

**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

```

1 | #ifndef LAB4_DERIVATIVEMETHODS_H
2 | #define LAB4_DERIVATIVEMETHODS_H
3 |
4 | #include <iostream>
5 | #include <vector>
6 | #include <functional>
7 | #include <cmath>
8 | #include <iomanip>
9 | #include "Matrix.h"

```

```

10
11 using namespace std;
12 using namespace numeric;
13
14 vector<vector<double>>
15 eulerMethod(const std::function<double(double, double, double)> &f, double x0, double
    y1_0, double y2_0, double h,
16             int n) {
17     vector<double> x(n), y1(n), y2(n);
18     x[0] = x0;
19     y1[0] = y1_0;
20     y2[0] = y2_0;
21
22     for (int i = 1; i < n; ++i) {
23         x[i] = x[i - 1] + h;
24         y1[i] = y1[i - 1] + h * y2[i - 1];
25         y2[i] = y2[i - 1] + h * f(x[i - 1], y1[i - 1], y2[i - 1]);
26     }
27
28     return vector<vector<double>>{x, y1, y2};
29
30 }
31
32
33 vector<vector<double>>
34 rungeKutta4(const std::function<double(double, double, double)> &f, double x0, double
    y1_0, double y2_0, double h,
35             int n) {
36     vector<double> x(n), y1(n), y2(n);
37     x[0] = x0;
38     y1[0] = y1_0;
39     y2[0] = y2_0;
40
41     for (int i = 1; i < n; ++i) {
42         x[i] = x[i - 1] + h;
43
44         double k1_y1 = h * y2[i - 1];
45         double k1_y2 = h * f(x[i - 1], y1[i - 1], y2[i - 1]);
46
47         double k2_y1 = h * (y2[i - 1] + 0.5 * k1_y2);
48         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);
49
50         double k3_y1 = h * (y2[i - 1] + 0.5 * k2_y2);
51         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);
52
53         double k4_y1 = h * (y2[i - 1] + k3_y2);
54         double k4_y2 = h * f(x[i - 1] + h, y1[i - 1] + k3_y1, y2[i - 1] + k3_y2);

```

```

55
56     y1[i] = y1[i - 1] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
57     y2[i] = y2[i - 1] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
58 }
59 return vector<vector<double>>{x, y1, y2};
60 }
61
62 vector<vector<double>>
63 adams4(const std::function<double(double, double, double)> &f, double x0, double y1_0,
        double y2_0, double h, int n) {
64     vector<double> x(n), y1(n), y2(n);
65     x[0] = x0;
66     y1[0] = y1_0;
67     y2[0] = y2_0;
68
69     for (int i = 1; i < 4; ++i) {
70         x[i] = x[i - 1] + h;
71
72         double k1_y1 = h * y2[i - 1];
73         double k1_y2 = h * f(x[i - 1], y1[i - 1], y2[i - 1]);
74
75         double k2_y1 = h * (y2[i - 1] + 0.5 * k1_y2);
76         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);
77
78         double k3_y1 = h * (y2[i - 1] + 0.5 * k2_y2);
79         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);
80
81         double k4_y1 = h * (y2[i - 1] + k3_y2);
82         double k4_y2 = h * f(x[i - 1] + h, y1[i - 1] + k3_y1, y2[i - 1] + k3_y2);
83
84         y1[i] = y1[i - 1] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
85         y2[i] = y2[i - 1] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
86     }
87
88     for (int i = 4; i < n; ++i) {
89         x[i] = x[i - 1] + h;
90
91         y1[i] = y1[i - 1] + h / 24 * (55 * y2[i - 1] - 59 * y2[i - 2] + 37 * y2[i - 3]
            - 9 * y2[i - 4]);
92         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]) +
93             37 * f(x[i - 3], y1[i - 3], y2[i - 3]) - 9 * f(x[i -
            4], y1[i - 4], y2[i - 4]));
94     }
95
96     return vector<vector<double>>{x, y1, y2};
97 }

```

```

98
99 void rungeRomberg(
100     const std::function<vector<vector<double>>>(const std::function<double(double,
101         double, double)> &, double,
102         double, double, double, int)> &method,
103     const std::function<double(double, double, double)> &f, double x0, double y1_0,
104         double y2_0, double h, int n) {
105     vector<double> y1_h, y2_h, y1_h2, y2_h2;
106
107     auto res1 = method(f, x0, y1_0, y2_0, h, n);
108     auto res2 = method(f, x0, y1_0, y2_0, h / 2, 2 * n);
109     y1_h = res1[1];
110     y2_h = res1[2];
111     y1_h2 = res2[1];
112     y2_h2 = res2[2];
113
114     const int width = 15;
115     const int precision = 6;
116
117     cout << "+-----+-----+-----+-----+-----+-----+" << endl;
118     cout << "| x | Error y | Error y' |" << endl;
119     cout << "+-----+-----+-----+-----+-----+-----+" << endl;
120     for (int i = 0; i < n; ++i) {
121         double x_i = x0 + i * h;
122         double error_y1 = (y1_h2[2 * i] - y1_h[i]) / (pow(2, 4) - 1);
123         double error_y2 = (y2_h2[2 * i] - y2_h[i]) / (pow(2, 4) - 1);
124         cout << "| " << setw(width - 2) << setprecision(precision) << x_i << " | "
125             << setw(width - 2) << setprecision(precision) << error_y1 << " | "
126             << setw(width - 2) << setprecision(precision) << error_y2 << " | " << endl;
127     }
128     cout << "+-----+-----+-----+-----+-----+-----+" << endl;
129 }
130
131 void shootingMethod(const std::function<double(double, double, double)>& f, double a,
132     double b, double alpha, double beta, double eps, double h, int n) {
133     vector<vector<double>> result;
134     vector<double> y_trial(n);
135     vector<double> s_values;
136     double s0 = 0;
137     double s1 = 1.0;
138     double y_b0, y_b1, s_new;
139
140     auto result0 = rungeKutta4(f, a, alpha, s0, h, n);
141     y_b0 = result0[1].back();
142     auto result1 = rungeKutta4(f, a, alpha, s1, h, n);
143     y_b1 = result1[1].back();
144
145     cout << "+-----+-----+-----+-----+-----+-----+" << endl;

```

```

144     cout << "| s | f(b, y, s) | |(s)| |" << endl;
145     cout << "+-----+-----+-----+" << endl;
146
147     while (abs(y_b1 - beta) > eps) {
148         s_new = s1 - (y_b1 - beta) * (s1 - s0) / (y_b1 - y_b0);
149         s_values.push_back(s_new);
150         cout << "| " << setw(13) << setprecision(6) << s1 << " | "
151             << setw(13) << setprecision(6) << y_b1 << " | "
152             << setw(13) << setprecision(6) << abs(y_b1 - beta) << " |" << endl;
153         s0 = s1;
154         y_b0 = y_b1;
155         s1 = s_new;
156         result = rungeKutta4(f, a, alpha, s1, h, n);
157         y_b1 = result[1].back();
158     }
159     cout << "| " << setw(13) << setprecision(6) << s1 << " | "
160         << setw(13) << setprecision(6) << y_b1 << " | "
161         << setw(13) << setprecision(6) << abs(y_b1 - beta) << " |" << endl;
162     cout << "+-----+-----+-----+" << endl;
163
164     cout << "+-----+-----+-----+" << endl;
165     cout << "| x | y |" << endl;
166     cout << "+-----+-----+-----+" << endl;
167
168     for (int i = 0; i < n; ++i) {
169         cout << "| " << setw(13) << setprecision(6) << result[0][i] << " | "
170             << setw(13) << setprecision(6) << result[1][i] << " |" << endl;
171     }
172     cout << "+-----+-----+-----+" << endl;
173 }
174
175 void finiteDifferenceMethod(double a, double b, double y0, double y1, double h,
176                             const std::function<double(double)>& p,
177                             const std::function<double(double)>& q,
178                             const std::function<double(double)>& f,
179                             vector<double>& x, vector<double>& y) {
180     int n = static_cast<int>((b - a) / h) + 1;
181     x.resize(n);
182     y.resize(n);
183     vector<double> rhs(n);
184
185     for (int i = 0; i < n; ++i) {
186         x[i] = a + i * h;
187     }
188     rhs[0] = h * h * f(x[0]) - (1 - p(x[0]) * h / 2) * y0;
189     rhs[n - 1] = h * h * f(x[n - 1]) - (1 + p(x[n - 1]) * h / 2) * y1;
190     for (int i = 1; i < n - 1; ++i) {
191         rhs[i] = h * h * f(x[i]);
192     }

```

```

193 Matrix<double> A(n, n);
194 for (int i = 0; i < n; ++i) {
195     A[i][i] = -2 + h * h * q(x[i]);
196     if (i > 0) A[i][i - 1] = 1 - p(x[i]) * h / 2;
197     if (i < n - 1) A[i][i + 1] = (1 + p(x[i]) * h / 2);
198 }
199
200 y = tridiagonalSolve(A, rhs);
201
202 cout << "+-----+" << endl;
203 cout << "| x | y |" << endl;
204 cout << "+-----+" << endl;
205 cout << "| " << setw(13) << setprecision(6) << a << " | "
206     << setw(13) << setprecision(6) << y0 << " |" << endl;
207 for (int i = 0; i < n; ++i) {
208     cout << "| " << setw(13) << setprecision(6) << x[i] << " | "
209         << setw(13) << setprecision(6) << y[i] << " |" << endl;
210 }
211 cout << "+-----+" << endl;
212 }
213
214
215 void printTable(double a, double b, double h, std::function<double(double)> yFunc) {
216     vector<double> x, y;
217     for (double xi = a; xi <= b; xi += h) {
218         x.push_back(xi);
219         y.push_back(yFunc(xi));
220     }
221
222     cout << "+-----+" << endl;
223     cout << "| x | y |" << endl;
224     cout << "+-----+" << endl;
225
226     for (size_t i = 0; i < x.size(); ++i) {
227         cout << "| " << setw(13) << setprecision(6) << x[i] << " | "
228             << setw(13) << setprecision(6) << y[i] << " |" << endl;
229     }
230     cout << "+-----+" << endl;
231 }
232
233 void printResult(const vector<vector<double>> &res) {
234     const int width = 15;
235     const int precision = 6;
236
237     cout << "+-----+" << endl;
238     cout << "| x | y | y' |" << endl;
239     cout << "+-----+" << endl;
240
241     for (int i = 0; i < res[0].size(); ++i) {

```



```

242         cout << "| " << setw(width - 2) << setprecision(precision) << res[0][i] << " |
243         << setw(width - 2) << setprecision(precision) << res[1][i] << " | "
244         << setw(width - 2) << setprecision(precision) << res[2][i] << " |" << endl;
245     }
246
247     cout << "+-----+-----+-----+" << endl;
248 }
249
250 #endif //LAB4_DERIVATIVEMETHODS_H

1 #include <iostream>
2 #include "derivativeMethods.h"
3 using namespace std;
4
5 double f(double x, double y1, double y2) {
6     return 2 * y1 + 4 * pow(x, 2) * pow(M_E, pow(x, 2));
7 }
8
9 double fExact(double x) {
10     return pow(M_E, pow(x, 2)) + pow(M_E, x * pow(2, 0.5)) + pow(M_E, x * pow(2, 0.5) *
11     (-1));
12 }
13 void compareWithExact(const vector<vector<double>>& result, double (*exactFunc)(double
14 )) {
15     const int width = 15;
16     const int precision = 6;
17
18     cout << "+-----+-----+-----+" << endl
19     ;
20     cout << "| x | y | Exact y | Error |" << endl;
21     cout << "+-----+-----+-----+" << endl
22     ;
23     for (size_t i = 0; i < result[0].size(); ++i) {
24         double exact_y = exactFunc(result[0][i]);
25         double error = abs(result[1][i] - exact_y);
26         cout << "| " << setw(width - 2) << setprecision(precision) << result[0][i] << "
27         | "
28         << setw(width - 2) << setprecision(precision) << result[1][i] << " | "
29         << setw(width - 2) << setprecision(precision) << exact_y << " | "
30         << setw(width - 2) << setprecision(precision) << error << " |" << endl;
31     }
32     cout << "+-----+-----+-----+" << endl
33     ;
34 }
35
36 int main() {
37     double x0 = 0, y1_0 = 1, y2_0 = 0, h = 0.1;

```

```

34 |     int n = 11;
35 |
36 |     cout << "Euler method:" << "\n";
37 |     auto eulerRes = eulerMethod(f, x0, y1_0, y2_0, h, n);
38 |     compareWithExact(eulerRes, fExact);
39 |
40 |     cout << "\nRunge-Kutta method:" << "\n";
41 |     auto rungeKuttaRes = rungeKutta4(f, x0, y1_0, y2_0, h, n);
42 |     compareWithExact(rungeKuttaRes, fExact);
43 |
44 |     cout << "\nAdams method:" << "\n";
45 |     auto adamsRes = adams4(f, x0, y1_0, y2_0, h, n);
46 |     compareWithExact(adamsRes, fExact);
47 |
48 |     cout << "\nEuler method error by Runge Romberg:" << "\n";
49 |     rungeRomberg(eulerMethod, f, x0, y1_0, y2_0, h, n);
50 |
51 |     cout << "\nRunge-Kutta method error by Runge Romberg:" << "\n";
52 |     rungeRomberg(rungeKutta4, f, x0, y1_0, y2_0, h, n);
53 |
54 |     cout << "\nAdams method error by Runge Romberg:" << "\n";
55 |     rungeRomberg(adams4, f, x0, y1_0, y2_0, h, n);
56 |
57 |     return 0;
58 | }

```

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

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) | $\|A(s)\|$ |
| 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 | 1.90281 | |
| 1.2 | 1.82466 | |
| 1.3 | 1.76086 | |
| 1.4 | 1.70818 | |
| 1.5 | 1.66429 | |
| 1.6 | 1.62748 | |
| 1.7 | 1.59645 | |
| 1.8 | 1.57022 | |
| 1.9 | 1.54801 | |
| 2 | 1.52922 | |
| | 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

```
1 #include <iostream>
2 #include "derivativeMethods.h"
3 using namespace std;
4
5 double f_m(double x, double y1, double y2) {
6     return (2 * y1) / (x * x * (x + 1));
7 }
8
9 double exactSolution(double x) {
10     return 1.0 / x + 1.0;
11 }
12
13 double rungeRombergError(const vector<double>& y_h, const vector<double>& y_h2, double
    r) {
14     double error = 0.0;
15     for (size_t i = 0; i < y_h.size(); ++i) {
16         error = max(error, abs(y_h2[2 * i] - y_h[i]) / (pow(2, r) - 1));
17     }
18     return error;
19 }
20
21 void rungeRombergErrorShooting(const vector<vector<double>>& res_h, const vector<
    vector<double>>& res_h2) {
22     const int width = 15;
23     const int precision = 6;
24
25     cout << "+-----+" << endl;
26     cout << "| x | Error y | Error y' |" << endl;
27     cout << "+-----+" << endl;
28     for (int i = 0; i < res_h[0].size(); ++i) {
29         double error_y1 = (res_h2[1][2 * i] - res_h[1][i]) / (pow(2, 4) - 1);
30         double error_y2 = (res_h2[2][2 * i] - res_h[2][i]) / (pow(2, 4) - 1);
31         cout << "| " << setw(width - 2) << setprecision(precision) << res_h[0][i] << "
            | "
32             << setw(width - 2) << setprecision(precision) << error_y1 << " | "
33             << setw(width - 2) << setprecision(precision) << error_y2 << " |" << endl;
34     }
35     cout << "+-----+" << endl;
36 }
37
38 int main() {
39     cout << "Real value:" << "\n";
40     printTable(1, 2, 0.2, exactSolution);
41
42     double a = 1, b = 2;
43     double alpha = 2, beta = 1.5;
44     double h = 0.2;
```

```

45     double eps = 0.0001;
46
47     cout << "\nShooting method:" << "\n";
48     shootingMethod(f_m, a, b, alpha, beta, eps, h, 6);
49     cout << "\nRunge-Romberg error estimate for Shooting Method:" << "\n";
50     rungeRomberg(rungeKutta4, f_m, a, alpha, 0, h, 6);
51
52     std::function<double(double)> p = [](double x) {
53         return 0;
54     };
55     std::function<double(double)> q = [](double x) {
56         return -2 / (x * x * (x + 1));
57     };
58     std::function<double(double)> f = [](double x) {
59         return 0.0;
60     };
61
62     double y0 = 2.0;
63     double y1 = 1.5;
64
65     vector<double> x_h, y_h, x_h2, y_h2;
66
67     cout << "\nFinite difference Method:" << "\n";
68     finiteDifferenceMethod(a, b, y0, y1, h, p, q, f, x_h, y_h);
69
70     cout << "\nFinite difference Method with h/2:" << "\n";
71     finiteDifferenceMethod(a, b, y0, y1, h / 2, p, q, f, x_h2, y_h2);
72
73     double error = rungeRombergError(y_h, y_h2, 2);
74
75     cout << "\nError Finite Difference Method:" << endl;
76     cout << "+-----+-----+" << endl;
77     cout << "| x | Error y |" << endl;
78     cout << "+-----+-----+" << endl;
79
80     double maxError = 0.0;
81     for (size_t i = 0; i < x_h.size(); ++i) {
82         double exact = exactSolution(x_h[i]);
83         double error = abs(y_h[i] - exact);
84         maxError = max(maxError, error);
85         cout << "| " << setw(13) << setprecision(6) << x_h[i] << " | "
86             << setw(13) << setprecision(6) << error << " |" << endl;
87     }
88     cout << "+-----+-----+" << endl;
89
90     return 0;
91 }

```