

**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: Ю. В. Кон
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: 12

$$\begin{aligned}(x^2 + 1)y'' - 2xy' + 2y &= 0, \\ y(0) &= 1, \\ y'(0) &= 1, \\ x \in [0, 1], h &= 0.1\end{aligned} \quad \left| \quad y = x - x^2 + 1\right.$$

Рис. 1: Входные данные

2 Результаты работы

```
Метод Эйлера:
x: 0, решение: 1, точное решение: 1, погрешность сравнением с точным решением: 0
x: 0.1, решение: 1.1, точное решение: 1.09, погрешность сравнением с точным решением: 0.01
x: 0.2, решение: 1.18, точное решение: 1.16, погрешность сравнением с точным решением: 0.02
x: 0.3, решение: 1.2398, точное решение: 1.21, погрешность сравнением с точным решением: 0.029802
x: 0.4, решение: 1.27921, точное решение: 1.24, погрешность сравнением с точным решением: 0.0392117
x: 0.5, решение: 1.29804, точное решение: 1.25, погрешность сравнением с точным решением: 0.0480422
x: 0.6, решение: 1.29612, точное решение: 1.24, погрешность сравнением с точным решением: 0.0561159
x: 0.7, решение: 1.27327, точное решение: 1.21, погрешность сравнением с точным решением: 0.0632668
x: 0.8, решение: 1.22934, точное решение: 1.16, погрешность сравнением с точным решением: 0.0693411
x: 0.9, решение: 1.1642, точное решение: 1.09, погрешность сравнением с точным решением: 0.0741973
x: 1, решение: 1.07771, точное решение: 1, погрешность сравнением с точным решением: 0.0777061
Погрешность методом Рунге-Ромберга-Ричардсона для метода Эйлера: 0.000399927

Метод Рунге-Кутты:
x: 0, решение: 1, точное решение: 1, погрешность сравнением с точным решением: 0
x: 0.1, решение: 1.09, точное решение: 1.09, погрешность сравнением с точным решением: 4.14591e-008
x: 0.2, решение: 1.16, точное решение: 1.16, погрешность сравнением с точным решением: 4.17591e-009
x: 0.3, решение: 1.21, точное решение: 1.21, погрешность сравнением с точным решением: 1.40984e-007
x: 0.4, решение: 1.24, точное решение: 1.24, погрешность сравнением с точным решением: 3.72462e-007
x: 0.5, решение: 1.25, точное решение: 1.25, погрешность сравнением с точным решением: 7.0078e-007
x: 0.6, решение: 1.24, точное решение: 1.24, погрешность сравнением с точным решением: 1.12655e-006
x: 0.7, решение: 1.21, точное решение: 1.21, погрешность сравнением с точным решением: 1.64904e-006
x: 0.8, решение: 1.16, точное решение: 1.16, погрешность сравнением с точным решением: 2.26652e-006
x: 0.9, решение: 1.09, точное решение: 1.09, погрешность сравнением с точным решением: 2.97671e-006
x: 1, решение: 1, точное решение: 1, погрешность сравнением с точным решением: 3.77712e-006
Погрешность методом Рунге-Ромберга-Ричардсона для метода Рунге-Кутты: 0.00316643

Метод Адамса:
x: 0, решение: 1, точное решение: 1, погрешность сравнением с точным решением: 0
x: 0.1, решение: 1.09, точное решение: 1.09, погрешность сравнением с точным решением: 4.14591e-008
x: 0.2, решение: 1.16, точное решение: 1.16, погрешность сравнением с точным решением: 4.17591e-009
x: 0.3, решение: 1.21, точное решение: 1.21, погрешность сравнением с точным решением: 1.40984e-007
x: 0.4, решение: 1.24, точное решение: 1.24, погрешность сравнением с точным решением: 3.72462e-007
x: 0.5, решение: 1.25, точное решение: 1.25, погрешность сравнением с точным решением: 9.95616e-007
x: 0.6, решение: 1.24, точное решение: 1.24, погрешность сравнением с точным решением: 1.56541e-006
x: 0.7, решение: 1.20185, точное решение: 1.21, погрешность сравнением с точным решением: 0.00814706
x: 0.8, решение: 1.15213, точное решение: 1.16, погрешность сравнением с точным решением: 0.00786829
x: 0.9, решение: 1.07347, точное решение: 1.09, погрешность сравнением с точным решением: 0.0165342
x: 1, решение: 0.975426, точное решение: 1, погрешность сравнением с точным решением: 0.0245741
Погрешность методом Рунге-Ромберга-Ричардсона для метода Адамса: 0.00319045
```

Рис. 2: Вывод программы в консоли

3 Исходный код

Файл с первым заданием четвертой лабораторной работы:

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <fstream>
5
6 using namespace std;
7
8 double exact_solution(double x){
9     return x - x*x + 1;
10 }
11
12 void euler(double h, double x_0, double y1_0, double y2_0, double end, vector<double>&
    x_val, vector<double>& y1_val, vector<double>& y2_val) {
13     double x = x_0;
14     double y1 = y1_0;
15     double y2 = y2_0;
16     while (x <= end) {
17         x_val.push_back(x);
18         y1_val.push_back(y1);
19         y2_val.push_back(y2);
20         double y1_new = y1 + h * y2;
21         double y2_new = y2 + h * ((2 * x * y2 - 2 * y1)/(x * x + 1));
22         y1 = y1_new;
23         y2 = y2_new;
24         x += h;
25     }
26 }
27
28 void runge_kutta(double h, double x_0, double y1_0, double y2_0, double end, vector<
    double>& x_val, vector<double>& y1_val, vector<double>& y2_val) {
29     double x = x_0;
30     double y1 = y1_0;
31     double y2 = y2_0;
32     while (x <= end) {
33         x_val.push_back(x);
34         y1_val.push_back(y1);
35         y2_val.push_back(y2);
36         double k1_1 = h * y2;
37         double k1_2 = h * ((2 * x * y2 - 2 * y1)/(x * x + 1));
38
39         double k2_1 = h * (y2 + 0.5 * k1_2);
40         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));
41
42         double k3_1 = h * (y2 + 0.5 * k2_2);
```

```

43     double k3_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k2_2) - 2 * (y1 + 0.5 *
44         k2_1)) / ((x + 0.5 * h) * (x + 0.5 * h) + 1));
45     double k4_1 = h * (y2 + k3_2);
46     double k4_2 = h * ((2 * (x + h) * (y2 + k3_2) - 2 * (y1 + k3_1)) / ((x + h) * (
47         x + h) + 1));
48     y1 += (k1_1 + 2 * k2_1 + 2 * k3_1 + k4_1) / 6;
49     y2 += (k1_2 + 2 * k2_2 + 2 * k3_2 + k4_2) / 6;
50     x += h;
51 }
52 }
53
54 void adam(double h, double x_0, double y1_0, double y2_0, double end, vector<double>&
55     x_val, vector<double>& y1_val, vector<double>& y2_val) {
56     double x = x_0;
57     double y1 = y1_0;
58     double y2 = y2_0;
59     vector<double> f_1, f_2;
60     for (int i = 0; i < 4; i++) {
61         x_val.push_back(x);
62         y1_val.push_back(y1);
63         y2_val.push_back(y2);
64
65         double k1_1 = h * y2;
66         double k1_2 = h * ((2 * x * y2 - 2 * y1) / (x * x + 1));
67
68         double k2_1 = h * (y2 + 0.5 * k1_2);
69         double k2_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k1_2) - 2 * (y1 + 0.5 *
70             k1_1)) / ((x + 0.5 * h) * (x + 0.5 * h) + 1));
71
72         double k3_1 = h * (y2 + 0.5 * k2_2);
73         double k3_2 = h * ((2 * (x + 0.5 * h) * (y2 + 0.5 * k2_2) - 2 * (y1 + 0.5 *
74             k2_1)) / ((x + 0.5 * h) * (x + 0.5 * h) + 1));
75
76         double k4_1 = h * (y2 + k3_2);
77         double k4_2 = h * ((2 * (x + h) * (y2 + k3_2) - 2 * (y1 + k3_1)) / ((x + h) * (
78             x + h) + 1));
79
80         y1 += (k1_1 + 2 * k2_1 + 2 * k3_1 + k4_1) / 6;
81         y2 += (k1_2 + 2 * k2_2 + 2 * k3_2 + k4_2) / 6;
82         x += h;
83
84         f_1.push_back(y2);
85         f_2.push_back((2 * x * y2 - 2 * y1) / (x * x + 1));
86     }
87 }
88
89 while (x <= end) {
90     x_val.push_back(x);

```

```

86     y1_val.push_back(y1);
87     y2_val.push_back(y2);
88     double y1_new = y1 + h / 24 * (55 * f_1.back() - 59 * f_1[f_1.size() - 2] + 37
89         * f_1[f_1.size() - 3] - 9 * f_1[f_1.size() - 4]);
90     double y2_new = y2 + h / 24 * (55 * f_2.back() - 59 * f_2[f_2.size() - 2] + 37
91         * f_2[f_2.size() - 3] - 9 * f_2[f_2.size() - 4]);
92     f_1.push_back(y2_new);
93     f_2.push_back((2 * x * y2_new - 2 * y1_new) / (x * x + 1));
94     y1 = y1_new;
95     y2 = y2_new;
96     x += h;
97 }
98 }
99
100 int main(){
101     double x_0 = 0.0;
102     double y1_0 = 1;
103     double y2_0 = 1;
104     double end = 1.0;
105     double h = 0.1;
106     vector<double> x_val, y1_val, y2_val;
107     vector<double> x_val_half, y1_val_half, y2_val_half;
108     euler(h, x_0, y1_0, y2_0, end, x_val, y1_val, y2_val);
109     ofstream fout("output.txt");
110     fout << " : " << endl;
111     for (size_t i = 0; i < x_val.size(); i++) {
112         double y_exact = exact_solution(x_val[i]);
113         fout << "x: " << x_val[i] << ", : " << y1_val[i] << ", : " << y_exact << ",
114             : " << fabs(y1_val[i] - y_exact) << endl;
115     }
116     fout << endl;
117     euler(h / 2, x_0, y1_0, y2_0, end, x_val_half, y1_val_half, y2_val_half);
118     double error = fabs(y1_val_half[y1_val_half.size() - 1] - y1_val[y1_val.size() -
119         1]) / (pow(2, 4) - 1);
120     fout << " -- : " << error << endl;
121     fout << endl;
122     x_val.clear();
123     y1_val.clear();
124     y2_val.clear();
125     x_val_half.clear();
126     y1_val_half.clear();
127     y2_val_half.clear();
128     runge_kutta(h, x_0, y1_0, y2_0, end, x_val, y1_val, y2_val);
129     fout << " -: " << endl;
130     for (size_t i = 0; i < x_val.size(); i++) {
131         double y_exact = exact_solution(x_val[i]);
132         fout << "x: " << x_val[i] << ", : " << y1_val[i] << ", : " << y_exact << ",
133             : " << fabs(y1_val[i] - y_exact) << endl;
134     }
135 }

```

```

130     fout << endl;
131     runge_kutta(h / 2, x_0, y1_0, y2_0, end, x_val_half, y1_val_half, y2_val_half);
132     error = fabs(y1_val_half[y1_val_half.size() - 1] - y1_val[y1_val.size() - 1]) / (
        pow(2, 4) - 1);
133     fout << "  --  -: " << error << endl;
134     fout << endl;
135     x_val.clear();
136     y1_val.clear();
137     y2_val.clear();
138     x_val_half.clear();
139     y1_val_half.clear();
140     y2_val_half.clear();
141     adam(h, x_0, y1_0, y2_0, end, x_val, y1_val, y2_val);
142     fout << " : " << endl;
143     for (size_t i = 0; i < x_val.size(); i++) {
144         double y_exact = exact_solution(x_val[i]);
145         fout << "x: " << x_val[i] << ", : " << y1_val[i] << ", : " << y_exact << ",
            : " << fabs(y1_val[i] - y_exact) << endl;
146     }
147     fout << endl;
148     adam(h / 2, x_0, y1_0, y2_0, end, x_val_half, y1_val_half, y2_val_half);
149     error = fabs(y1_val_half[y1_val_half.size() - 1] - y1_val[y1_val.size() - 1]) / (
        pow(2, 4) - 1);
150     fout << "  --  : " << error << endl;
151     return 0;
152 }

```

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: 12

$x(x-1)y'' - xy' + y = 0$ $y'(1) = 3$ $y(3) - 3y'(3) = -4$	$y(x) = 2 + x + 2x \ln x $
--	----------------------------

Рис. 3: Входные данные

5 Результаты работы

```
Точное решение:
3.0003 3.63791 4.34249 5.10441 5.91645 6.77303 7.66967 8.60273 9.56915 10.5664 11.5922

Метод стрельбы:
3.0003 3.63169 4.33043 5.08802 5.89747 6.75321 7.6507 8.5862 9.55657 10.5592 11.5917
Метод difference:
3.0003 4.22716 4.84554 5.46875 6.09883 6.73901 7.39475 8.07546 8.79801 9.5938 10.8235

Погрешности методом Рунге-Ромберга-Ричардсона
Метод стрельбы: 7.0564e-012
Метод difference: 0.0316271

Погрешности сравнением с точным решением
Метод стрельбы:
0 0.00622174 0.0120579 0.0163902 0.0189788 0.019817 0.0189709 0.0165285 0.0125815 0.00721792 0.000519726
Метод difference:
0 0.589255 0.503046 0.364348 0.182377 0.0340174 0.274921 0.527265 0.771136 0.972574 0.7687
```

Рис. 4: Вывод программы в консоли

6 Исходный код

Файл со вторым заданием четвертой лабораторной работы:

```
1  #include <iostream>
2  #include <cmath>
3  #include <vector>
4  #include <fstream>
5
6  using namespace std;
7
8  double f(double x, double y, double z){
9      return (x*z-y)/(x*x-x);
10 }
11
12 double exact_solution(double x){
13     return 2+x+2*x*log(fabs(x));
14 }
15
16 double p(double x){
17     return -x;
18 }
19
20 double q(double x){
21     return 1;
22 }
23
24 vector<double> runge_kutta(double a, double b, double h, vector<double> x, int n,
25     double y0, double z0) {
26     vector<double> y(n);
27     vector<double> z(n);
28
29     vector<double> K(4);
30     vector<double> L(4);
31
32     y[0] = y0;
33     z[0] = z0;
34
35     for (int i = 1; i < n; ++i) {
36
37         K[0] = h * z[i - 1];
38         L[0] = h * f(x[i - 1], y[i - 1], z[i-1]);
39
40         for (int j = 1; j < 4; ++j) {
41             K[j] = h * (z[i - 1] + L[j - 1] / 2);
42             L[j] = h * f(x[i - 1] + h / 2, y[i - 1] + K[j - 1] / 2, z[i - 1] + L[j - 1]
43                 / 2);
44         }
45
46         double dy = (K[0] + 2 * K[1] + 2 * K[2] + K[3]) / 6;
```



```

45     double dz = (L[0] + 2 * L[1] + 2 * L[2] + L[3]) / 6;
46     y[i] = y[i - 1] + dy;
47     z[i] = z[i - 1] + dz;
48 }
49
50 return y;
51 }
52
53 vector<double> shooting_method(double a, double b, double h, vector<double> x, int n,
54     double eps, double y0, double y1){
55     double eta0 = 1;
56     double eta = 0.8;
57
58     double F0 = runge_kutta(a, b, h, x, n, y0, eta0)[n-1] - y1;
59     double F = runge_kutta(a, b, h, x, n, y0, eta)[n-1] - y1;
60
61     while(abs(F) > eps){
62         double c = eta;
63         eta = eta - F*(eta - eta0)/(F - F0);
64         eta0 = c;
65         F0 = F;
66         F = runge_kutta(a, b, h, x, n, y0, eta)[n - 1] - y1;
67     }
68     return runge_kutta(a, b, h, x, n, y0, eta);
69 }
70 vector<double> difference_method(double a, double b, double h, vector<double> x, int n
71     , double y0, double y1) {
72     vector<double> A, B, C, D, P(n), Q(n), res(n);
73
74     A.push_back(0);
75     B.push_back(-2 + h * h * q(x[1]));
76     C.push_back(1 + p(x[1]) * h / 2);
77     D.push_back(-(1 - (p(x[1]) * h) / 2) * y0);
78     for (int i = 2; i < n; ++i) {
79         A.push_back(1 - p(x[i]) * h / 2);
80         B.push_back(-2 + h * h * q(x[i]));
81         C.push_back(1 + p(x[i]) * h / 2);
82         D.push_back(0);
83     }
84     A.push_back(1 - p(x[n - 2]) * h / 2);
85     B.push_back(-2 + h * h * q(x[n - 2]));
86     C.push_back(0);
87     D.push_back(-(1 + (p(x[n - 2]) * h) / 2) * y1);
88
89     P[0] = (-C[0] / B[0]);
90     Q[0] = (D[0] / B[0]);
91     for (int i = 1; i <= n; ++i) {

```

```

92     P[i] = (-C[i] / (B[i] + A[i] * P[i - 1]));
93     Q[i] = ((D[i] - A[i] * Q[i - 1]) / (B[i] + A[i] * P[i - 1]));
94 }
95
96 res[n-1] = Q[n-1];
97 for (int i = n - 2; i > 0; --i)
98     res[i] = P[i] * res[i + 1] + Q[i];
99 res[0] = y0;
100 res[n] = y1;
101 return res;
102 }
103
104 int main(){
105     double a = 1.0001;
106     double b = 3;
107
108     double y0 = 2+a+2*a*log(a);
109     double y1 = 2+b+2*b*log(b);
110     double h = 0.2;
111     double eps = 0.0001;
112
113     ofstream fout("output.txt");
114
115     int n = 11;
116
117     vector<double> x(n), y(n);
118     for (int i = 0; i < n; ++i){
119         x[i] = h * i + a;
120         y[i] = exact_solution(x[i]);
121     }
122
123     vector<double> shoot = shooting_method(a, b, h, x, n, eps, y0, y1);
124
125     fout << endl << " :" << endl;
126     for (int i = 0; i < n; ++i){
127         fout << y[i] << " ";
128     }
129     fout << endl;
130     fout << endl << " :" << endl;
131     for (int i = 0; i < n; ++i){
132         fout << shoot[i] << " ";
133     }
134     vector<double> diff = difference_method(a, b, h, x, n, y0, y1);
135
136     fout << endl << " difference:" << endl;
137     for (int i = 0; i < n; ++i){
138         fout << diff[i] << " ";
139     }
140     fout << endl;

```

```

141
142     fout << endl << "  --" << endl;
143     vector<double> shoot_half = shooting_method(a, b, h/2, x, n, eps, y0, y1);
144     double error = fabs(shoot_half[shoot_half.size() - 1] - shoot[shoot.size() - 1]) /
        (pow(2, 4) - 1);
145     fout << " : " << error << endl;
146
147     vector<double> diff_half = difference_method(a, b, h/2, x, n, y0, y1);
148     error = fabs(diff_half[diff_half.size() - 1] - diff[diff.size() - 1]) / (pow(2, 4)
        - 1);
149
150     fout << " difference: " << error << endl;
151     fout << endl;
152
153     fout << "      " << endl;
154     fout << " : " << endl;
155     for (int i = 0; i < n; ++i){
156         fout << abs(shoot[i] - y[i]) << " ";
157     }
158     fout << endl << " difference: " << endl;
159     for (int i = 0; i < n; ++i){
160         fout << abs(diff[i] - y[i]) << " ";
161     }
162     return 0;
163 }

```