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:	И. К. Сайфуллин
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

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

1 Постановка задачи

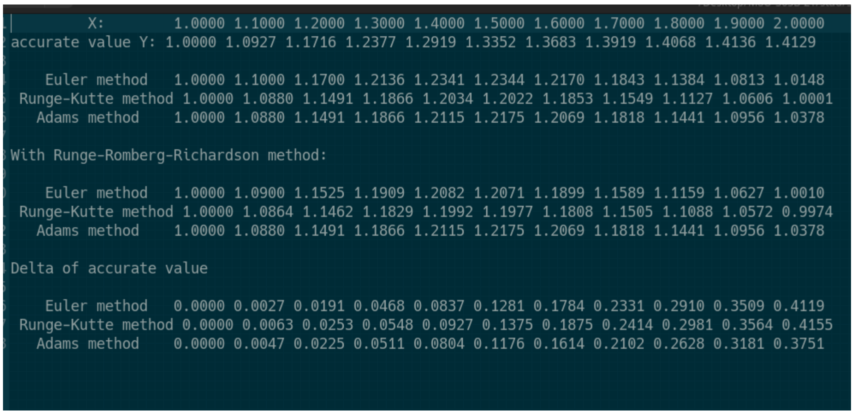
4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием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

Вариант: 19

$$y'' + \frac{1}{x}y' + \frac{2}{x}y = 0, \qquad y(1) = 1, \qquad y'(1) = 1, \qquad x \in [1,2], h = 0.1 \tag{1}$$

$$y = (cos(2) - sin(2))cos(2x^{1/2}) + (cos(2) + sin(2))cos(2x^{1/2}) \tag{2}$$

2 Результаты работы


```
           X:        1.0000 1.1000 1.2000 1.3000 1.4000 1.5000 1.6000 1.7000 1.8000 1.9000 2.0000
accurate value Y: 1.0000 1.0927 1.1716 1.2377 1.2919 1.3352 1.3683 1.3919 1.4068 1.4136 1.4129

    Euler method   1.0000 1.1000 1.1700 1.2136 1.2341 1.2344 1.2170 1.1843 1.1384 1.0813 1.0148
 Runge-Kutte method 1.0000 1.0880 1.1491 1.1866 1.2034 1.2022 1.1853 1.1549 1.1127 1.0606 1.0001
   Adams method    1.0000 1.0880 1.1491 1.1866 1.2115 1.2175 1.2069 1.1818 1.1441 1.0956 1.0378

With Runge-Romberg-Richardson method:

    Euler method   1.0000 1.0900 1.1525 1.1909 1.2082 1.2071 1.1899 1.1589 1.1159 1.0627 1.0010
 Runge-Kutte method 1.0000 1.0864 1.1462 1.1829 1.1992 1.1977 1.1808 1.1505 1.1088 1.0572 0.9974
   Adams method    1.0000 1.0880 1.1491 1.1866 1.2115 1.2175 1.2069 1.1818 1.1441 1.0956 1.0378

Delta of accurate value

    Euler method   0.0000 0.0027 0.0191 0.0468 0.0837 0.1281 0.1784 0.2331 0.2910 0.3509 0.4119
 Runge-Kutte method 0.0000 0.0063 0.0253 0.0548 0.0927 0.1375 0.1875 0.2414 0.2981 0.3564 0.4155
   Adams method    0.0000 0.0047 0.0225 0.0511 0.0804 0.1176 0.1614 0.2102 0.2628 0.3181 0.3751
```

Рис. 1: Вывод в консоли

3 Исходный код

Lab4.1.cpp

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>

using namespace std;

double function(double x, double y, double z) {
    return -z / x - 2 * y / x;
}

double accurate_function(double x) {
    double sqrt_x = sqrt(x);
    return (cos(2) - sin(2)) * cos(2 * sqrt_x) + (cos(2) + sin(2)) * sin(2 * sqrt_x);
}

vector<double> Euler(double a, double b, double h) {
    int steps = (b - a) / h + 1;
    vector<double> x(steps);
    vector<double> y(steps, 1.0);
    vector<double> z(steps, 1.0);
    x[0] = a;
    for (int i = 1; i < steps; ++i) {
        x[i] = x[i - 1] + h;
```

```
        z[i] = z[i - 1] + h * function(x[i - 1], y[i - 1], z[i - 1]);
        y[i] = y[i - 1] + h * z[i - 1];
    }
    return y;
}

vector<vector <double>> Runge_Kutty(double a, double b, double h) {
    int steps = (b - a) / h + 1;
    vector<double> x(steps);
    vector<double> y(steps, 1);
    vector<double> z(steps, 1);
    vector<double> K(4, 0.0);
    vector<double> L(4, 0.0);
    x[0] = a;
    for (int i = 1; i < steps; ++i) {
        x[i] = x[i - 1] + h;
        for (int j = 1; j < K.size(); ++j) {
            K[0] = h * z[i - 1];
            L[0] = h * function(x[i - 1], y[i - 1], z[i - 1]);
            K[j] = h * (z[i - 1] + L[j - 1] / 2);
            L[j] = h * function(x[i - 1] + h / 2, y[i - 1] + K[j - 1] / 2, z[i - 1] + L
                [j - 1] / 2);
        }
        double deltay = (K[0] + 2 * K[1] + 2 * K[2] + K[3]) / 6;
        double deltaz = (L[0] + 2 * L[1] + 2 * L[2] + L[3]) / 6;
        y[i] = y[i - 1] + deltay;
        z[i] = z[i - 1] + deltaz;
    }
    return {y, z};
}


vector<double> Adams(double a, double b, double h) {
    int steps = (b - a) / h + 1;
    vector<double> x;
    vector<double> y(steps, 0);
    vector<double> z(steps, 0);
    for (double i = a; i < b+h; i += h) {
        x.push_back(i);
    }
    vector<double> y_start = Runge_Kutty(a, a + 3 * h, h)[0];
    vector<double> z_start = Runge_Kutty(a, a + 3 * h, h)[1];
    for (int i = 0; i < y_start.size(); ++i) {
        y[i] = y_start[i];
        z[i] = z_start[i];
    }
    for (int i = 4; i < steps; ++i) {
        z[i] = (z[i - 1] + h * (
```

```
                    55 * function(x[i - 1], y[i - 1], z[i - 1]) - 59 * function(x[i - 2], y[i - 2], z[i - 2])
                    + 37 * function(x[i - 3], y[i - 3], z[i - 3]) - 9 * function(x[i - 4], y[i - 4], z[i - 4])) / 24);
        y[i] = y[i - 1] + h * z[i - 1];
    }
    return y;
}

vector<vector<double>> RRR_method(double a, double b, double h) {
    vector<double> Euler1, Runge_Kutty1, Adams1;
    vector<double> Euler_norm = Euler(a, b, h);
    vector<double> Euler_half = Euler(a, b, h / 2);
    vector<double> Runge_Kutty_norm = Runge_Kutty(a, b, h)[0];
    vector<double> Runge_Kutty_half = Runge_Kutty(a, b, h / 2)[0];
    vector<double> Adams_norm = Adams(a, b, h);
    vector<double> Adams_half = Adams(a, b, h / 2);
    int steps = (b - a) / h + 1;
    Euler1.resize(steps);
    Runge_Kutty1.resize(steps);
    Adams1.resize(steps);
    for (int i = 0; i < steps; ++i) {
        Euler1[i] = Euler_norm[i] + (Euler_half[i * 2] - Euler_norm[i]) / (1 - 0.5 * 0.5);
        Runge_Kutty1[i] = Runge_Kutty_norm[i] + (Runge_Kutty_half[i * 2] - Runge_Kutty_norm[i]) / (1 - 0.5 * 0.5);
        Adams1[i] = Adams_norm[i] + (Adams_norm[i] - Adams_norm[i]) / (1 - 0.5 * 0.5);
    }
    return {Euler1, Runge_Kutty1, Adams1};
}


int main() {
    ofstream fout("answer1.txt");
    double h = 0.1;
    double a = 1;
    double b = 2;
    fout.precision(4);
    fout << fixed;
    vector<double> x, y;
    for (double i = a; i < b+h; i += h) {
        x.push_back(i);
        y.push_back(accurate_function(i));
    }
    fout << " X: ";
    for (int i = 0; i < x.size(); ++i) {
        fout << x[i] << " ";
    }
    fout << endl;
```

```
    fout << "accurate value Y: ";
    for (int i = 0; i < y.size(); ++i) {
        fout << y[i] << " ";
    }
    fout << endl << endl;
    fout << " Euler method ";
    for (int i = 0; i < Euler(a, b, h).size(); ++i) {
        fout << Euler(a, b, h)[i] << " ";
    }
    fout << endl;
    fout << " Runge-Kutte method ";
    for (int i = 0; i < Runge_Kutty(a, b, h)[0].size(); ++i) {
        fout << Runge_Kutty(a, b, h)[0][i] << " ";
    }
    fout << endl;
    fout << " Adams method ";
    vector <double> res_adams = Adams(a, b, h);
    for (int i = 0; i < res_adams.size(); ++i) {
        fout << res_adams[i] << " ";
    }
    fout << endl << endl;
    fout << "With Runge-Romberg-Richardson method: \n" << endl;
    vector<vector<double>> result = RRR_method(a, b, h);
    fout << " Euler method ";
    for (int i = 0; i < result[0].size(); ++i) {
        fout << result[0][i] << " ";
    }
    fout << endl;
    fout << " Runge-Kutte method ";
    for (int i = 0; i < result[1].size(); ++i) {
        fout << result[1][i] << " ";
    }
    fout << endl;
    fout << " Adams method ";
    for (int i = 0; i < result[2].size(); ++i) {
        fout << result[2][i] << " ";
    }
    fout << endl << endl;
    fout << "Delta of accurate value \n" << endl;
    fout << " Euler method ";
    for (int i = 0; i < result[0].size(); ++i) {
        fout << abs(result[0][i] - y[i]) << " ";
    }
    fout << endl;
    fout << " Runge-Kutte method ";
    for (int i = 0; i < result[1].size(); ++i) {
        fout << abs(result[1][i] - y[i]) << " ";
    }
    fout << endl;
```

```
    fout << " Adams method ";
    for (int i = 0; i < result[2].size(); ++i) {
        fout << abs(result[2][i] - y[i]) << " ";
    }
    fout << endl;
    return 0;
}
```

4 Постановка задачи

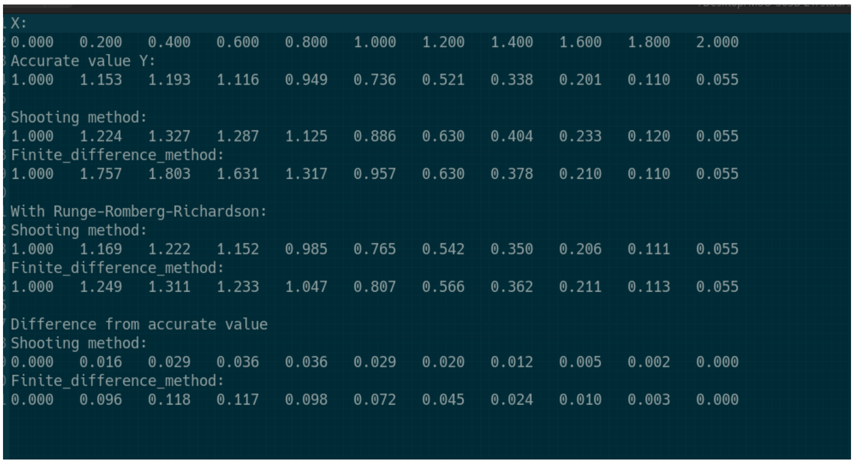
4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения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

Вариант: 19

$$y'' + 4xy' + (4x^2 + 2)y = 0, \qquad y'(0) = 1, \qquad 4y(2) - y'(2) = 23e^{-4} \tag{3}$$

$$y(x) = (1 + x)e^{-x^2} \tag{4}$$

5 Результаты работы



```
X:
0.000   0.200   0.400   0.600   0.800   1.000   1.200   1.400   1.600   1.800   2.000
Accurate value Y:
1.000   1.153   1.193   1.116   0.949   0.736   0.521   0.338   0.201   0.110   0.055

Shooting method:
1.000   1.224   1.327   1.287   1.125   0.886   0.630   0.404   0.233   0.120   0.055
Finite_difference_method:
1.000   1.757   1.803   1.631   1.317   0.957   0.630   0.378   0.210   0.110   0.055

With Runge-Romberg-Richardson:
Shooting method:
1.000   1.169   1.222   1.152   0.985   0.765   0.542   0.350   0.206   0.111   0.055
Finite_difference_method:
1.000   1.249   1.311   1.233   1.047   0.807   0.566   0.362   0.211   0.113   0.055

Difference from accurate value
Shooting method:
0.000   0.016   0.029   0.036   0.036   0.029   0.020   0.012   0.005   0.002   0.000
Finite_difference_method:
0.000   0.096   0.118   0.117   0.098   0.072   0.045   0.024   0.010   0.003   0.000
```

Рис. 2: Вывод в консоли

6 Исходный код

Lab4.2.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>

using namespace std;

double function(double x, double y, double z) {
    return -4*x*z - (4*x*x + 2) * y;
}

double accurate_function(double x) {
    return (1+x) * exp(-x*x);
}

double p(double x) {
    return 4*x;
}

double q(double x) {
    return (4*x*x + 2);
}

```

```
vector<double> Runge_Kutty(double a, double b, double h, double y0, double z0) {
    int N = ((b - a) / h) + 1;
    vector<double> x(N), y(N), z(N);
    vector<double> K(4,0), L(4,0);

    x[0] = a;
    y[0] = y0;
    z[0] = z0;

    for (int i = 1; i < N; ++i) {
        x[i] = a + i * h;
        for (int j = 1; j < 4; ++j) {
            K[0] = h * z[i - 1];
            L[0] = h * function(x[i - 1], y[i - 1], z[i - 1]);
            K[j] = h * (z[i - 1] + L[j - 1] / 2);
            L[j] = h * function(x[i - 1] + h / 2, y[i - 1] + K[j - 1] / 2, z[i - 1] + L
                [j - 1] / 2);
        }
        double deltay = (K[0] + 2 * K[1] + 2 * K[2] + K[3]) / 6;
        double deltaz = (L[0] + 2 * L[1] + 2 * L[2] + L[3]) / 6;
        y[i] = y[i - 1] + deltay;
        z[i] = z[i - 1] + deltaz;
    }
    return y;
}

vector<double> shooting_method(double a, double b, double h, double e, double y0,
    double y1) {
    double nu1 = 1.0, nu2 = 0.8;
    double f1, f2;

    f1 = Runge_Kutty(a, b, h, y0, nu1).back() - y1;
    f2 = Runge_Kutty(a, b, h, y0, nu2).back() - y1;

    while (std::abs(f2) > e) {
        double temp = nu2;
        nu2 = nu2 - f2 * (nu2 - nu1) / (f2 - f1);
        nu1 = temp;
        f1 = f2;
        f2 = Runge_Kutty(a, b, h, y0, nu2).back() - y1;
    }
    return Runge_Kutty(a, b, h, y0, nu2);
}

vector<double> finite_difference_method(double a, double b, double h, double alfa,
    double beta, double delta, double gamma, double y0, double y1) {
    int N = ((b - a) / h);
    vector<double> x(N), A, B, C, D, P(N), Q(N), ans(N);

```

```
    x[0] = a;
    x[1] = a + h;
    A.push_back(0);
    B.push_back(-2 + h * h * q(x[1]));
    C.push_back(1 + p(x[1]) * h / 2);
    D.push_back(-(1 - (p(x[1]) * h) / 2) * y0);
    for (int i = 2; i < N; ++i) {
        x[i] = a + i * h;
        A.push_back(1 - p(x[i]) * h / 2);
        B.push_back(-2 + h * h * q(x[i]));
        C.push_back(1 + p(x[i]) * h / 2);
        D.push_back(0);
    }
    A.push_back(1 - p(x[N - 2]) * h / 2);
    B.push_back(-2 + h * h * q(x[N - 2]));
    C.push_back(0);
    D.push_back(-(1 + (p(x[N - 2]) * h) / 2) * y1);

    P[0] = (-C[0] / B[0]);
    Q[0] = (D[0] / B[0]);
    for (int i = 1; i <= N; ++i) {
        P[i] = (-C[i] / (B[i] + A[i] * P[i - 1]));
        Q[i] = ((D[i] - A[i] * Q[i - 1]) / (B[i] + A[i] * P[i - 1]));
    }

    ans[N-1] = Q[N-1];
    for (int i = N - 2; i > 0; --i)
        ans[i] = P[i] * ans[i + 1] + Q[i];
    ans[0] = y0;
    ans[N] = y1;
    return ans;
}

pair<vector<double>, vector<double>> RRR_method(double a, double b, double h, double e
    , double y0, double y1, double alfa, double beta, double gamma, double delta) {
    vector<double> shooting_method1, finite_difference_method1;
    vector<double> shooting_method_norm = shooting_method(a, b, h, e, y0, y1);
    vector<double> shooting_method_half = shooting_method(a, b, h / 2, e, y0, y1);
    vector<double> finite_difference_method_norm = finite_difference_method(a, b, h,
        alfa, beta, delta, gamma, y0, y1);
    vector<double> finite_difference_method_half = finite_difference_method(a, b, h /
        2, alfa, beta, delta, gamma, y0, y1);
    int N = static_cast<int>((b - a) / h);

    shooting_method1.resize(N + 1);
    finite_difference_method1.resize(N + 1);

    for (int i = 0; i <= N; ++i) {
```

```
        shooting_method1[i] = shooting_method_norm[i] + (shooting_method_half[2 * i] -
            shooting_method_norm[i]) / (1 - 0.5 * 0.5);
        finite_difference_method1[i] = finite_difference_method_norm[i] + (
            finite_difference_method_half[2 * i] - finite_difference_method_norm[i]) /
            (1 - 0.5 * 0.5);
    }
    return make_pair(shooting_method1, finite_difference_method1);
}



int main() {
    double a = 0, b = 2, h = 0.2, e = 0.001, alfa = 0, beta = 1, delta = -1, gamma = 4;
    double y0 = 1, y1 = 3 * exp(-4);
    ofstream fout("answer2.txt");
    vector<double> x, y, y_exact, shooting, finite_difference;
    vector<double> shooting_method_result, finite_difference_method_result;

    int N = ((b - a) / h);
    x.resize(N + 1);
    y.resize(N + 1);
    y_exact.resize(N + 1);
    fout.precision(3);
    fout << fixed;
    for (int i = 0; i <= N; ++i) {
        x[i] = a + i * h;
        y_exact[i] = accurate_function(x[i]);
    }

    shooting_method_result = shooting_method(a, b, h, e, y0, y1);
    finite_difference_method_result = finite_difference_method(a, b, h, alfa, beta,
        delta, gamma, y0, y1);
    tie(shooting, finite_difference) = RRR_method(a, b, h, e, y0, y1, alfa, beta, delta
        , gamma);

    fout << "X:" << endl;
    for (int i = 0; i <= N; ++i)
        fout << x[i] << "\t";
    fout << endl;

    fout << "Accurate value Y:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << y_exact[i] << "\t";
    fout << endl << endl;

    fout << "Shooting method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << shooting_method_result[i] << "\t";
    fout << endl;
```

```
    fout << "Finite_difference_method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << finite_difference_method_result[i] << "\t";
    fout << endl << endl;

    fout << "With Runge-Romberg-Richardson:" << endl;
    fout << "Shooting method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << shooting[i] << "\t";
    fout << endl;

    fout << "Finite_difference_method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << finite_difference[i] << "\t";
    fout << endl << endl;

    fout << "Difference from accurate value" << endl;
    fout << "Shooting method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << abs(shooting[i] - y_exact[i]) << "\t";
    fout << endl;

    fout << "Finite_difference_method:"<< endl;
    for (int i = 0; i <= N; ++i)
        fout << abs(finite_difference[i] - y_exact[i]) << "\t";
    fout << endl;

    return 0;
}
```