << " |" << endl;

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

cout << "| " << setw(13) << setprecision(6) << x[i] << " | "

<< setw(13) << setprecision(6) << y[i] << " |" << endl;

}

cout << "+---------------+---------------+" << endl;

}

void printTable(double a, double b, double h, std::function<double(double)> yFunc) {

vector<double> x, y;

for (double xi = a; xi <= b; xi += h) {

x.push\_back(xi);

y.push\_back(yFunc(xi));

}

cout << "+---------------+---------------+" << endl;

cout << "| x | y |" << endl;

cout << "+---------------+---------------+" << endl;

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

cout << "| " << setw(13) << setprecision(6) << x[i] << " | "

<< setw(13) << setprecision(6) << y[i] << " |" << endl;

}

cout << "+---------------+---------------+" << endl;

}

void printResult(const vector<vector<double>> &res) {

const int width = 15;

const int precision = 6;

cout << "+---------------+---------------+---------------+" << endl;

cout << "| x | y | y' |" << endl;

cout << "+---------------+---------------+---------------+" << endl;

for (int i = 0; i < res[0].size(); ++i) {

cout << "| " << setw(width - 2) << setprecision(precision) << res[0][i] << " | "

<< setw(width - 2) << setprecision(precision) << res[1][i] << " | "

<< setw(width - 2) << setprecision(precision) << res[2][i] << " |" << endl;

}

cout << "+---------------+---------------+---------------+" << endl;

}

#endif //LAB4\_DERIVATIVEMETHODS\_H

4.1) Методы Эйлера, Рунге-Кутты и Адамса 4-го порядка:

#include <iostream>

#include "derivativeMethods.h"

using namespace std;

double f(double x, double y1, double y2) {

return 2 \* cos(x) - y1;

}

double fExact(double x) {

return x \* sin(x) + cos(x);

}

void compareWithExact(const vector<vector<double>>& result, double (\*exactFunc)(double)) {

const int width = 15;

const int precision = 6;

cout << "+---------------+---------------+---------------+---------------+" << endl;

cout << "| x | y | Exact y | Error |" << endl;

cout << "+---------------+---------------+---------------+---------------+" << endl;

for (size\_t i = 0; i < result[0].size(); ++i) {

double exact\_y = exactFunc(result[0][i]);

double error = abs(result[1][i] - exact\_y);

cout << "| " << setw(width - 2) << setprecision(precision) << result[0][i] << " | "

<< setw(width - 2) << setprecision(precision) << result[1][i] << " | "

<< setw(width - 2) << setprecision(precision) << exact\_y << " | "

<< setw(width - 2) << setprecision(precision) << error << " |" << endl;

}

cout << "+---------------+---------------+---------------+---------------+" << endl;

}

int main() {

double x0 = 0, y1\_0 = 1, y2\_0 = 0, h = 0.1;

int n = 11;

cout << "Euler method:" << "\n";

auto eulerRes = eulerMethod(f, x0, y1\_0, y2\_0, h, n);

compareWithExact(eulerRes, fExact);

cout << "\nRunge-Kutta method:" << "\n";

auto rungeKuttaRes = rungeKutta4(f, x0, y1\_0, y2\_0, h, n);

compareWithExact(rungeKuttaRes, fExact);

cout << "\nAdams method:" << "\n";

auto adamsRes = adams4(f, x0, y1\_0, y2\_0, h, n);

compareWithExact(adamsRes, fExact);

cout << "\nEuler method error by Runge Romberg:" << "\n";

rungeRomberg(eulerMethod, f, x0, y1\_0, y2\_0, h, n);

cout << "\nRunge-Kutta method error by Runge Romberg:" << "\n";

rungeRomberg(rungeKutta4, f, x0, y1\_0, y2\_0, h, n);

cout << "\nAdams method error by Runge Romberg:" << "\n";

rungeRomberg(adams4, f, x0, y1\_0, y2\_0, h, n);

return 0;

}

4.2) Метод стрельбы, конечно-разностный метод:

#include <iostream>

#include "derivativeMethods.h"

using namespace std;

double f\_m(double x, double y1, double y2) {

return (2 \* y1) / (x \* x \* (x + 1));

}

double exactSolution(double x) {

return 1.0 / x + 1.0;

}

double rungeRombergError(const vector<double>& y\_h, const vector<double>& y\_h2, double r) {

double error = 0.0;

for (size\_t i = 0; i < y\_h.size(); ++i) {

error = max(error, abs(y\_h2[2 \* i] - y\_h[i]) / (pow(2, r) - 1));

}

return error;

}

void rungeRombergErrorShooting(const vector<vector<double>>& res\_h, const vector<vector<double>>& res\_h2) {

const int width = 15;

const int precision = 6;

cout << "+---------------+---------------+---------------+" << endl;

cout << "| x | Error y | Error y' |" << endl;

cout << "+---------------+---------------+---------------+" << endl;

for (int i = 0; i < res\_h[0].size(); ++i) {

double error\_y1 = (res\_h2[1][2 \* i] - res\_h[1][i]) / (pow(2, 4) - 1);

double error\_y2 = (res\_h2[2][2 \* i] - res\_h[2][i]) / (pow(2, 4) - 1);

cout << "| " << setw(width - 2) << setprecision(precision) << res\_h[0][i] << " | "

<< setw(width - 2) << setprecision(precision) << error\_y1 << " | "

<< setw(width - 2) << setprecision(precision) << error\_y2 << " |" << endl;

}

cout << "+---------------+---------------+---------------+" << endl;

}

int main() {

cout << "Real value:" << "\n";

printTable(1, 2, 0.2, exactSolution);

double a = 1, b = 2;

double alpha = 2, beta = 1.5;

double h = 0.2;

double eps = 0.0001;

cout << "\nShooting method:" << "\n";

shootingMethod(f\_m, a, b, alpha, beta, eps, h, 6);

cout << "\nRunge-Romberg error estimate for Shooting Method:" << "\n";

rungeRomberg(rungeKutta4, f\_m, a, alpha, 0, h, 6);

std::function<double(double)> p = [](double x) {

return 0;

};

std::function<double(double)> q = [](double x) {

return -2 / (x \* x \* (x + 1));

};

std::function<double(double)> f = [](double x) {

return 0.0;

};

double y0 = 2.0;

double y1 = 1.5;

vector<double> x\_h, y\_h, x\_h2, y\_h2;

cout << "\nFinite difference Method:" << "\n";

finiteDifferenceMethod(a, b, y0, y1, h, p, q, f, x\_h, y\_h);

cout << "\nFinite difference Method with h/2:" << "\n";

finiteDifferenceMethod(a, b, y0, y1, h / 2, p, q, f, x\_h2, y\_h2);

double error = rungeRombergError(y\_h, y\_h2, 2);

cout << "\nError Finite Difference Method:" << endl;

cout << "+---------------+---------------+" << endl;

cout << "| x | Error y |" << endl;

cout << "+---------------+---------------+" << endl;

double maxError = 0.0;

for (size\_t i = 0; i < x\_h.size(); ++i) {

double exact = exactSolution(x\_h[i]);

double error = abs(y\_h[i] - exact);

maxError = max(maxError, error);

cout << "| " << setw(13) << setprecision(6) << x\_h[i] << " | "

<< setw(13) << setprecision(6) << error << " |" << endl;

}

cout << "+---------------+---------------+" << endl;

return 0;

}

# **Выводы**

В ходе лабораторной работы были реализованы методы Эйлера (явный и модифицированный), метод Адамса 4-го порядка, Рунге-Кутта 4-го порядка, метод стрельбы и конечно-разностный метод. Результаты работы алгоритмов оценены с помощью сравнения с точным решением, а также методом Рунге-Ромберга.