Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра математической кибернетики

Лабораторные работы по курсу «Численные методы»

Лабораторная работа №4

Студент: Ершов С.Г.
Преподаватель: Пивоваров Д.Е.
Дата:
Оценка:
Подпись:

Москва, 2024

4 Методы решения обыкновенных дифференциальных уравнений

1 Решение задачи Коши для ОДУ

1.1 Постановка задачи

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. 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.

1.2 Консоль

```
$ make
g++ -g -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
$ cat tests/1.in
1 2
2 4 0.1
$ ./solution <tests/1.in
Метод Эйлера:
x = [1.000000,1.100000,1.200000,1.300000,1.400000,1.500000,1.600000,1.700000,1.800
y = [2.000000,2.414842,2.858788,3.331076,3.831064,4.358201,4.912010,5.492073,6.098
Погрешность вычислений:
0.000006
Метод Рунге-Кутты:
x = [1.000000,1.100000,1.200000,1.300000,1.400000,1.500000,1.600000,1.700000,1.800
y = [2.000000,2.414842,2.858788,3.331076,3.831064,4.358201,4.912010,5.492073,6.098
Погрешность вычислений:
0.000000
Метод Адамса:
x = [1.000000,1.100000,1.200000,1.300000,1.400000,1.500000,1.600000,1.700000,1.800
y = [2.000000,2.414842,2.858788,3.331076,3.831062,4.358197,4.912005,5.492067,6.098
Погрешность вычислений:
0.000000
```

1.3 Исходный код

```
#ifndef SIMPLE_DESOLVE_HPP
#define SIMPLE_DESOLVE_HPP

#include <functional>

#include "../de_utils.hpp"

/* f(x, y, z) */
using func = std::function<double(double, double, double)>;
using vect = std::vector<tddd>;
using vec = std::vector<double>;

const double EPS = 1e-9;

bool leq(double a, double b) { return (a < b) or (std::abs(b - a) < EPS); }

class euler {
    private:
     double l, r;
     func f, g;
     double y0, z0;

    public:
     euler(const double _l, const double _r, const func _f, const func _g,
           const double _y0, const double _z0)
         : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}

     vect solve(double h) {
         vect res;
         double xk = l;
         double yk = y0;
         double zk = z0;
         res.push_back(std::make_tuple(xk, yk, zk));
         while (leq(xk + h, r)) {
             double dy = h * f(xk, yk, zk);
             double dz = h * g(xk, yk, zk);
             xk += h;
             yk += dy;
             zk += dz;
             res.push_back(std::make_tuple(xk, yk, zk));
         }
         return res;
     }
};

class runge {
    private:
```

```
48      double l, r;
49      func f, g;
50      double y0, z0;
51
52    public:
53      runge(const double _l, const double _r, const func _f, const func _g,
54            const double _y0, const double _z0)
55          : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
56
57      vect solve(double h) {
58          vect res;
59          double xk = l;
60          double yk = y0;
61          double zk = z0;
62          res.push_back(std::make_tuple(xk, yk, zk));
63          while (leq(xk + h, r)) {
64              double K1 = h * f(xk, yk, zk);
65              double L1 = h * g(xk, yk, zk);
66              double K2 = h * f(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
67              double L2 = h * g(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
68              double K3 = h * f(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
69              double L3 = h * g(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
70              double K4 = h * f(xk + h, yk + K3, zk + L3);
71              double L4 = h * g(xk + h, yk + K3, zk + L3);
72              double dy = (K1 + 2.0 * K2 + 2.0 * K3 + K4) / 6.0;
73              double dz = (L1 + 2.0 * L2 + 2.0 * L3 + L4) / 6.0;
74              xk += h;
75              yk += dy;
76              zk += dz;
77              res.push_back(std::make_tuple(xk, yk, zk));
78          }
79          return res;
80      }
81  };
82
83  class adams {
84    private:
85      double l, r;
86      func f, g;
87      double y0, z0;
88
89    public:
90      adams(const double _l, const double _r, const func _f, const func _g,
91            const double _y0, const double _z0)
92          : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
93
94      double calc_tuple(func f, tddd xyz) {
95          return f(std::get<0>(xyz), std::get<1>(xyz), std::get<2>(xyz));
96      }
```

```
97
98      vect solve(double h) {
99          if (l + 3.0 * h > r) {
100             throw std::invalid_argument("h is too big");
101         }
102         runge first_points(l, l + 3.0 * h, f, g, y0, z0);
103         vect res = first_points.solve(h);
104         size_t cnt = res.size();
105         double xk = std::get<0>(res.back());
106         double yk = std::get<1>(res.back());
107         double zk = std::get<2>(res.back());
108         while (leq(xk + h, r)) {
109             /* Predictor */
110              double dy = (h / 24.0) * (55.0 * calc_tuple(f, res[cnt - 1]) -
111                                        59.0 * calc_tuple(f, res[cnt - 2]) +
112                                        37.0 * calc_tuple(f, res[cnt - 3]) -
113                                        9.0 * calc_tuple(f, res[cnt - 4]));
114              double dz = (h / 24.0) * (55.0 * calc_tuple(g, res[cnt - 1]) -
115                                        59.0 * calc_tuple(g, res[cnt - 2]) +
116                                        37.0 * calc_tuple(g, res[cnt - 3]) -
117                                        9.0 * calc_tuple(g, res[cnt - 4]));
118             double xk1 = xk + h;
119             double yk1 = yk + dy;
120             double zk1 = zk + dz;
121             res.push_back(std::make_tuple(xk1, yk1, zk1));
122             ++cnt;
123             /* Corrector */
124             dy = (h / 24.0) * (9.0 * calc_tuple(f, res[cnt - 1]) +
125                                19.0 * calc_tuple(f, res[cnt - 2]) -
126                                5.0 * calc_tuple(f, res[cnt - 3]) +
127                                1.0 * calc_tuple(f, res[cnt - 4]));
128             dz = (h / 24.0) * (9.0 * calc_tuple(g, res[cnt - 1]) +
129                                19.0 * calc_tuple(g, res[cnt - 2]) -
130                                5.0 * calc_tuple(g, res[cnt - 3]) +
131                                1.0 * calc_tuple(g, res[cnt - 4]));
132             xk += h;
133             yk += dy;
134             zk += dz;
135             res.pop_back();
136             res.push_back(std::make_tuple(xk, yk, zk));
137         }
138         return res;
139     }
140 };
141
142 double runge_romberg(const vect& y_2h, const vect& y_h, double p) {
143     double coef = 1.0 / (std::pow(2, p) - 1.0);
144     double res = 0.0;
145     for (size_t i = 0; i < y_2h.size(); ++i) {
```

```
146             res = std::max(res, coef * std::abs(std::get<1>(y_2h[i]) -
147                                                 std::get<1>(y_h[2 * i])));
148         }
149         return res;
150     }
151
152     #endif /* SIMPLE_DESOLVE_HPP */
```

2 Решение краевых задач

2.1 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.

2.2 Консоль

```
$ make
g++ -g -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
$ cat tests/1.in
0.1  0.0001
$ ./solution <tests/1.in
Метод стрельбы:
x = [1.000000,1.100000,1.200000,1.300000,1.400000,1.500000,1.600000,1.700000,1.800
y = [3.082723,3.898207,4.896683,6.111204,7.580066,9.347569,11.464879,13.991012,16.
Погрешность вычислений:
0.142434
Конечно-разностный метод:
x = [1.000000,1.100000,1.200000,1.300000,1.400000,1.500000,1.600000,1.700000,1.800
y = [1.171219,1.986704,3.035416,4.365472,6.033397,8.105462,10.659212,13.785218,17.
Погрешность вычислений:
0.342752
```

2.3 Исходный код

```
#ifndef   BOUNDARY_SOLVER_HPP
#define   BOUNDARY_SOLVER_HPP

 #include  <cmath>

#include "../lab1_2/tridiag.hpp"
#include "../lab4_1/simple_desolve.hpp"

class shooting {
    private:
     double a, b;
     func f, g;
     double alpha, beta, y0;
     double delta, gamma, y1;

    public:
     shooting(const double _a, const double _b, const func _f, const func _g,
              const double _alpha, const double _beta, const double _y0,
              const double _delta, const double _gamma, const double _y1)
         : a(_a),
           b(_b),
           f(_f),
           g(_g),
           alpha(_alpha),
           beta(_beta),
           y0(_y0),
           delta(_delta),
           gamma(_gamma),
           y1(_y1) {}

     double get_start_cond(double eta) { return (y0 - alpha * eta) / beta; }

     double get_eta_next(double eta_prev, double eta, const vect sol_prev,
                         const vect sol) {
         double yb_prev  = std::get<1>(sol_prev.back());
         double zb_prev  = std::get<2>(sol_prev.back());
         double phi_prev = delta * yb_prev + gamma * zb_prev - y1;
         double yb = std::get<1>(sol.back());
         double zb = std::get<2>(sol.back());
         double phi = delta * yb + gamma * zb - y1;
         return eta - (eta - eta_prev) / (phi - phi_prev) * phi;
     }

     vect solve(double h, double eps) {
         double eta_prev = 1.0;
         double eta = 0.8;
         while (1) {
```

```
48              double runge_z0_prev = get_start_cond(eta_prev);
49              euler de_solver_prev(a, b, f, g, eta_prev, runge_z0_prev);
50              vect sol_prev = de_solver_prev.solve(h);
51
52              double runge_z0 = get_start_cond(eta);
53              euler de_solver(a, b, f, g, eta, runge_z0);
54              vect sol = de_solver.solve(h);
55
56              double eta_next = get_eta_next(eta_prev, eta, sol_prev, sol);
57              if (std::abs(eta_next - eta) < eps) {
58                  return sol;
59              } else {
60                  eta_prev = eta;
61                  eta = eta_next;
62              }
63          }
64      }
65  };
66
67  class fin_dif {
68      private:
69       using fx = std::function<double(double)>;
70       using tridiag = tridiag_t<double>;
71
72       double a, b;
73       fx p, q, f;
74       double alpha, beta, y0;
75       double delta, gamma, y1;
76
77      public:
78       fin_dif(const double _a, const double _b, const fx _p, const fx _q,
79               const fx _f, const double _alpha, const double _beta,
80               const double _y0, const double _delta, const double _gamma,
81               const double _y1)
82           : a(_a),
83             b(_b),
84             p(_p),
85             q(_q),
86             f(_f),
87             alpha(_alpha),
88             beta(_beta),
89             y0(_y0),
90             delta(_delta),
91             gamma(_gamma),
92             y1(_y1) {}
93
94       vect solve(double h) {
95           size_t n = (b - a) / h;
96           vec xk(n + 1);
```

```
97          for (size_t i = 0; i <= n; ++i) {
98              xk[i] = a + h * i;
99          }
100         vec a(n + 1);
101         vec b(n + 1);
102         vec c(n + 1);
103         vec d(n + 1);
104         b[0] = h * alpha - beta;
105         c[0] = beta;
106         d[0] = h * y0;
107         a.back() = -gamma;
108         b.back() = h * delta + gamma;
109         d.back() = h * y1;
110         for (size_t i = 1; i < n; ++i) {
111             a[i] = 1.0 - p(xk[i]) * h * 0.5;
112             b[i] = -2.0 + h * h * q(xk[i]);
113             c[i] = 1.0 + p(xk[i]) * h * 0.5;
114             d[i] = h * h * f(xk[i]);
115         }
116         tridiag sys_eq(a, b, c);
117         vec yk = sys_eq.solve(d);
118         vect res;
119         for (size_t i = 0; i <= n; ++i) {
120             res.push_back(std::make_tuple(xk[i], yk[i], NAN));
121         }
122         return res;
123     }
124 };
125
126 #endif /* BOUNDARY_SOLVER_HPP */
```