Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент:	Г. С. Будайчиев
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

4.1 Методы Эйлера, Рунге-Кутты и Адамса

1 Постановка задачи

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки . 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.

Вариант: 3

3	$y'' - 2y - 4x^2 e^{x^2} = 0$, $y(0) = 3$, $y'(0) = 0$, $x \in [0,1], h = 0.1$	$y = e^{x^2} + e^{x\sqrt{2}} + e^{-x\sqrt{2}}$

Рис. 1: Входные данные

2 Результаты работы

Euler method:

x	y	Exact y	Error
0	1	3	2
0.1	1	3.03008	2.03008
0.2	1.02	3.12135	2.10135
0.3	1.0604	3.27689	2.21649
0.4	1.12287	3.50214	2.37926
0.5	1.21049	3.80521	2.59472
0.6	1.32807	4.19758	2.86951
0.7	1.48271	4.69501	3.2123
0.8	1.68454	5.31897	3.63443
0.9	1.94803	6.09877	4.15074
1	2.29375	7.07465	4.7809

Runge-Kutta method:

x	y	Exact y	Error
0	1	3	2
0.1	1.01005	3.03008	2.02003
0.2	1.04081	3.12135	2.08054
0.3	1.09417	3.27689	2.18272
0.4	1.17351	3.50214	2.32863
0.5	1.28402	3.80521	2.52119
0.6	1.43332	4.19758	2.76426
0.7	1.6323	4.69501	3.06271
0.8	1.89646	5.31897	3.42251
0.9	2.24788	6.09877	3.85089
1	2.71824	7.07465	4.35641

Adams method:

x	y	Exact y	Error
0	1	3	2
0.1	1.01005	3.03008	2.02003
0.2	1.04081	3.12135	2.08054
0.3	1.09417	3.27689	2.18272
0.4	1.17342	3.50214	2.32872
0.5	1.28365	3.80521	2.52156
0.6	1.43255	4.19758	2.76503
0.7	1.63085	4.69501	3.06416
0.8	1.89394	5.31897	3.42504
0.9	2.24368	6.09877	3.85509
1	2.71143	7.07465	4.36322

Рис. 2: Вывод программы в консоли

3 Исходный код

Общий файл для всех подзадач 4 лабораторной

```
#ifndef LAB4_DERIVATIVEMETHODS_H
#define LAB4_DERIVATIVEMETHODS_H

#include <iostream>
#include <vector>
#include <functional>
#include <cmath>
#include <iomanip>
#include "Matrix.h"
```

```cpp

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);
```

```cpp

        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};
}
```

```cpp

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) | |(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
```

```
#include <iostream>
#include "derivativeMethods.h"
using namespace std;

double f(double x, double y1, double y2) {
    return 2 * y1 + 4 * pow(x, 2) * pow(M_E, pow(x, 2));
}

double fExact(double x) {
    return pow(M_E, pow(x, 2)) + pow(M_E, x * pow(2, 0.5)) + pow(M_E, x * pow(2, 0.5) * (-1));
}

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 и конечно-разностный метод

4 Постановка задачи

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.

Вариант: 3

3	$x^2(x+1)\,y''-2y=0$, $y'(1)=-1$, $2y(2)-4y'(2)=4$	$y(x)=\dfrac{1}{x}+1$

Рис. 3: Входные данные

5 Результаты работы

Real value:

x	y
1	2
1.2	1.83333
1.4	1.71429
1.6	1.625
1.8	1.55556
2	1.5

Shooting method:

s	f(b, y, s)	|╨д(s)|
1	3.63451	2.13451
-1.00014	1.5	2.22045e-16

x	y
1	2
1.2	1.83335
1.4	1.7143
1.6	1.62501
1.8	1.55556
2	1.5

Runge-Romberg error estimate for Shooting Method:

x	Error y	Error y'
1	0	0
1.2	-2.20325e-06	-3.19318e-06
1.4	-3.65202e-06	-4.42292e-06
1.6	-4.89303e-06	-5.06652e-06
1.8	-6.08797e-06	-5.48978e-06
2	-7.29077e-06	-5.81095e-06

Finite difference Method:

x	y
1	2
1	1.80724
1.2	1.68677
1.4	1.60889
1.6	1.55837
1.8	1.52659
2	1.50827

Finite difference Method with h/2:

x	y
1	2
1	1.90281
1.1	1.82466
1.2	1.76086
1.3	1.70818
1.4	1.66429
1.5	1.62748
1.6	1.59645
1.7	1.57022
1.8	1.54801
1.9	1.52922
2	1.51335

Error Finite Difference Method:

x	Error y
1	0.192762
1.2	0.146567
1.4	0.105396
1.6	0.0666257
1.8	0.0289657
2	0.00826737

Рис. 4: Вывод программы в консоли

6 Исходный код

```
#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;
}
```