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: Первухин А.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: 17

17	$xy''-(x+1)y'+y=0,$ $y(1)=2+e,$ $y'(1)=1+e,$ $x\in[1,2], h=0.1$	$y=x+1+e^x$

Рис. 1: Входные данные

2 Результаты работы

```
------------ ------------
x: 1 y1: 4.71828 y2: 3.71828
x: 1.1 y1: 5.09011 y2: 3.99011
x: 1.2 y1: 5.48912 y2: 4.28912
x: 1.3 y1: 5.91803 y2: 4.61803
x: 1.4 y1: 6.37984 y2: 4.97984
x: 1.5 y1: 6.87782 y2: 5.37782
x: 1.6 y1: 7.4156 y2: 5.8156
x: 1.7 y1: 7.99716 y2: 6.29716
x: 1.8 y1: 8.62688 y2: 6.82688
x: 1.9 y1: 9.30957 y2: 7.40957
x: 2 y1: 10.0505 y2: 8.05052
    : 0.133333
   : 0.338533

------------  - :------------
x: 1 y1: 4.71828 y2: 3.71828
x: 1.1 y1: 5.10417 y2: 4.00417
x: 1.2 y1: 5.52012 y2: 4.32012
x: 1.3 y1: 5.9693 y2: 4.6693
x: 1.4 y1: 6.4552 y2: 5.0552
x: 1.5 y1: 6.98169 y2: 5.48169
```

```
x: 1.6 y1: 7.55303 y2: 5.95303
x: 1.7 y1: 8.17394 y2: 6.47394
x: 1.8 y1: 8.84964 y2: 7.04964
x: 1.9 y1: 9.58589 y2: 7.68589
x: 2 y1: 10.3891 y2: 8.38905
    : 0.133333
   : 5.66578e-06

------------ ------------
x: 1 y1: 4.71828 y2: 3.71828
x: 1.1 y1: 5.10417 y2: 4.00417
x: 1.2 y1: 5.52012 y2: 4.32012
x: 1.3 y1: 5.9693 y2: 4.6693
x: 1.4 y1: 6.45519 y2: 5.05519
x: 1.5 y1: 6.98166 y2: 5.48166
x: 1.6 y1: 7.55299 y2: 5.95299
x: 1.7 y1: 8.17388 y2: 6.47388
x: 1.8 y1: 8.84956 y2: 7.04956
x: 1.9 y1: 9.58577 y2: 7.68577
x: 2 y1: 10.3889 y2: 8.3889
    : 0.133333
   : 0.000156
```

3 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>

using namespace std;

double f1(double x, double y1, double y2) {
    return y2;
}

double f2(double x, double y1, double y2) {
    return ((x + 1) * y2 - y1) / x;
}

double exactSolution(double x) {
    return x + 1 + exp(x);
}

double RungeRomberg(double y_h, double y_h2, int p) {
    return fabs((y_h2 - y_h) / (pow(2, p) - 1));
}

vector<double> Euler(double h, double x0, double y10, double y20, int steps) {
```

```
25      vector<double> x(steps + 1), y1(steps + 1), y2(steps + 1);
26      x[0] = x0; y1[0] = y10; y2[0] = y20;
27      ofstream fout("output.txt");
28      for (int i = 0; i < steps; ++i) {
29          y1[i + 1] = y1[i] + h * f1(x[i], y1[i], y2[i]);
30          y2[i + 1] = y2[i] + h * f2(x[i], y1[i], y2[i]);
31          x[i + 1] = x[i] + h;
32      }
33
34      fout << "------------ ------------\n";
35      for (int i = 0; i <= steps; ++i) {
36          fout << "x: " << x[i] << " y1: " << y1[i] << " y2: " << y2[i] << '\n';
37      }
38      fout << "    : " << RungeRomberg(y1[steps], y2[steps], 4) << endl;
39      fout.close();
40      return y1;
41  }
42
43  vector<double> RungeKutt(double h, double x0, double y10, double y20, int steps) {
44      vector<double> x(steps + 1), y1(steps + 1), y2(steps + 1);
45      x[0] = x0; y1[0] = y10; y2[0] = y20;
46      ofstream fout("output.txt", ios::app);
47      for (int i = 0; i < steps; ++i) {
48          double k1_y1 = h * f1(x[i], y1[i], y2[i]);
49          double k1_y2 = h * f2(x[i], y1[i], y2[i]);
50
51          double k2_y1 = h * f1(x[i] + h/2, y1[i] + k1_y1/2, y2[i] + k1_y2/2);
52          double k2_y2 = h * f2(x[i] + h/2, y1[i] + k1_y1/2, y2[i] + k1_y2/2);
53
54          double k3_y1 = h * f1(x[i] + h/2, y1[i] + k2_y1/2, y2[i] + k2_y2/2);
55          double k3_y2 = h * f2(x[i] + h/2, y1[i] + k2_y1/2, y2[i] + k2_y2/2);
56
57          double k4_y1 = h * f1(x[i] + h, y1[i] + k3_y1, y2[i] + k3_y2);
58          double k4_y2 = h * f2(x[i] + h, y1[i] + k3_y1, y2[i] + k3_y2);
59
60          y1[i + 1] = y1[i] + (k1_y1 + 2*k2_y1 + 2*k3_y1 + k4_y1) / 6;
61          y2[i + 1] = y2[i] + (k1_y2 + 2*k2_y2 + 2*k3_y2 + k4_y2) / 6;
62          x[i + 1] = x[i] + h;
63      }
64      fout << endl;
65      fout << "------------  - :------------\n";
66      for (int i = 0; i <= steps; ++i) {
67          fout << "x: " << x[i] << " y1: " << y1[i] << " y2: " << y2[i] << '\n';
68      }
69      fout << "    : " << RungeRomberg(y1[steps], y2[steps], 4) << endl;
70      fout.close();
71      return y1;
72  }
73
```

```
74 vector<double> Adams(double h, double x0, double y10, double y20, int steps) {vector<
       double> x(steps + 1), y1(steps + 1), y2(steps + 1);
75     x[0] = x0; y1[0] = y10; y2[0] = y20;
76     ofstream fout("output.txt", ios::app);
77
78     for (int i = 0; i < 3; ++i) {
79         double k1_y1 = h * f1(x[i], y1[i], y2[i]);
80         double k1_y2 = h * f2(x[i], y1[i], y2[i]);
81
82         double k2_y1 = h * f1(x[i] + h / 2, y1[i] + k1_y1 / 2, y2[i] + k1_y2 / 2);
83         double k2_y2 = h * f2(x[i] + h / 2, y1[i] + k1_y1 / 2, y2[i] + k1_y2 / 2);
84
85         double k3_y1 = h * f1(x[i] + h / 2, y1[i] + k2_y1 / 2, y2[i] + k2_y2 / 2);
86         double k3_y2 = h * f2(x[i] + h / 2, y1[i] + k2_y1 / 2, y2[i] + k2_y2 / 2);
87
88         double k4_y1 = h * f1(x[i] + h, y1[i] + k3_y1, y2[i] + k3_y2);
89         double k4_y2 = h * f2(x[i] + h, y1[i] + k3_y1, y2[i] + k3_y2);
90
91         y1[i + 1] = y1[i] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
92         y2[i + 1] = y2[i] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
93         x[i+1] = x[i] + h;
94     }
95
96     for (int i = 3; i < steps; ++i) {
97         double f1i = f1(x[i], y1[i], y2[i]);
98         double f2i = f2(x[i], y1[i], y2[i]);
99
100        double f1im1 = f1(x[i] - h, y1[i - 1], y2[i - 1]);
101        double f2im1 = f2(x[i] - h, y1[i - 1], y2[i - 1]);
102
103        double f1im2 = f1(x[i] - 2 * h, y1[i - 2], y2[i - 2]);
104        double f2im2 = f2(x[i] - 2 * h, y1[i - 2], y2[i - 2]);
105
106        double f1im3 = f1(x[i] - 3 * h, y1[i - 3], y2[i - 3]);
107        double f2im3 = f2(x[i] - 3 * h, y1[i - 3], y2[i - 3]);
108
109        y1[i + 1] = y1[i] + (h / 24) * (55 * f1i - 59 * f1im1 + 37 * f1im2 - 9 * f1im3)
               ;
110        y2[i + 1] = y2[i] + (h / 24) * (55 * f2i - 59 * f2im1 + 37 * f2im2 - 9 * f2im3)
               ;
111        x[i + 1] = x[i] + h;
112    }
113    fout << endl;
114    fout << "------------ ------------" << endl;
115    for (int i = 0; i <= steps; ++i) {
116        fout << "x: " << x[i] << " y1: " << y1[i] << " y2: " << y2[i] << '\n';
117    }
118    fout << "    : " << RungeRomberg(y1[steps], y2[steps], 4) << endl;
119    fout.close();
```

```
120     return y1;
121 }
122 
123 void Error(const vector<double>& numeric, double x0, double h, int steps) {
124     double maxError = 0.0;
125     ofstream fout("output.txt", ios::app);
126     for (int i = 0; i <= steps; ++i) {
127         double exact = exactSolution(x0 + i * h);
128         double error = fabs(numeric[i] - exact);
129         if (error > maxError) {
130             maxError = error;
131         }
132     }
133     fout << "   : " << maxError << '\n';
134 }
135 
136 int main() {
137     double h = 0.1;
138     double x0 = 1.0;
139     double y10 = 2 + exp(1);
140     double y20 = 1 + exp(1);
141     int steps = (int)((2.0 - 1.0) / h);
142 
143     vector<double> eulerResult = Euler(h, x0, y10, y20, steps);
144     Error(eulerResult, x0, h, steps);
145 
146     vector<double> rungeKuttaResult = RungeKutt(h, x0, y10, y20, steps);
147     Error(rungeKuttaResult, x0, h, steps);
148 
149     vector<double> adamsResult = Adams(h, x0, y10, y20, steps);
150     Error(adamsResult, x0, h, steps);
151 
152     return 0;
153 }
```

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: 17

17	$(x^2-1)\,y''+(x-3)y'-y=0$, $y'(0)=0$, $y'(1)+y(1)=-0.75$	$y(x)=x-3+\frac{1}{x+1}$

Рис. 2: Входные данные

5 Результаты работы

```
Метод стрельбы:
x = 1.000000, y = 0.000000, точное y = -1.500000
Погрешность метода стрельбы (метод Рунге-Ромберга): 0.000000

Метод конечных разностей:
x = 1.000000, y = -0.746958, точное y = -1.500000
Погрешность метода конечных разностей (метод Рунге-Ромберга): 0.000506
```

Рис. 3: Вывод программы

6 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <functional>
#include <iomanip>
#include <fstream>

using namespace std;


double ExactSolution(double x) {
    return x - 3 + 1 / (x + 1);
}

vector<double> Odes(double x, const vector<double>& Y) {
    double y = Y[0];
    double dy = Y[1];
    double d2y = (y - (x - 3) * dy) / (x * x - 1);
    return {dy, d2y};
}

vector<vector<double>> RungeKutta(function<vector<double>(double, const vector<double
    >&)> f,
                                  double x0, const vector<double>& Y0, double xf, int N
                                      ) {
    vector<vector<double>> result;
    result.push_back(Y0);
    double h = (xf - x0) / N;
    double x = x0;
    vector<double> Y = Y0;
    for (int i = 1; i <= N; ++i) {
        vector<double> k1 = f(x, Y);
        vector<double> k2 = f(x + h / 2, {Y[0] + h / 2 * k1[0], Y[1] + h / 2 * k1[1]});
        vector<double> k3 = f(x + h / 2, {Y[0] + h / 2 * k2[0], Y[1] + h / 2 * k2[1]});
        vector<double> k4 = f(x + h, {Y[0] + h * k3[0], Y[1] + h * k3[1]});
        Y[0] += h / 6 * (k1[0] + 2 * k2[0] + 2 * k3[0] + k4[0]);
        Y[1] += h / 6 * (k1[1] + 2 * k2[1] + 2 * k3[1] + k4[1]);
        x += h;
        result.push_back(Y);
    }
    return result;
}

double ShootingFunc(double s, double x_end, int N) {
    vector<double> Y0 = {0, s};
    auto sol = RungeKutta(Odes, 0, Y0, x_end, N);
    double y1 = sol.back()[0];
```

```
46      double dy1 = sol.back()[1];
47      return dy1 + y1 + 0.75;
48  }
49
50  vector<vector<double>> Shooting(double s_guess, double x_end, int N) {
51      double s = s_guess;
52      double epsilon = 1e-6;
53      double delta = 1e-4;
54      double f_s = ShootingFunc(s, x_end, N);
55
56      while (abs(f_s) > epsilon) {
57          double f_s_delta = ShootingFunc(s + delta, x_end, N);
58          double s_new = s - f_s * delta / (f_s_delta - f_s);
59          s = s_new;
60          f_s = ShootingFunc(s, x_end, N);
61      }
62
63      return RungeKutta(Odes, 0, {0, s}, x_end, N);
64  }
65
66  vector<double> FiniteDifference(int N) {
67      double h = 1.0 / N;
68      vector<double> x(N + 1);
69      vector<double> A(N + 1);
70      vector<double> B(N + 1);
71      vector<double> C(N + 1);
72      vector<double> D(N + 1);
73      vector<double> y(N + 1, 0.0);
74
75      for (int i = 0; i <= N; ++i) {
76          x[i] = i * h;
77      }
78
79      for (int i = 1; i < N; ++i) {
80          double xi = x[i];
81          A[i] = (xi * xi - 1) / (h * h) - (xi - 3) / (2 * h);
82          B[i] = -2 * (xi * xi - 1) / (h * h);
83          C[i] = (xi * xi - 1) / (h * h) + (xi - 3) / (2 * h);
84          D[i] = 0;
85      }
86
87      B[0] = 1; D[0] = 0; // y'(0) = 0
88      B[N] = 1 + h; A[N] = -1; D[N] = -0.75 * h; // y'(1) + y(1) = -0.75
89
90      for (int i = 1; i <= N; ++i) {
91          double m = A[i] / B[i - 1];
92          B[i] -= m * C[i - 1];
93          D[i] -= m * D[i - 1];
94      }
```

```
95
96      y[N] = D[N] / B[N];
97      for (int i = N - 1; i >= 0; --i) {
98          y[i] = (D[i] - C[i] * y[i + 1]) / B[i];
99      }
100
101     return y;
102 }
103
104 double RungeRomberg(const vector<double>& y2h, const vector<double>& yh, int N) {
105     double error = 0.0;
106     for (int i = 0; i <= N; ++i) {
107         error = max(error, abs(y2h[2 * i] - yh[i]) / 3.0);
108     }
109     return error;
110 }
111
112 int main() {
113     ofstream fout("output.txt");
114     fout << fixed << setprecision(6);
115     int N = 100;
116     double x_end = 1.0;
117
118     double s_guess = -1;
119     auto sol_shooting = Shooting(s_guess, x_end, N);
120     auto sol_shooting_2N = Shooting(s_guess, x_end, 2 * N);
121
122     auto sol_fd = FiniteDifference(N);
123     auto sol_fd_2N = FiniteDifference(2 * N);
124
125     vector<double> x(N + 1);
126     vector<double> exact(N + 1);
127     for (int i = 0; i <= N; ++i) {
128         x[i] = i * 1.0 / N;
129         exact[i] = ExactSolution(x[i]);
130     }
131
132     double error_fd = RungeRomberg(sol_fd_2N, sol_fd, N);
133     vector<double> y_shooting(N + 1);
134     vector<double> y_shooting_2N(2 * N + 1);
135     for (int i = 0; i <= N; ++i) {
136         y_shooting[i] = sol_shooting[i][0];
137     }
138     for (int i = 0; i <= 2 * N; ++i) {
139         y_shooting_2N[i] = sol_shooting_2N[i][0];
140     }
141     double error_shooting = RungeRomberg(y_shooting_2N, y_shooting, N);
142
143     fout << " :\n";
```

```
144     fout << "x = " << x[N] << ", y = " << sol_shooting[N][0] << ",  y = " << exact[N]
            << "\n";
145 
146     fout << "   ( -): " << error_shooting << "\n";
147 
148     fout << "\n  :\n";
149     fout << "x = " << x[N] << ", y = " << sol_fd[N] << ",  y = " << exact[N];
150 
151     fout << "\n    ( -): " << error_fd << "\n";
152 
153     return 0;
154 }
```