

**Московский авиационный институт
(национальный исследовательский университет)**

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 4 по курсу «Численные методы»

Студент: Н. О. Тимофеева
Преподаватель: Д. Е. Пивоваров
Группа: М8О-308Б-19
Вариант: 19
Дата:
Оценка:

Москва, 2022

Лабораторная работа 4

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

4.1

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.

Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 | #include <functional>
5 | #include <tuple>
6 |
7 | using namespace std;
8 |
9 | const double EPS = 0.000000001;
10 | void printData(const vector<tuple<double, double, double>> & v) {
11 |     cout << "x = [";
12 |     for (int i = 0; i < v.size(); ++i) {
13 |         if (i) {
14 |             cout << ", ";
15 |         }
16 |         cout << get<0>(v[i]);
17 |     }
18 |     cout << "]\n";
19 |     cout << "y = [";
20 |     for (int i = 0; i < v.size(); ++i) {
21 |         if (i) {
22 |             cout << ", ";
23 |         }
24 |         cout << get<1>(v[i]);
25 |     }
26 |     cout << "]\n";
27 | }
28 | bool border(double a, double b) {
29 |     return (a < b) or (abs(b - a) < EPS);
30 | }
31 | class euler {
32 | private:
33 |     double l, r, y0, z0;
34 |     function<double(double, double, double)> f, g;
35 | public:
36 |     euler(const double _l, const double _r,
37 |           const function<double(double, double, double)> _f, const function<double(double, double, double)>
           _g,
38 |           const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
39 |     vector<tuple<double, double, double>> solve(double h) {
40 |         double xi = l;
```

```

41     double yi = y0;
42     double zi = z0;
43     vector<tuple<double, double, double>> res;
44     res.push_back(make_tuple(xi, yi, zi));
45     while (border(xi + h, r)) {
46         double dy = h * f(xi, yi, zi);
47         double dz = h * g(xi, yi, zi);
48         xi += h;
49         yi += dy;
50         zi += dz;
51         res.push_back(make_tuple(xi, yi, zi));
52     }
53     return res;
54 }
55 };
56 class rungeKutta {
57 private:
58     double l, r, y0, z0;
59     function<double(double, double, double)> f, g;
60 public:
61     rungeKutta(const double _l, const double _r,
62               const function<double(double, double, double)> _f, const function<double(double, double, double)>
               _g,
63               const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
64     vector<tuple<double, double, double>> solve(double h) {
65         double xi = l;
66         double yi = y0;
67         double zi = z0;
68         vector<tuple<double, double, double>> res;
69         res.push_back(make_tuple(xi, yi, zi));
70         while (border(xi + h, r)) {
71             double k1 = h * f(xi, yi, zi);
72             double l1 = h * g(xi, yi, zi);
73             double k2 = h * f(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
74             double l2 = h * g(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
75             double k3 = h * f(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
76             double l3 = h * g(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
77             double k4 = h * f(xi + h, yi + k3, zi + l3);
78             double l4 = h * g(xi + h, yi + k3, zi + l3);
79             double dy = (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
80             double dz = (l1 + 2.0 * l2 + 2.0 * l3 + l4) / 6.0;
81             xi += h;
82             yi += dy;
83             zi += dz;
84             res.push_back(make_tuple(xi, yi, zi));
85         }
86         return res;
87     }
88 };
89 class adams {
90 private:
91     double l, r, y0, z0;
92     function<double(double, double, double)> f, g;
93 public:
94     adams(const double _l, const double _r,
95           const function<double(double, double, double)> _f, const function<double(double, double, double)>
           _g,
96           const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
97     double calc_tuple(function<double(double, double, double)> f, tuple<double, double, double> xyz) {
98         return f(get<0>(xyz), get<1>(xyz), get<2>(xyz));
99     }
100     vector<tuple<double, double, double>> solve(double h) {

```

```

101     if (1 + 3.0 * h > r) {
102         throw invalid_argument(" ");
103     }
104     rungeKutta firstPoints(1, 1 + 3.0 * h, f, g, y0, z0);
105     vector<tuple<double, double, double>> res = firstPoints.solve(h);
106     size_t cnt = res.size();
107     double xk = get<0>(res.back());
108     double yk = get<1>(res.back());
109     double zk = get<2>(res.back());
110     while (border(xk + h, r)) {
111         double dy = (h / 24.0) * (55.0 * calc_tuple(f, res[cnt - 1]) - 59.0 * calc_tuple(f, res[cnt -
112             2]) + 37.0 * calc_tuple(f, res[cnt - 3]) - 9.0 * calc_tuple(f, res[cnt - 4]));
113         double dz = (h / 24.0) * (55.0 * calc_tuple(g, res[cnt - 1]) - 59.0 * calc_tuple(g, res[cnt -
114             2]) + 37.0 * calc_tuple(g, res[cnt - 3]) - 9.0 * calc_tuple(g, res[cnt - 4]));
115         double xk1 = xk + h;
116         double yk1 = yk + dy;
117         double zk1 = zk + dz;
118         res.push_back(make_tuple(xk1, yk1, zk1));
119         ++cnt;
120         dy = (h / 24.0) * (9.0 * calc_tuple(f, res[cnt - 1]) + 19.0 * calc_tuple(f, res[cnt - 2]) - 5.0
121             * calc_tuple(f, res[cnt - 3])
122             + 1.0 * calc_tuple(f, res[cnt - 4]));
123         dz = (h / 24.0) * (9.0 * calc_tuple(g, res[cnt - 1]) + 19.0 * calc_tuple(g, res[cnt - 2]) - 5.0
124             * calc_tuple(g, res[cnt - 3])
125             + 1.0 * calc_tuple(g, res[cnt - 4]));
126         xk += h;
127         yk += dy;
128         zk += dz;
129         res.pop_back();
130         res.push_back(make_tuple(xk, yk, zk));
131     }
132     return res;
133 }
134 };
135 double rungeRomberg(const vector<tuple<double, double, double>> & y_2h, const vector<tuple<double, double,
136     double>> & y_h, double p) {
137     double coef = 1.0 / (pow(2, p) - 1.0);
138     double res = 0.0;
139     for (int i = 0; i < y_2h.size(); ++i) {
140         res = max(res, coef * abs(get<1>(y_2h[i]) - get<1>(y_h[2 * i])));
141     }
142     return res;
143 }
144 double f(double x, double y, double z) {
145     return z;
146 }
147 double g(double x, double y, double z) {
148     return (12*y/(x*x));
149 }
150 int main() {
151     cout.precision(5);
152     cout << fixed;
153     double l, r, y0, z0, h;
154     cin >> l >> r;
155     cin >> y0 >> z0 >> h;
156     euler mEuler(1, r, f, g, y0, z0);
157     vector<tuple<double, double, double>> solEuler = mEuler.solve(h);
158     cout << " : " << endl;
159     printData(solEuler);
160     double eulerError = rungeRomberg(mEuler.solve(h), mEuler.solve(h / 2), 1);
161     cout << " = " << eulerError << "\n\n";
162     rungeKutta mRunge(1, r, f, g, y0, z0);

```

```

158     vector<tuple<double, double, double>> solRunge = mRunge.solve(h);
159     cout << " -:" << endl;
160     printData(solRunge);
161     double rungeError = rungeRomberg(mRunge.solve(h), mRunge.solve(h / 2), 4);
162     cout << " = " << rungeError << "\n\n";
163     adams mAdams(l, r, f, g, y0, z0);
164     vector<tuple<double, double, double>> solAdams = mAdams.solve(h);
165     cout << " ::" << endl;
166     printData(solAdams);
167     double adamsError = rungeRomberg(mAdams.solve(h), mAdams.solve(h / 2), 4);
168     cout << " = " << adamsError << endl;
169 }

```

Входные данные

```

test:
1 2
2 1 0.1

```

Консоль

```

natalya@natalya-Ideapad-Z570:~/NumMeth/Lab4/lab4-1$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab4/lab4-1$ ./a.out <test
Метод Эйлера:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.10000,2.44000,2.98826,3.73986,4.70364,5.89640,7.34001,9.06002,11.08480,13.44514]
Погрешность = 1.18886

Метод Рунге-Кутты:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.21546,2.65236,3.31131,4.20604,5.35876,6.79765,8.55548,10.66883,13.17756,16.12455]
Погрешность = 0.00003

Метод Адамса:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.21546,2.65236,3.31131,4.20536,5.35786,6.79673,8.55461,10.66802,13.17683,16.12390]
Погрешность = 0.00007

```

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.

Исходный код

```
1 | #include <iostream>
2 | #include <cmath>
3 | #include <exception>
4 | #include <vector>
5 | #include <tuple>
6 | #include <functional>
7 |
8 | using namespace std;
9 |
10 | const double EPS = 0.000000001;
11 | template<class T>
12 | class tridiag_t {
13 | private:
14 |     int n;
15 |     std::vector<T> a;
16 |     std::vector<T> b;
17 |     std::vector<T> c;
18 | public:
19 |     tridiag_t(const int & _n) : n(_n), a(n), b(n), c(n) {}
20 |     tridiag_t(const std::vector<T> & _a, const std::vector<T> & _b, const std::vector<T> & _c) {
21 |         if (!(_a.size() == _b.size() and _a.size() == _c.size())) {
22 |             throw std::invalid_argument("Sizes of a, b, c are invalid");
23 |         }
24 |         n = _a.size();
25 |         a = _a;
26 |         b = _b;
27 |         c = _c;
28 |     }
29 |     std::vector<T> solve(const std::vector<T> & d) {
30 |         int m = d.size();
31 |         if (n != m) {
32 |             throw std::invalid_argument("Size of vector d is invalid");
33 |         }
34 |         std::vector<T> p(n);
35 |         p[0] = -c[0] / b[0];
36 |         std::vector<T> q(n);
37 |         q[0] = d[0] / b[0];
38 |         for (int i = 1; i < n; ++i) {
39 |             p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
40 |             q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
41 |         }
42 |         std::vector<T> x(n);
43 |         x.back() = q.back();
44 |         for (int i = n - 2; i >= 0; --i) {
45 |             x[i] = p[i] * x[i + 1] + q[i];
46 |         }
47 |         return x;
48 |     }
```

```

49     friend std::istream & operator >> (std::istream & in, tridiag_t<T> & tridiag) {
50         in >> tridiag.b[0] >> tridiag.c[0];
51         for (int i = 1; i < tridiag.n - 1; ++i) {
52             in >> tridiag.a[i] >> tridiag.b[i] >> tridiag.c[i];
53         }
54         in >> tridiag.a.back() >> tridiag.b.back();
55         return in;
56     }
57     ~tridiag_t() = default;
58 };
59 void printData(const std::vector<std::tuple<double, double, double>> & v) {
60     std::cout << "x = [";
61     for (int i = 0; i < v.size(); ++i) {
62         if (i) {
63             std::cout << ", ";
64         }
65         std::cout << std::get<0>(v[i]);
66     }
67     std::cout << "]\n";
68     std::cout << "y = [";
69     for (int i = 0; i < v.size(); ++i) {
70         if (i) {
71             std::cout << ", ";
72         }
73         std::cout << std::get<1>(v[i]);
74     }
75     std::cout << "]\n";
76 }
77 bool border(double a, double b) {
78     return (a < b) or (std::abs(b - a) < EPS);
79 }
80 class rungeKutta {
81 private:
82     double l, r, y0, z0;
83     std::function<double(double, double, double)> f, g;
84 public:
85     rungeKutta(const double _l, const double _r,
86         const std::function<double(double, double, double)> _f, const std::function<double(double, double,
87             double)> _g,
88         const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
89     std::vector<std::tuple<double, double, double>> solve(double h) {
90         double xi = l;
91         double yi = y0;
92         double zi = z0;
93         std::vector<std::tuple<double, double, double>> res;
94         res.push_back(std::make_tuple(xi, yi, zi));
95         while (border(xi + h, r)) {
96             double k1 = h * f(xi, yi, zi);
97             double l1 = h * g(xi, yi, zi);
98             double k2 = h * f(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
99             double l2 = h * g(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
100            double k3 = h * f(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
101            double l3 = h * g(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
102            double k4 = h * f(xi + h, yi + k3, zi + l3);
103            double l4 = h * g(xi + h, yi + k3, zi + l3);
104            double dy = (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
105            double dz = (l1 + 2.0 * l2 + 2.0 * l3 + l4) / 6.0;
106            xi += h;
107            yi += dy;
108            zi += dz;
109            res.push_back(std::make_tuple(xi, yi, zi));
110        }

```

```

110         return res;
111     }
112 };
113 double rungeRomberg(const std::vector<std::tuple<double, double, double>> & y_2h, const std::vector<std::
    tuple<double, double, double>> & y_h, double p) {
114     double coef = 1.0 / (std::pow(2, p) - 1.0);
115     double res = 0.0;
116     for (int i = 0; i < y_2h.size(); ++i) {
117         res = std::max(res, coef * std::abs(std::get<1>(y_2h[i]) - std::get<1>(y_h[2 * i])));
118     }
119     return res;
120 }
121 class shooting {
122 private:
123     double a, b;
124     std::function<double(double, double, double)> f, g;
125     double alpha, beta, y0;
126     double delta, gamma, y1;
127
128 public:
129     shooting(const double _a, const double _b,
130             const std::function<double(double, double, double)> _f,
131             const std::function<double(double, double, double)> _g,
132             const double _alpha, const double _beta, const double _y0,
133             const double _delta, const double _gamma, const double _y1)
134     : a(_a), b(_b), f(_f), g(_g),
135       alpha(_alpha), beta(_beta), y0(_y0),
136       delta(_delta), gamma(_gamma), y1(_y1) {}
137
138     double get_start_cond(double eta) {
139         return (y0 - alpha * eta) / beta;
140     }
141
142     double get_eta_next(double eta_prev, double eta, const std::vector<std::tuple<double, double, double>>
        sol_prev, const std::vector<std::tuple<double, double, double>> sol) {
143         double yb_prev = std::get<1>(sol_prev.back());
144         double zb_prev = std::get<2>(sol_prev.back());
145         double phi_prev = delta * yb_prev + gamma * zb_prev - y1;
146         double yb = std::get<1>(sol.back());
147         double zb = std::get<2>(sol.back());
148         double phi = delta * yb + gamma * zb - y1;
149         return eta - (eta - eta_prev) / (phi - phi_prev) * phi;
150     }
151     std::vector<std::tuple<double, double, double>> solve(double h, double eps) {
152         double eta_prev = 1.0;
153         double eta = 0.8;
154         while (1) {
155             double rungeKutta_z0_prev = get_start_cond(eta_prev);
156             rungeKutta_de_solver_prev(a, b, f, g, eta_prev, rungeKutta_z0_prev);
157             std::vector<std::tuple<double, double, double>> sol_prev = de_solver_prev.solve(h);
158
159             double rungeKutta_z0 = get_start_cond(eta);
160             rungeKutta_de_solver(a, b, f, g, eta, rungeKutta_z0);
161             std::vector<std::tuple<double, double, double>> sol = de_solver.solve(h);
162
163             double eta_next = get_eta_next(eta_prev, eta, sol_prev, sol);
164             if (std::abs(eta_next - eta) < eps) {
165                 return sol;
166             } else {
167                 eta_prev = eta;
168                 eta = eta_next;
169             }

```



```

170     }
171 }
172 };
173 class finDif {
174 private:
175     using fx = std::function<double(double)>;
176     using tridiag = tridiag_t<double>;
177
178     double a, b;
179     fx p, q, f;
180     double alpha, beta, y0;
181     double delta, gamma, y1;
182
183 public:
184     finDif(const double _a, const double _b,
185           const fx _p, const fx _q, const fx _f,
186           const double _alpha, const double _beta, const double _y0,
187           const double _delta, const double _gamma, const double _y1)
188       : a(_a), b(_b), p(_p), q(_q), f(_f),
189         alpha(_alpha), beta(_beta), y0(_y0),
190         delta(_delta), gamma(_gamma), y1(_y1) {}
191
192     std::vector<std::tuple<double, double, double>> solve(double h) {
193         int n = (b - a) / h;
194         std::vector<double> xk(n + 1);
195         for (int i = 0; i <= n; ++i) {
196             xk[i] = a + h * i;
197         }
198         std::vector<double> a(n + 1);
199         std::vector<double> b(n + 1);
200         std::vector<double> c(n + 1);
201         std::vector<double> d(n + 1);
202         b[0] = h * alpha - beta;
203         c[0] = beta;
204         d[0] = h * y0;
205         a.back() = -gamma;
206         b.back() = h * delta + gamma;
207         d.back() = h * y1;
208         for (int i = 1; i < n; ++i) {
209             a[i] = 1.0 - p(xk[i]) * h * 0.5;
210             b[i] = -2.0 + h * h * q(xk[i]);
211             c[i] = 1.0 + p(xk[i]) * h * 0.5;
212             d[i] = h * h * f(xk[i]);
213         }
214         tridiag sys_eq(a, b, c);
215         std::vector<double> yk = sys_eq.solve(d);
216         std::vector<std::tuple<double, double, double>> res;
217         for (int i = 0; i <= n; ++i) {
218             res.push_back(std::make_tuple(xk[i], yk[i], NAN));
219         }
220         return res;
221     }
222 };
223 double f(double x, double y, double z) {
224     return z;
225 }
226 double fx(double x) {
227     return 0.0;
228 }
229 double g(double x, double y, double z) {
230     return ((-4 * x) * z - (4 * x * x + 2) * y);
231 }

```

```

232 double px(double x) {
233     return 4 * x;
234 }
235 double qx(double x) {
236     return 4 * x * x + 2;
237 }
238 int main() {
239     cout.precision(5);
240     cout << fixed;
241     double h, eps;
242     cin >> h >> eps;
243     /*
244      3
245     alpha * y(a) + beta * y'(a) = y0
246     delta * y(b) + gamma * y'(b) = y1
247     */
248     double a = 0, b = 2;
249     double alpha = 0, beta = 1, y0 = 1;
250     double delta = 4, gamma = -1, y1 = 23 * exp(-4);
251     shooting mShooting(a, b, f, g, alpha, beta, y0, delta, gamma, y1);
252     vector<tuple<double, double, double>> shootingSol = mShooting.solve(h, eps);
253     cout << " : " << endl;
254     printData(shootingSol);
255     double shootingError = rungeRomberg(mShooting.solve(h, eps), mShooting.solve(h / 2, eps), 4);
256     cout << " = " << shootingError << "\n\n";
257     finDif mFinDif(a, b, px, qx, fx, alpha, beta, y0, delta, gamma, y1);
258     vector<tuple<double, double, double>> finDifSol = mFinDif.solve(h);
259     cout << " - : " << endl;
260     printData(finDifSol);
261     double finDifError = rungeRomberg(mFinDif.solve(h), mFinDif.solve(h / 2), 2);
262     cout << " = " << finDifError << "\n\n";
263 }

```

Входные данные

```

test:
5
0.1
0.0001

```

Консоль

```

natalya@natalya-Ideapad-Z570:~/NumMeth/m4-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/m4-2$ ./a.out <test
Метод стрельбы:
x = [0.00000,0.10000,0.20000,0.30000,0.40000,0.50000,0.60000,0.70000,0.80000,0.90000,1.00000,1.10000,
1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [1.00074,1.08979,1.15366,1.18879,1.19363,1.16878,1.11680,1.04192,0.94952,0.84556,0.73602,0.62642,
0.52140,0.42451,0.33814,0.26355,0.20103,0.15008,0.10967,0.07846,0.05495]
Погрешность вычислений = 0.00005

```

Конечно-разностный метод:

x = [0.00000,0.10000,0.20000,0.30000,0.40000,0.50000,0.60000,0.70000,0.80000,0.90000,1.00000,1.10000,
1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]

y = [0.69395,0.79395,0.87415,0.93002,0.95887,0.96000,0.93474,0.88619,0.81878,0.73777,0.64872,0.55692,
0.46700,0.38266,0.30653,0.24014,0.18407,0.13812,0.10151,0.07311,0.05165]

Погрешность вычислений = 0.04388