

**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: Голошумов М.С.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

4.1 Методы Эйлера, Рунге-Кутты и Адамса

1 Постановка задачи

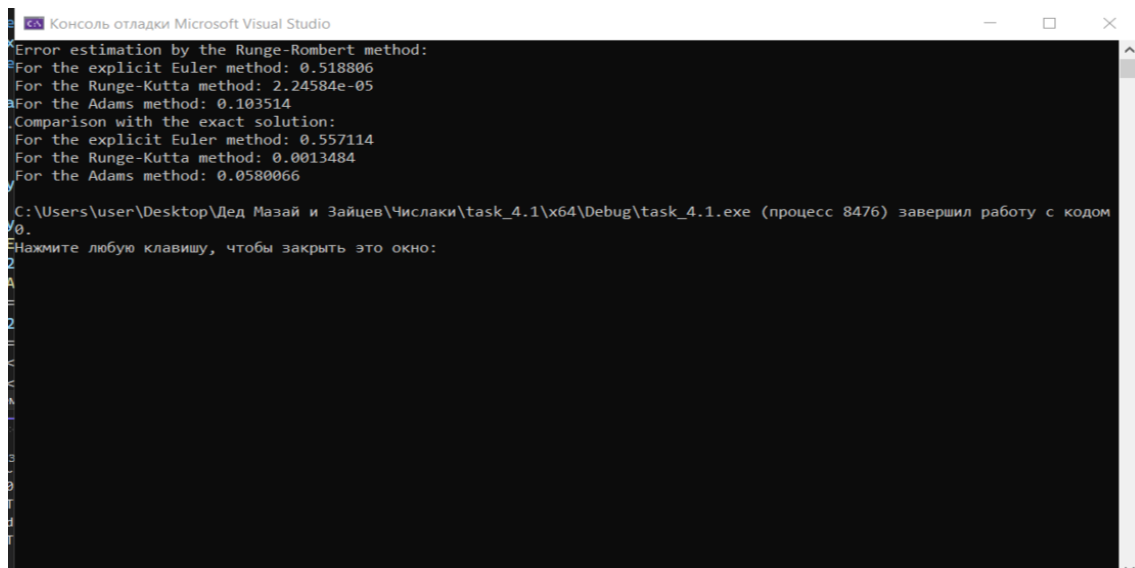
Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода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

Вариант: 4

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

Рис. 1: Входные данные

2 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Error estimation by the Runge-Rombert method:
For the explicit Euler method: 0.518806
For the Runge-Kutta method: 2.24584e-05
For the Adams method: 0.103514
Comparison with the exact solution:
For the explicit Euler method: 0.557114
For the Runge-Kutta method: 0.0013484
For the Adams method: 0.0580066
C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\task_4.1\x64\Debug\task_4.1.exe (процесс 8476) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 2: Вывод программы в консоли

3 Исходный код

```
1 | #include <cmath>
2 | #include <vector>
3 | #include <iostream>
4 | #include <utility>
5 | #include <functional>
6 |
7 | using namespace std;
8 | using vect = vector<double>;
9 |
10 |
11 | vect f(vect& x) {
12 |     vect res;
13 |     for (double val : x)
14 |         res.push_back((1 + exp(val * val / 2)) / val);
15 |     return res;
16 | }
17 |
18 |
19 | double ddf(double x, double f, double df) {
20 |     return (x * (x * x - 1) * df + (x * x + 1) * f) / (x * x);
21 | }
22 |
23 |
24 | double sqr_error(const vect& a, const vect& b) {
25 |     double res = 0;
26 |     for (int i = 0; i < a.size(); i++)
27 |         res += pow(a[i] - b[i], 2);
28 |     return sqrt(res);
29 | }
30 |
31 | vect Euler(function<double(double, double, double)> ddy, pair<double, double>& borders
32 |     , double y0, double z0, double h) {
33 |     double x = borders.first;
34 |     int N = (borders.second - borders.first) / h + 1;
35 |     vect y(N), z(N);
36 |
37 |     y[0] = y0;
38 |     z[0] = z0;
39 |
40 |     for (int i = 0; i < N - 1; i++) {
41 |         y[i + 1] = y[i] + h * z[i];
42 |         z[i + 1] = z[i] + h * ddy(x, y[i], z[i]);
43 |         x += h;
44 |     }
45 |     return y;
46 | }
```

```

47
48
49 pair<vect, vect> RungeKutta(function<double(double, double, double)> ddy, pair<double,
    double>& borders, double y0, double z0, double h) {
50     double x = borders.first;
51     int N = (borders.second - borders.first) / h + 1;
52     vect y(N), z(N);
53
54     y[0] = y0;
55     z[0] = z0;
56
57     for (int i = 0; i < N - 1; i++) {
58         double K1 = h * ddy(x, y[i], z[i]);
59         double L1 = h * ddy(x, y[i], z[i]);
60         double K2 = h * (z[i] + 0.5 * L1);
61         double L2 = h * ddy(x + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1);
62         double K3 = h * (z[i] + 0.5 * L2);
63         double L3 = h * ddy(x + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2);
64         double K4 = h * (z[i] + L3);
65         double L4 = h * ddy(x + h, y[i] + K3, z[i] + L3);
66         double delta_y = (K1 + 2 * K2 + 2 * K3 + K4) / 6;
67         double delta_z = (L1 + 2 * L2 + 2 * L3 + L4) / 6;
68         y[i + 1] = y[i] + delta_y;
69         z[i + 1] = z[i] + delta_z;
70         x += h;
71     }
72
73     return { y, z };
74 }
75
76
77 vect Adams(function<double(double, double, double)> ddy, pair<double, double>& borders
    , vect calc_y, vect& calc_z, double h) {
78     double x = borders.first;
79     int N = (borders.second - borders.first) / h + 1;
80     vect y, z;
81
82     for (int i = 0; i < 4; i++) {
83         y.push_back(calc_y[i]);
84         z.push_back(calc_z[i]);
85     }
86
87     for (int i = 3; i < N - 1; i++) {
88         double z_next = z[i] + (h / 24) * (55 * ddy(x, y[i], z[i]) - 59 * ddy(x - h, y[
            i - 1], z[i - 1]) + 37 * ddy(x - 2 * h, y[i - 2], z[i - 2]) - 9 * ddy(x - 3
            * h, y[i - 3], z[i - 3]));
89         double y_next = y[i] + (h / 24) * (55 * z[i] - 59 * z[i - 1] + 37 * z[i - 2] -
            9 * z[i - 3]);

```

```

90     double z_i = z[i] + (h / 24) * (9 * ddy(x + h, y_next, z_next) + 19 * ddy(x, y[
        i], z[i]) - 5 * ddy(x - h, y[i - 1], z[i - 1]) + 1 * ddy(x - 2 * h, y[i -
        2], z[i - 2]));
91     z.push_back(z_i);
92     double y_i = y[i] + (h / 24) * (9 * z_next + 19 * z[i] - 5 * z[i - 1] + 1 * z[i
        - 2]);
93     y.push_back(y_i);
94     x += h;
95 }
96
97 return y;
98 }
99
100
101 vect RungeRomberg(vect y1, vect y2, double h1, double h2, double p) {
102     if (h1 > h2) {
103         int k = h1 / h2;
104         vect y(y1.size());
105         for (int i = 0; i < y1.size(); i++)
106             y[i] = y2[i * k] + (y2[i * k] - y1[i]) / (pow(k, p) - 1);
107         return y;
108     }
109     else {
110         int k = h2 / h1;
111         vect y(y2.size());
112         for (int i = 0; i < y2.size(); i++)
113             y[i] = y1[i * k] + (y1[i * k] - y2[i]) / (pow(k, p) - 1);
114         return y;
115     }
116 }
117
118
119 int main() {
120     pair<double, double> borders = { 1, 2 };
121     double y0 = 2.649, z0 = -1.0, h = 0.1;
122     double h2 = h / 2;
123     vect x = { borders.first };
124     double last_el = borders.first;
125     while (last_el < borders.second) {
126         last_el += h;
127         x.push_back(last_el);
128     }
129
130     vect y = f(x);
131
132     vect y1, y2, y3, z, z2, y1_2, y2_2, y3_2;
133     y1 = Euler(ddf, borders, y0, z0, h);
134     tie(y2, z) = RungeKutta(ddf, borders, y0, z0, h);
135     y3 = Adams(ddf, borders, y2, z, h);

```

```

136 | y1_2 = Euler(ddf, borders, y0, z0, h2);
137 | tie(y2_2, z2) = RungeKutta(ddf, borders, y0, z0, h2);
138 | y3_2 = Adams(ddf, borders, y2_2, z2, h2);
139 | cout << "Error estimation by the Runge-Romberg method:" << endl;
140 | cout << "For the explicit Euler method: " << sqr_error(y1, RungeRomberg(y1, y1_2, h
    | , h2, 1)) << endl;
141 | cout << "For the Runge-Kutta method: " << sqr_error(y2, RungeRomberg(y2, y2_2, h,
    | h2, 4)) << endl;
142 | cout << "For the Adams method: " << sqr_error(y3, RungeRomberg(y3, y3_2, h, h2, 4))
    | << endl;
143 | cout << "Comparison with the exact solution:" << endl;
144 | cout << "For the explicit Euler method: " << sqr_error(y1, y) << endl;
145 | cout << "For the Runge-Kutta method: " << sqr_error(y2, y) << endl;
146 | cout << "For the Adams method: " << sqr_error(y3, y) << endl;
147 | }

```

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: 4

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

Рис. 3: Входные данные

5 Результаты работы

```
Консоль отладки Microsoft Visual Studio
Shooting method:
x: 1.000000000 y: -2.295632162 exact y: 3.772588722
x: 1.100000000 y: -1.928422811 exact y: 3.651033498
x: 1.200000000 y: -1.576601124 exact y: 3.557676988
x: 1.300000000 y: -1.234866305 exact y: 3.485678435
x: 1.400000000 y: -0.899573432 exact y: 3.430178528
x: 1.500000000 y: -0.568165019 exact y: 3.387635773
x: 1.600000000 y: -0.238818196 exact y: 3.355412196
x: 1.700000000 y: 0.089781703 exact y: 3.331505632
x: 1.800000000 y: 0.41850629 exact y: 3.314371520
x: 1.900000000 y: 0.748306984 exact y: 3.302801197
x: 2.000000000 y: 1.079441542 exact y: 3.295836866
Runge-Rumberg error:
0.00000790

Finite-difference method:
x: 1.000000000 y: 3.772588722 exact y: 3.772588722
x: 1.100000000 y: 3.820589488 exact y: 3.651033498
x: 1.200000000 y: 3.794312571 exact y: 3.557676988
x: 1.300000000 y: 3.673193111 exact y: 3.485678435
x: 1.400000000 y: 3.409259815 exact y: 3.430178528
x: 1.500000000 y: 2.833808648 exact y: 3.387635773
x: 1.600000000 y: -0.685212066 exact y: 3.355412196
x: 1.700000000 y: 0.243357186 exact y: 3.331505632
x: 1.800000000 y: 0.652332049 exact y: 3.314371520
x: 1.900000000 y: 0.906203610 exact y: 3.302801197
x: 2.000000000 y: 1.079441542 exact y: 3.295836866
Runge-Rumberg error:
0.889422893
```

Рис. 4: Вывод программы в консоли

6 Исходный код

```
1 | #include <cmath>
2 | #include <iostream>
3 | #include <vector>
4 | #include <tuple>
5 | #include <iostream>
6 | #include <vector>
7 | #include <functional>
8 | #include <fstream>
9 | #include <iomanip>
10 |
11 | using namespace std;
12 | using tddd = tuple<double, double, double>;
13 | using func = function<double(double, double, double)>;
14 | using vect = vector<tddd>;
15 | using vec = vector<double>;
16 |
17 | const double PI = acos(-1.0);
18 | const double EPS = 1e-9;
19 |
20 | double exact_y(double x) {
21 |     return -1 + 2/x + (2*(x+1)/x)*log(abs(x+1));
22 | }
23 |
24 | double g(double x, double y, double dy) {
25 |     return 2 * y/(x*x*(x+1));
26 | }
27 |
28 | double f(double x, double y, double dy) {
29 |     (void)x;
30 |     (void)y;
31 |     return dy;
32 | }
33 |
34 | double px(double x) {
35 |     return -tan(x);
36 | }
37 |
38 | double qx(double x) {
39 |     return 2.0;
40 | }
41 |
42 | double fx(double x) {
43 |     (void)x;
44 |     return 0.0;
45 | }
46 |
47 | bool leq(double a, double b) {
```



```

48     return (a < b) || (abs(b - a) < EPS);
49 }
50
51 template <class T>
52 class Trid {
53 private:
54     const T EPS = 1e-6;
55
56     int n;
57     vector<T> a, b, c;
58 public:
59     Trid(const int& size) : n(size), a(n), b(n), c(n) {}
60     Trid(vector<double>& _a, vector<double>& _b, vector<double>& _c) : n(_a.size()), a(
        _a), b(_b), c(_c) {};
61
62     vector<T> Solve(const vector<T>& d) {
63         vector<T> p(n);
64         p[0] = -c[0] / b[0];
65         vector<T> q(n);
66         q[0] = d[0] / b[0];
67         for (int i = 1; i < n; ++i) {
68             p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
69             q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
70         }
71         vector<T> x(n);
72         x.back() = q.back();
73         for (int i = n - 2; i >= 0; --i) {
74             x[i] = p[i] * x[i + 1] + q[i];
75         }
76         return x;
77     }
78
79     friend istream& operator >> (istream& in, Trid<T>& t) {
80         in >> t.b[0] >> t.c[0];
81         for (int i = 1; i < t.n - 1; ++i) {
82             in >> t.a[i] >> t.b[i] >> t.c[i];
83         }
84         in >> t.a.back() >> t.b.back();
85         return in;
86     }
87
88     ~Trid() = default;
89 };
90
91
92 vect runge_solve(double l, double r, double y0, double z0, double h) {
93     vect res;
94     double xk = l;
95     double yk = y0;

```

```

96     double zk = z0;
97     res.push_back(make_tuple(xk, yk, zk));
98     while (leq(xk + h, r)) {
99         double K1 = h * f(xk, yk, zk);
100        double L1 = h * g(xk, yk, zk);
101        double K2 = h * f(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
102        double L2 = h * g(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
103        double K3 = h * f(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
104        double L3 = h * g(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
105        double K4 = h * f(xk + h, yk + K3, zk + L3);
106        double L4 = h * g(xk + h, yk + K3, zk + L3);
107        double dy = (K1 + 2.0 * K2 + 2.0 * K3 + K4) / 6.0;
108        double dz = (L1 + 2.0 * L2 + 2.0 * L3 + L4) / 6.0;
109        xk += h;
110        yk += dy;
111        zk += dz;
112        res.push_back(make_tuple(xk, yk, zk));
113    }
114    return res;
115 }
116
117 double runge_romberg_max(const vect& y_2h, const vect& y_h, double p) {
118     double coef = 1.0 / (pow(2, p) - 1.0);
119     double res = 0.0;
120     for (size_t i = 0; i < y_2h.size(); ++i) {
121         res = max(res, coef * abs(get<1>(y_2h[i]) - get<1>(y_h[2 * i])));
122     }
123     return res;
124 }
125
126 double runge_romberg(double y1, double y2, int64_t p) {
127     return abs((y1 - y2) / (pow(2, p) - 1));
128 }
129
130 class fin_dif {
131 private:
132
133     double a, b;
134     func p, q, f;
135     double alpha, beta, y0;
136     double delta, gamma, y1;
137
138 public:
139     fin_dif(const double _a, const double _b,
140             const func _p, const func _q, const func _f,
141             const double _alpha, const double _beta, const double _y0,
142             const double _delta, const double _gamma, const double _y1)
143         : a(_a), b(_b), p(_p), q(_q), f(_f),
144         alpha(_alpha), beta(_beta), y0(_y0),

```

```

145     delta(_delta), gamma(_gamma), y1(_y1) {}
146
147
148 };
149
150 vect shooting_solve(double a, double b, double alpha, double beta, double y0, double
    delta, double gamma, double y1, double h, double eps) {
151     double eta_prev = 0.9;
152     double eta = 0.8;
153     while (1) {
154         vect sol_prev = runge_solve(a, b, eta_prev, y0, h),
155             sol = runge_solve(a, b, eta, y0, h);
156
157         double yb_prev = get<1>(sol_prev.back());
158         double zb_prev = get<2>(sol_prev.back());
159         double phi_prev = delta * yb_prev + gamma * zb_prev - y1;
160         double yb = get<1>(sol.back());
161         double zb = get<2>(sol.back());
162         double phi = delta * yb + gamma * zb - y1;
163         double eta_next = eta - (eta - eta_prev) / (phi - phi_prev) * phi;
164         if (abs(eta_next - eta) < eps) {
165             return sol;
166         }
167         else {
168             eta_prev = eta;
169             eta = eta_next;
170         }
171     }
172 }
173
174 using tridiag = Trid<double>;
175 vect fin_dif_solve(double a, double b,
176     double alpha, double beta, double y0,
177     double delta, double gamma, double y1, double h) {
178     size_t n = (b - a) / h;
179     vec xk(n + 1);
180     for (size_t i = 0; i <= n; ++i) {
181         xk[i] = a + h * i;
182     }
183     vec A(n + 1);
184     vec B(n + 1);
185     vec C(n + 1);
186     vec D(n + 1);
187     B[0] = (alpha - beta / h);
188     C[0] = beta / h;
189     D[0] = y0;
190     A.back() = -gamma / h;
191     B.back() = delta + gamma / h;
192     D.back() = y1;

```

```

193     for (size_t i = 1; i < n; ++i) {
194         A[i] = 1.0 - px(xk[i]) * h * 0.5;
195         B[i] = -2.0 + h * h * qx(xk[i]);
196         C[i] = 1.0 + px(xk[i]) * h * 0.5;
197         D[i] = h * h * fx(xk[i]);
198     }
199     tridiag sys_eq(A, B, C);
200     vec yk = sys_eq.Solve(D);
201     vect res;
202     for (size_t i = 0; i <= n; ++i) {
203         res.push_back(make_tuple(xk[i], yk[i], NAN));
204     }
205     return res;
206 }
207
208
209
210 int main() {
211     cout.precision(6);
212     cout << fixed;
213     double h = 0.1, eps = 0.001;
214
215     double a = 1, b = 2;
216     double alpha = 1, beta = 0, y0 = 1 + 4*log(2);
217     double delta = 1, gamma = 0, y1 = -1 + 3 * log(2);
218     cout << "Shooting method:" << endl;
219     vector<tddd> sol_shooting_h1 = shooting_solve(a, b, alpha, beta, y0, delta, gamma,
220         y1, h, eps),
221         sol_shooting_h2 = shooting_solve(a, b, alpha, beta, y0, delta, gamma, y1, h /
222             2, eps);
223     for (int i = 0; i < sol_shooting_h1.size(); ++i) {
224         double ex_y = exact_y(get<0>(sol_shooting_h1[i]));
225         cout << fixed << setprecision(9) << "x: " << get<0>(sol_shooting_h1[i]) << " y:
226             " << get<1>(sol_shooting_h1[i]) << " exact y: " << ex_y << endl;
227     }
228     cout << "Runge-Rumberg error:" << endl;
229     double shooting_err = runge_romberg_max(sol_shooting_h1, sol_shooting_h2, 4);
230     cout << shooting_err << endl << endl;
231
232     vector<tddd> sol_fin_dif_h1 = fin_dif_solve(a, b, alpha, beta, y0, delta, gamma, y1
233         , h),
234         sol_fin_dif_h2 = fin_dif_solve(a, b, alpha, beta, y0, delta, gamma, y1, h / 2);
235     cout << "Finite-difference method:" << endl;
236     for (int i = 0; i < sol_fin_dif_h1.size(); ++i) {
237         double ex_y = exact_y(get<0>(sol_fin_dif_h1[i]));
238         cout << fixed << setprecision(9) << "x: " << get<0>(sol_fin_dif_h1[i]) << " y:
239             " << get<1>(sol_fin_dif_h1[i]) << " exact y: " << ex_y << endl;
240     }
241     cout << "Runge-Rumberg error:" << endl;

```

```
237 || double fin_dif_err = runge_romberg_max(sol_fin_dif_h1, sol_fin_dif_h2, 2);  
238 || cout << fin_dif_err << endl;  
239 || }
```