

**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: Наумов Г.К.  
Преподаватель: Пивоваров Д.Е.  
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: 16**

16	$(x^2 - 1)y'' - 2xy' + 2y = 0,$ $y(2) = 7,$ $y'(2) = 5,$ $x \in [2, 3], h = 0.1$	$y = x^2 + x + 1$
----	---	-------------------

Рис. 1: Входные данные

### 2 Результаты работы

```

1 Euler method:
2 x: 2, my solution: 7, exact solution: 10.3891, error: 3.38906
3 x: 2.1, my solution: 7.5, exact solution: 11.2662, error: 3.76617
4 x: 2.2, my solution: 8.02, exact solution: 12.225, error: 4.20501
5 x: 2.3, my solution: 8.56006, exact solution: 13.2742, error: 4.71412
6 x: 2.4, my solution: 9.12023, exact solution: 14.4232, error: 5.30295
7 x: 2.5, my solution: 9.70056, exact solution: 15.6825, error: 5.98194
8 x: 2.6, my solution: 10.3011, exact solution: 17.0637, error: 6.76266
9 x: 2.7, my solution: 10.9218, exact solution: 18.5797, error: 7.65788
10 x: 2.8, my solution: 11.5629, exact solution: 20.2446, error: 8.68176
11 x: 2.9, my solution: 12.2242, exact solution: 22.0741, error: 9.84991
12
13 Euler method error:
14 Exact solution in x = 3: 24.0855
15 My solution in x = 3: 12.2242
16 Error: 0.0486225
17
18 Runge-Kutta method:
19 x: 2, my solution: 7, exact solution: 10.3891, error: 3.38906
20 x: 2.1, my solution: 7.51, exact solution: 11.2662, error: 3.75617
21 x: 2.2, my solution: 8.04, exact solution: 12.225, error: 4.18501
22 x: 2.3, my solution: 8.59, exact solution: 13.2742, error: 4.68418

```

```

23 | x: 2.4, my solution: 9.16, exact solution: 14.4232, error: 5.26318
24 | x: 2.5, my solution: 9.75, exact solution: 15.6825, error: 5.93249
25 | x: 2.6, my solution: 10.36, exact solution: 17.0637, error: 6.70374
26 | x: 2.7, my solution: 10.99, exact solution: 18.5797, error: 7.58973
27 | x: 2.8, my solution: 11.64, exact solution: 20.2446, error: 8.60465
28 | x: 2.9, my solution: 12.31, exact solution: 22.0741, error: 9.76414
29 |
30 | Runge-Kutta method error:
31 | Exact solution in x = 3: 24.0855
32 | My solution in x = 3: 12.31
33 | Error: 0.0459999
34 |
35 | Adams method:
36 | x: 2, my solution: 7, exact solution: 10.3891, error: 3.38906
37 | x: 2.1, my solution: 7.51, exact solution: 11.2662, error: 3.75617
38 | x: 2.2, my solution: 8.04, exact solution: 12.225, error: 4.18501
39 | x: 2.3, my solution: 8.59, exact solution: 13.2742, error: 4.68418
40 | x: 2.4, my solution: 9.16, exact solution: 14.4232, error: 5.26318
41 | x: 2.5, my solution: 9.75, exact solution: 15.6825, error: 5.93249
42 | x: 2.6, my solution: 10.36, exact solution: 17.0637, error: 6.70374
43 | x: 2.7, my solution: 10.9876, exact solution: 18.5797, error: 7.59216
44 | x: 2.8, my solution: 11.6376, exact solution: 20.2446, error: 8.60702
45 | x: 2.9, my solution: 12.305, exact solution: 22.0741, error: 9.76916
46 |
47 | Adams method error:
48 | Exact solution in x = 3: 24.0855
49 | My solution in x = 3: 12.305
50 | Error: 0.0457717

```

### 3 Исходный код

```

1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 |
5 |
6 | void eulerMethod(double h, double x0, double y10, double y20, double x_end, std::
    vector<double>& x_vals, std::vector<double>& y1_vals, std::vector<double>& y2_vals
    ) {
7 |     double x = x0;
8 |     double y1 = y10;
9 |     double y2 = y20;
10 |
11 |     while (x <= x_end) {
12 |         x_vals.push_back(x);
13 |         y1_vals.push_back(y1);
14 |         y2_vals.push_back(y2);
15 |
16 |         double y1_new = y1 + h * y2;

```

```

17     double y2_new = y2 + h * ((2 * x * y2 - 2 * y1) / (x * x - 1));
18
19     y1 = y1_new;
20     y2 = y2_new;
21     x += h;
22 }
23 }
24
25 void rungeKuttaMethod(double h, double x0, double y10, double y20, double x_end, std::
    vector<double>& x_vals, std::vector<double>& y1_vals, std::vector<double>& y2_vals
    ) {
26     double x = x0;
27     double y1 = y10;
28     double y2 = y20;
29
30     while (x <= x_end) {
31         x_vals.push_back(x);
32         y1_vals.push_back(y1);
33         y2_vals.push_back(y2);
34
35         double k1_1 = h * y2;
36         double k1_2 = h * ((2 * x * y2 - 2 * y1) / (x * x - 1));
37
38         double k2_1 = h * (y2 + 0.5 * k1_2);
39         double k2_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k1_2) - 2 * (y1 + 0.5 *
            k1_1)) / ((x + 0.5 * h) * (x + 0.5 * h) - 1));
40
41         double k3_1 = h * (y2 + 0.5 * k2_2);
42         double k3_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k2_2) - 2 * (y1 + 0.5 *
            k2_1)) / ((x + 0.5 * h) * (x + 0.5 * h) - 1));
43
44         double k4_1 = h * (y2 + k3_2);
45         double k4_2 = h * ((2 * (x + h) * (y2 + k3_2) - 2 * (y1 + k3_1)) / ((x + h) * (
            x + h) - 1));
46
47         y1 += (k1_1 + 2 * k2_1 + 2 * k3_1 + k4_1) / 6;
48         y2 += (k1_2 + 2 * k2_2 + 2 * k3_2 + k4_2) / 6;
49         x += h;
50     }
51 }
52
53 void adamsMethod(double h, double x0, double y10, double y20, double x_end, std::
    vector<double>& x_vals, std::vector<double>& y1_vals, std::vector<double>& y2_vals
    ) {
54     double x = x0;
55     double y1 = y10;
56     double y2 = y20;
57
58     std::vector<double> f1, f2;

```

```

59
60 // -
61 for (int i = 0; i < 4; i++) {
62     x_vals.push_back(x);
63     y1_vals.push_back(y1);
64     y2_vals.push_back(y2);
65
66     double k1_1 = h * y2;
67     double k1_2 = h * ((2 * x * y2 - 2 * y1) / (x * x - 1));
68
69     double k2_1 = h * (y2 + 0.5 * k1_2);
70     double k2_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k1_2) - 2 * (y1 + 0.5 *
71         k1_1)) / ((x + 0.5 * h) * (x + 0.5 * h) - 1));
72
73     double k3_1 = h * (y2 + 0.5 * k2_2);
74     double k3_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k2_2) - 2 * (y1 + 0.5 *
75         k2_1)) / ((x + 0.5 * h) * (x + 0.5 * h) - 1));
76
77     double k4_1 = h * (y2 + k3_2);
78     double k4_2 = h * ((2 * (x + h) * (y2 + k3_2) - 2 * (y1 + k3_1)) / ((x + h) * (
79         x + h) - 1));
80
81     y1 += (k1_1 + 2 * k2_1 + 2 * k3_1 + k4_1) / 6;
82     y2 += (k1_2 + 2 * k2_2 + 2 * k3_2 + k4_2) / 6;
83     x += h;
84
85     f1.push_back(y2);
86     f2.push_back((2 * x * y2 - 2 * y1) / (x * x - 1));
87 }
88
89 while (x <= x_end) {
90     x_vals.push_back(x);
91     y1_vals.push_back(y1);
92     y2_vals.push_back(y2);
93
94     double y1_new = y1 + h / 24 * (55 * f1.back() - 59 * f1[f1.size() - 2] + 37 *
95         f1[f1.size() - 3] - 9 * f1[f1.size() - 4]);
96     double y2_new = y2 + h / 24 * (55 * f2.back() - 59 * f2[f2.size() - 2] + 37 *
97         f2[f2.size() - 3] - 9 * f2[f2.size() - 4]);
98
99     f1.push_back(y2_new);
100     f2.push_back((2 * x * y2_new - 2 * y1_new) / (x * x - 1));
101
102     y1 = y1_new;
103     y2 = y2_new;
104     x += h;
105 }
106 }

```

```

103 double exactSolution(double x) {
104     return x + 1 + exp(x);
105 }
106
107 int main() {
108     double h = 0.1;
109     double x0 = 2.0;
110     double y10 = 7.0;
111     double y20 = 5.0;
112     double x_end = 3.0;
113     std::vector<double> x_vals, y1_vals, y2_vals;
114     std::vector<double> x_vals_half, y1_vals_half, y2_vals_half;
115
116     std::cout << "Euler method: " << std::endl;
117     eulerMethod(h, x0, y10, y20, x_end, x_vals, y1_vals, y2_vals);
118     for (size_t i = 0; i < x_vals.size(); i++) {
119         double y_exact = exactSolution(x_vals[i]);
120         std::cout << "x: " << x_vals[i] << ", my solution: " << y1_vals[i] << ", exact
            solution: " << y_exact << ", error: " << fabs(y1_vals[i] - y_exact) << std
            ::endl;
121     }
122     std::cout << std::endl;
123     std::cout << "Euler method error: " << std::endl;
124     eulerMethod(h / 2, x0, y10, y20, x_end, x_vals_half, y1_vals_half, y2_vals_half);
125     double error = fabs(y1_vals_half[y1_vals_half.size() - 1] - y1_vals[y1_vals.size()
        - 1]) / (pow(2, 4) - 1);
126     std::cout << "Exact solution in x = " << x_end << ": " << exactSolution(x_end) <<
        std::endl;
127     std::cout << "My solution in x = " << x_end << ": " << y1_vals[y1_vals.size() - 1]
        << std::endl;
128     std::cout << "Error: " << error << std::endl;
129     std::cout << std::endl;
130     x_vals.clear();
131     y1_vals.clear();
132     y2_vals.clear();
133     x_vals_half.clear();
134     y1_vals_half.clear();
135     y2_vals_half.clear();
136
137     std::cout << "Runge-Kutta method: " << std::endl;
138     rungeKuttaMethod(h, x0, y10, y20, x_end, x_vals, y1_vals, y2_vals);
139     for (size_t i = 0; i < x_vals.size(); i++) {
140         double y_exact = exactSolution(x_vals[i]);
141         std::cout << "x: " << x_vals[i] << ", my solution: " << y1_vals[i] << ", exact
            solution: " << y_exact << ", error: " << fabs(y1_vals[i] - y_exact) << std
            ::endl;
142     }
143     std::cout << std::endl;
144     std::cout << "Runge-Kutta method error: " << std::endl;

```

```

145     rungeKuttaMethod(h / 2, x0, y10, y20, x_end, x_vals_half, y1_vals_half,
        y2_vals_half);
146     error = fabs(y1_vals_half[y1_vals_half.size() - 1] - y1_vals[y1_vals.size() - 1]) /
        (pow(2, 4) - 1);
147     std::cout << "Exact solution in x = " << x_end << ": " << exactSolution(x_end) <<
        std::endl;
148     std::cout << "My solution in x = " << x_end << ": " << y1_vals[y1_vals.size() - 1]
        << std::endl;
149     std::cout << "Error: " << error << std::endl;
150     std::cout << std::endl;
151     x_vals.clear();
152     y1_vals.clear();
153     y2_vals.clear();
154     x_vals_half.clear();
155     y1_vals_half.clear();
156     y2_vals_half.clear();
157
158     std::cout << "Adams method: " << std::endl;
159     adamsMethod(h, x0, y10, y20, x_end, x_vals, y1_vals, y2_vals);
160     for (size_t i = 0; i < x_vals.size(); i++) {
161         double y_exact = exactSolution(x_vals[i]);
162         std::cout << "x: " << x_vals[i] << ", my solution: " << y1_vals[i] << ", exact
            solution: " << y_exact << ", error: " << fabs(y1_vals[i] - y_exact) << std
                ::endl;
163     }
164     std::cout << std::endl;
165     std::cout << "Adams method error: " << std::endl;
166     adamsMethod(h / 2, x0, y10, y20, x_end, x_vals_half, y1_vals_half, y2_vals_half);
167     error = fabs(y1_vals_half[y1_vals_half.size() - 1] - y1_vals[y1_vals.size() - 1]) /
        (pow(2, 4) - 1);
168     std::cout << "Exact solution in x = " << x_end << ": " << exactSolution(x_end) <<
        std::endl;
169     std::cout << "My solution in x = " << x_end << ": " << y1_vals[y1_vals.size() - 1]
        << std::endl;
170     std::cout << "Error: " << error << std::endl;
171     std::cout << std::endl;
172     x_vals.clear();
173     y1_vals.clear();
174     y2_vals.clear();
175     x_vals_half.clear();
176     y1_vals_half.clear();
177     y2_vals_half.clear();
178
179     return 0;
180 }

```

## 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: 16**

16	$y'' - \operatorname{tg} x y' + 2y = 0,$ $y(0) = 2,$ $y\left(\frac{\pi}{6}\right) = 2.5 - 0.5 \cdot \ln 3$	$y(x) = \sin x + 2 - \sin x \cdot \ln \left( \frac{1 + \sin x}{1 - \sin x} \right)$
----	--	---

Рис. 2: Входные данные

### 5 Результаты работы

```

1 Shooting method:
2 x: 0, my solution: 2,| exact solution: 2,| error: 0
3 x: 0.05, my solution: 2.04206,| exact solution: 2.04498,| error: 0.0029172
4 x: 0.1, my solution: 2.07401,| exact solution: 2.07983,| error: 0.0058271
5 x: 0.15, my solution: 2.09572,| exact solution: 2.10444,| error: 0.00872244
6 x: 0.2, my solution: 2.10707,| exact solution: 2.11867,| error: 0.011596
7 x: 0.25, my solution: 2.10795,| exact solution: 2.12239,| error: 0.0144405
8 x: 0.3, my solution: 2.09824,| exact solution: 2.11549,| error: 0.017249
9 x: 0.35, my solution: 2.0778,| exact solution: 2.09781,| error: 0.0200143
10 x: 0.4, my solution: 2.0465,| exact solution: 2.06923,| error: 0.0227296
11 x: 0.45, my solution: 2.00419,| exact solution: 2.02957,| error: 0.0253881
12 x: 0.5, my solution: 1.95069,| exact solution: 1.97868,| error: 0.0279831
13
14 Shooting method error:
15 Exact solution in x = 0.523599: 1.95069
16 My solution in x = 0.523599: 1.95069
17 Error: 2.66454e-16
18
19 Finite Difference method:
20 x: 0, my solution: 2,| exact solution: 2,| error: 0
21 x: 0.05, my solution: 2.04154,| exact solution: 2.04498,| error: 0.00344151
22 x: 0.1, my solution: 2.07278,| exact solution: 2.07983,| error: 0.0070568

```



```

23 x: 0.15, my solution: 2.09352,| exact solution: 2.10444,| error: 0.0109164
24 x: 0.2, my solution: 2.10368,| exact solution: 2.11867,| error: 0.014985
25 x: 0.25, my solution: 2.10327,| exact solution: 2.12239,| error: 0.019119
26 x: 0.3, my solution: 2.09242,| exact solution: 2.11549,| error: 0.0230651
27 x: 0.35, my solution: 2.07135,| exact solution: 2.09781,| error: 0.0264586
28 x: 0.4, my solution: 2.04041,| exact solution: 2.06923,| error: 0.0288224
29 x: 0.45, my solution: 2.00001,| exact solution: 2.02957,| error: 0.0295658
30 x: 0.5, my solution: 1.95069,| exact solution: 1.97868,| error: 0.0279831
31
32 Finite Difference method error:
33 Exact solution in x = 0.523599: 1.95069
34 My solution in x = 0.523599: 1.95069
35 Error: 0

```

## 6 Исходный код

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4
5
6  void rungeKutta(double h, double x0, double y0, double dy0, double x_end, std::vector<
    double>& x_vals, std::vector<double>& y_vals) {
7      double x = x0;
8      double y = y0;
9      double dy = dy0;
10
11     while (x <= x_end) {
12         x_vals.push_back(x);
13         y_vals.push_back(y);
14
15         double k1 = h * dy;
16         double l1 = h * (tan(x) * dy - 2 * y);
17         double k2 = h * (dy + 0.5 * l1);
18         double l2 = h * (tan(x + 0.5 * h) * (dy + 0.5 * l1) - 2 * (y + 0.5 * k1));
19         double k3 = h * (dy + 0.5 * l2);
20         double l3 = h * (tan(x + 0.5 * h) * (dy + 0.5 * l2) - 2 * (y + 0.5 * k2));
21         double k4 = h * (dy + l3);
22         double l4 = h * (tan(x + h) * (dy + l3) - 2 * (y + k3));
23
24         y += (k1 + 2 * k2 + 2 * k3 + k4) / 6;
25         dy += (l1 + 2 * l2 + 2 * l3 + l4) / 6;
26         x += h;
27     }
28 }
29
30 double shootingMethod(double h, double x0, double y0, double x_end, double y_end,
    double initial_guess) {
31     double tolerance = 1e-6;

```

```

32     double guess1 = initial_guess;
33     double guess2 = initial_guess + 0.1;
34
35     double f1, f2;
36
37     while (true) {
38         std::vector<double> x_vals, y_vals1, y_vals2;
39         rungeKutta(h, x0, y0, guess1, x_end, x_vals, y_vals1);
40         rungeKutta(h, x0, y0, guess2, x_end, x_vals, y_vals2);
41
42         f1 = y_vals1.back() - y_end;
43         f2 = y_vals2.back() - y_end;
44
45         if (fabs(f1) < tolerance) {
46             return guess1;
47         }
48         if (fabs(f2) < tolerance) {
49             return guess2;
50         }
51
52         double guess_new = guess1 - f1 * (guess2 - guess1) / (f2 - f1);
53         guess1 = guess2;
54         guess2 = guess_new;
55     }
56 }
57
58 void finiteDifferenceMethod(double h, double x0, double y0, double x_end, double y_end
59     , std::vector<double>& x_vals, std::vector<double>& y_vals) {
60     int n = (x_end - x0) / h;
61     std::vector<double> a(n + 1), b(n + 1), c(n + 1), d(n + 1), y(n + 1);
62
63     for (int i = 0; i <= n; ++i) {
64         x_vals.push_back(x0 + i * h);
65     }
66
67     a[0] = 0;
68     b[0] = 1;
69     c[0] = 0;
70     d[0] = y0;
71
72     for (int i = 1; i < n; ++i) {
73         double x = x0 + i * h;
74         a[i] = 1 / (h * h) - tan(x) / (2 * h);
75         b[i] = -2 / (h * h) + 2;
76         c[i] = 1 / (h * h) + tan(x) / (2 * h);
77         d[i] = 0;
78     }
79     a[n] = 0;

```

```

80     b[n] = 1;
81     c[n] = 0;
82     d[n] = y_end;
83
84     for (int i = 1; i <= n; ++i) {
85         double m = a[i] / b[i - 1];
86         b[i] -= m * c[i - 1];
87         d[i] -= m * d[i - 1];
88     }
89
90     y[n] = d[n] / b[n];
91     for (int i = n - 1; i >= 0; --i) {
92         y[i] = (d[i] - c[i] * y[i + 1]) / b[i];
93     }
94
95     for (int i = 0; i <= n; ++i) {
96         y_vals.push_back(y[i]);
97     }
98 }
99
100 double exactSolution(double x) {
101     return sin(x) + 2 - sin(x) * log((1 + sin(x)) / (1 - sin(x)));
102 }
103
104 int main() {
105     double h = 0.05;
106     double x0 = 0.0;
107     double y0 = 2.0;
108     double x_end = M_PI / 6;
109     double y_end = 2.5 - 0.5 * log(3.0);
110     std::vector<double> x_vals, y_vals;
111     std::vector<double> x_vals_half, y_vals_half;
112
113     std::cout << "Shooting method: " << std::endl;
114     double initial_guess = 0.0;
115     double dy0 = shootingMethod(h, x0, y0, x_end, y_end, initial_guess);
116     rungeKutta(h, x0, y0, dy0, x_end, x_vals, y_vals);
117     for (size_t i = 0; i < x_vals.size(); i++) {
118         double y_exact = exactSolution(x_vals[i]);
119         std::cout << "x: " << x_vals[i] << ", my solution: " << y_vals[i] << ",| exact
            solution: " << y_exact << ",| error: " << fabs(y_vals[i] - y_exact) << std
                ::endl;
120     }
121     std::cout << std::endl;
122     std::cout << "Shooting method error: " << std::endl;
123     dy0 = shootingMethod(h/2, x0, y0, x_end, y_end, initial_guess);
124     rungeKutta(h/2, x0, y0, dy0, x_end, x_vals_half, y_vals_half);
125     double error = fabs(y_vals_half[y_vals_half.size() - 1] - y_vals[y_vals.size() -
        1]) / (pow(2, 4) - 1);

```

```

126     std::cout << "Exact solution in x = " << x_end << ": " << exactSolution(x_end) <<
        std::endl;
127     std::cout << "My solution in x = " << x_end << ": " << y_vals[y_vals.size() - 1] <<
        std::endl;
128     std::cout << "Error: " << error << std::endl;
129     std::cout << std::endl;
130     x_vals.clear();
131     y_vals.clear();
132     x_vals_half.clear();
133     y_vals_half.clear();
134
135
136     std::cout << "Finite Difference method: " << std::endl;
137     finiteDifferenceMethod(h, x0, y0, x_end, y_end, x_vals, y_vals);
138     for (size_t i = 0; i < x_vals.size(); i++) {
139         double y_exact = exactSolution(x_vals[i]);
140         std::cout << "x: " << x_vals[i] << ", my solution: " << y_vals[i] << ",| exact
            solution: " << y_exact << ",| error: " << fabs(y_vals[i] - y_exact) << std
                ::endl;
141     }
142     std::cout << std::endl;
143     std::cout << "Finite Difference method error: " << std::endl;
144     finiteDifferenceMethod(h/2, x0, y0, x_end, y_end, x_vals_half, y_vals_half);
145     error = fabs(y_vals_half[y_vals_half.size() - 1] - y_vals[y_vals.size() - 1]) / (
        pow(2, 4) - 1);
146     std::cout << "Exact solution in x = " << x_end << ": " << exactSolution(x_end) <<
        std::endl;
147     std::cout << "My solution in x = " << x_end << ": " << y_vals[y_vals.size() - 1] <<
        std::endl;
148     std::cout << "Error: " << error << std::endl;
149     std::cout << std::endl;
150     x_vals.clear();
151     y_vals.clear();
152     x_vals_half.clear();
153     y_vals_half.clear();
154     return 0;
155 }

```