Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент: О. В. Гребнева
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

4.1. Решение задачи Коши для ОДУ 2-го порядка на указанном отрезке с помощью методов 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 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.

Вариант: 5

Задача Коши: $y'' - (1 + 2tg^2x)y = 0,$
$y(0) = 1,$
$y'(0) = 2,$
$x \in [0, 1], h = 0.1$

Точное решение: $y = \frac{1}{cos(x)} + sin(x) + \frac{x}{cos(x)}$

2 Результаты работы

```
Значения x:
0.0000 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000
Точное решение y:
1.0000 1.2054 1.4231 1.6563 1.9094 2.1887 2.5032 2.8669 3.3009 3.8399 4.5431

Метод Эйлера:
1.0000 1.2000 1.4100 1.6322 1.8697 2.1267 2.4090 2.7253 3.0882 3.5171 4.0423
Метод Рунге-Кутты:
1.0000 1.2044 1.4208 1.6521 1.9024 2.1776 2.4862 2.8410 3.2614 3.7790 4.4467
Метод Адамса:
1.0000 1.2044 1.4208 1.6521 1.8935 2.1574 2.4509 2.7847 3.1749 3.6462 4.2392

Погрешности методом Рунге-Ромберга-Ричардсона
Для Эйлера:
0.0000 0.0100 0.0297 0.0684 0.1471 0.3194
Для Рунге-Кутты:
0.0000 0.0001 0.0004 0.0010 0.0022 0.0053
Для Адамса:
0.0000 0.0001 0.0002 0.0014 0.0031 0.0126

Погрешности сравнением с точным решением
Для Эйлера:
0.0000 0.0054 0.0131 0.0241 0.0397 0.0620 0.0942 0.1416 0.2127 0.3228 0.5008
Для Рунге-Кутты:
0.0000 0.0009 0.0023 0.0042 0.0070 0.0111 0.0171 0.0259 0.0395 0.0609 0.0964
Для Адамса:
0.0000 0.0009 0.0023 0.0042 0.0159 0.0313 0.0524 0.0822 0.1261 0.1937 0.3039
```

3 Исходный код

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

using namespace std;

double f(double x, double y){
    return (1 + 2*tan(x)*tan(x))*y;
}

double accurate_solution(double x){
    return 1/cos(x) + sin(x) + x/cos(x);
```

```
14 }
15
16 vector<double> euler(double a, double b, double h, vector<double> x, int n){
17     vector<double> y(n, 1.0);
18     vector<double> z(n, 2.0);
19
20     for (int i = 1; i < n; ++i) {
21         y[i] = y[i - 1] + h * z[i - 1];
22         z[i] = z[i - 1] + h * f(x[i - 1], y[i - 1]);
23     }
24
25     return y;
26 }
27
28 pair<vector<double>, vector<double>> runge_kutta(double a, double b, double h, vector<
       double> x, int n) {
29     vector<double> y(n, 1.0);
30     vector<double> z(n, 2.0);
31
32     vector<double> K(4);
33     vector<double> L(4);
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]);
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);
43         }
44
45         double dy = (K[0] + 2 * K[1] + 2 * K[2] + K[3]) / 6;
46         double dz = (L[0] + 2 * L[1] + 2 * L[2] + L[3]) / 6;
47         y[i] = y[i - 1] + dy;
48         z[i] = z[i - 1] + dz;
49     }
50
51     return make_pair(y, z);
52 }
53
54 vector<double> adams(double a, double b, double h, vector<double> x, int n) {
55     vector<double> y(n, 0.0);
56     vector<double> z(n, 0.0);
57
58     vector<double> y0 = runge_kutta(a, a + 3 * h, h, x, n).first;
59     for (int i = 0; i < y0.size(); ++i) {
60         y[i] = y0[i];
61     }
```

```
62
63      vector<double> z0 = runge_kutta(a, a + 3 * h, h, x, n).second;
64      for (int i = 0; i < z0.size(); ++i) {
65          z[i] = z0[i];
66      }
67
68      for (int i = 4; i < n; ++i) {
69          y[i] = y[i - 1] + h * z[i - 1];
70          z[i] = z[i - 1] + h * (55 * f(x[i - 1], y[i - 1]) - 59 * f(x[i - 2], y[i - 2])
71                      + 37 * f(x[i - 3], y[i - 3]) - 9 * f(x[i - 4], y[i - 4])) / 24;
72      }
73
74      return y;
75  }
76
77  vector<vector<double>> RRR_inaccuracy(double a, double b, double h, vector<double> x,
        int n) {
78      vector<double> euler1(n);
79      vector<double> runge_kutta1(n);
80      vector<double> adams1(n);
81
82      double h2 = h*2;
83      int n2 = (b - a) / h2 + 1;
84      //cout << n2 << endl;
85      vector<double> x2(n);
86      for (int i = 0; i < n2; ++i){
87          x2[i] = h2 * i;
88          //cout << x2[i] << " ";
89      }
90
91      vector<double> euler_h = euler(a, b, h, x, n);
92      vector<double> euler_2h = euler(a, b, h2, x2, n2);
93
94      vector<double> runge_kutta_h = runge_kutta(a, b, h, x, n).first;
95      vector<double> runge_kutta_2h = runge_kutta(a, b, h2, x2, n2).first;
96
97      vector<double> adams_h = adams(a, b, h, x, n);
98      vector<double> adams_2h = adams(a, b, h2, x2, n2);
99
100
101     for (int i = 0; i < n2; ++i) {
102         //      = 1
103         euler1[i] = abs(euler_h[i*2] - euler_2h[i]);
104         runge_kutta1[i] = abs((runge_kutta_h[i*2] - runge_kutta_2h[i]) / 15);
105         adams1[i] = abs((adams_h[i*2] - adams_2h[i]) / 15);
106     }
107
108     return {euler1, runge_kutta1, adams1};
109 }
```

```
110
111 int main() {
112     double a = 0.0;
113     double b = 1.0;
114     double h = 0.1;
115
116     int n = (b - a) / h + 1;
117     int n2 = (b - a) / 2 / h + 1;
118     vector<double> x(n), y(n);
119
120     ofstream fout("answer.txt");
121     fout.precision(4);
122     fout << fixed;
123
124     for (int i = 0; i < n; ++i){
125         x[i] = h * i;
126         y[i] = accurate_solution(x[i]);
127     }
128
129     fout << " x:" << endl;
130     for (int i = 0; i < n; ++i){
131         fout << x[i] << " ";
132     }
133
134     fout << endl << "  y:" << endl;
135     for (int i = 0; i < n; ++i){
136         fout << y[i] << " ";
137     }
138     fout << endl;
139
140     fout << endl << " :" << endl;
141     for (int i = 0; i < n; ++i){
142         fout << euler(a, b, h, x, n)[i] << " ";
143     }
144
145     fout << endl << " -:" << endl;
146     for (int i = 0; i < n; ++i){
147         fout << runge_kutta(a, b, h, x, n).first[i] << " ";
148     }
149
150     fout << endl << " :" << endl;
151     for (int i = 0; i < n; ++i){
152         fout << adams(a, b, h, x, n)[i] << " ";
153     }
154     fout << endl;
155
156     fout << endl << "  --" << endl << " :" << endl;
157     for (int i = 0; i < n2; ++i){
158         fout << abs(RRR_inaccuracy(a, b, h, x, n)[0][i]) << " ";
```

```
159     }
160 
161     fout << endl << " -:" << endl;
162     for (int i = 0; i < n2; ++i){
163         fout << abs(RRR_inaccuracy(a, b, h, x, n)[1][i]) << " ";
164     }
165     fout << endl << " :" << endl;
166     for (int i = 0; i < n2; ++i){
167         fout << abs(RRR_inaccuracy(a, b, h, x, n)[2][i]) << " ";
168     }
169     fout << endl;
170 
171     fout << endl << "    " << endl << " :" << endl;
172     for (int i = 0; i < n; ++i){
173         fout << abs(euler(a, b, h, x, n)[i] - y[i]) << " ";
174     }
175 
176     fout << endl << " -:" << endl;
177     for (int i = 0; i < n; ++i){
178         fout << abs(runge_kutta(a, b, h, x, n).first[i] - y[i]) << " ";
179     }
180     fout << endl << " :" << endl;
181     for (int i = 0; i < n; ++i){
182         fout << abs(adams(a, b, h, x, n)[i] - y[i]) << " ";
183     }
184 
185     return 0;
186 }
```

4.2. Решение краевой задачи для ОДУ 2-го порядка на указанном отрезке с помощью метода стрельбы и конечно-разностного метода

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.

Вариант: 30 (был заменён, так как в 5 варианте неверно указано точное решение)

Краевая задача: $(x^2+1)y'' - 2y = 0,$
$y'(0) = 0,$
$y(2) - y'(2) = 1$

Точное решение: $y(x) = x^2 + 1$

2 Результаты работы

```
Значения x:
0.0000 0.2000 0.4000 0.6000 0.8000 1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
Точное решение y:
1.0000 1.0400 1.1600 1.3600 1.6400 2.0000 2.4400 2.9600 3.5600 4.2400 5.0000

Метод shooting:
1.0000 1.0390 1.1590 1.3580 1.6380 1.9980 2.4380 2.9590 3.5590 4.2390 5.0000
Метод difference:
1.0000 0.8350 0.8540 0.9420 1.1040 1.3560 1.7160 2.2130 2.8870 3.7920 5.0000

Погрешности методом Рунге-Ромберга-Ричардсона

Метод shooting погрешность:
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Метод difference погрешность:
0.0000 0.0030 0.0020 0.0010 0.0010 0.0010 0.0000 0.0000 0.0000 0.0000 0.0000

Погрешности сравнением с точным решением

Метод shooting погрешность:
0.0000 0.0010 0.0010 0.0020 0.0020 0.0020 0.0020 0.0010 0.0010 0.0010 0.0000
Метод difference погрешность:
0.0000 0.2050 0.2650 0.3900 0.5160 0.6320 0.7180 0.7450 0.6730 0.4490 0.0000
```

3 Исходный код

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

using namespace std;

double f(double x, double y, double z){
    return (2*y)/(x*x+1);
}

double accurate_solution(double x){
    return x*x+1;
}

double p(double x){
    return 0;
}

double q(double x){
    return -2;
}

vector<double> runge_kutta(double a, double b, double h, vector<double> x, int n,
    double y0, double z0) {
    vector<double> y(n);
    vector<double> z(n);

    vector<double> K(4);
    vector<double> L(4);

    y[0] = y0;
    z[0] = z0;

    for (int i = 1; i < n; ++i) {

        K[0] = h * z[i - 1];
        L[0] = h * f(x[i - 1], y[i - 1], z[i-1]);

        for (int j = 1; j < 4; ++j) {
            K[j] = h * (z[i - 1] + L[j - 1] / 2);
            L[j] = h * f(x[i - 1] + h / 2, y[i - 1] + K[j - 1] / 2, z[i - 1] + L[j - 1]
                / 2);
        }

        double dy = (K[0] + 2 * K[1] + 2 * K[2] + K[3]) / 6;
        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,
        double eps, double y0, double y1){
54      double eta0 = 1;
55      double eta = 0.8;
56
57      double F0 = runge_kutta(a, b, h, x, n, y0, eta0)[n-1] - y1;
58      double F = runge_kutta(a, b, h, x, n, y0, eta)[n-1] - y1;
59
60      while(abs(F) > eps){
61          double c = eta;
62          eta = eta - F*(eta - eta0)/(F - F0);
63          eta0 = c;
64          F0 = F;
65          F = runge_kutta(a, b, h, x, n, y0, eta)[n - 1] - y1;
66      }
67      return runge_kutta(a, b, h, x, n, y0, eta);
68  }
69
70  vector<double> difference_method(double a, double b, double h, vector<double> x, int n
        , double alpha, double beta, double delta, double gamma, double y0, double y1) {
71      vector<double> A, B, C, D, P(n), Q(n), sol(n);
72
73      A.push_back(0);
74      B.push_back(-2 + h * h * q(x[1]));
75      C.push_back(1 + p(x[1]) * h / 2);
76      D.push_back(-(1 - (p(x[1]) * h) / 2) * y0);
77      for (int i = 2; i < n; ++i) {
78
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      sol[n-1] = Q[n-1];
97      for (int i = n - 2; i > 0; --i)
98          sol[i] = P[i] * sol[i + 1] + Q[i];
99      sol[0] = y0;
100     sol[n] = y1;
101     return sol;
102 }
103
104 pair<vector<double>, vector<double>> RRR_inaccuracy(double a, double b, double h,
        vector<double> x, int n, double eps, double y0, double y1, double alpha, double
        beta, double gamma, double delta) {
105     vector<double> shooting_method1(n), difference_method1(n);
106
107     double h2 = h/2;
108     int n2 = (b - a) / h2 + 1;
109
110     vector<double> x2(n);
111     for (int i = 0; i < n2; ++i){
112         x2[i] = h2 * i;
113     }
114     vector<double> shooting_method_h = shooting_method(a, b, h, x, n, eps, y0, y1);
115     vector<double> shooting_method_h2 = shooting_method(a, b, h2, x2, n2, eps, y0, y1);
116     vector<double> difference_method_h = difference_method(a, b, h, x, n, alpha, beta,
            delta, gamma, y0, y1);
117     vector<double> difference_method_h2 = difference_method(a, b, h2, x2, n2, alpha,
            beta, delta, gamma, y0, y1);
118
119     for (int i = 0; i < n; ++i) {
120         shooting_method1[i] = (shooting_method_h2[2 * i] - shooting_method_h[i]) / 15;
121         difference_method1[i] = (difference_method_h2[2 * i] - difference_method_h[i])
                / 15;
122     }
123     return make_pair(shooting_method1, difference_method1);
124 }
125
126 int main(){
127     double a = 0;
128     double b = 2;
129     double alpha = 0;
130     double beta = 1;
131     double delta = -1;
132     double gamma = 1;
133
134     double y0 = a*a+1;
135     double y1 = b*b+1;
136     double h = 0.2;
```

```
137     double eps = 0.001;
138
139     ofstream fout("answer.txt");
140     fout.precision(4);
141     fout << fixed;
142
143     int n = 11;
144
145     vector<double> x(n), y(n);
146     for (int i = 0; i < n; ++i){
147         x[i] = h * i + a;
148         y[i] = accurate_solution(x[i]);
149     }
150
151     vector<double> shoot = shooting_method(a, b, h, x, n, eps, y0, y1);
152     fout << " x:" << endl;
153     for (int i = 0; i < n; ++i){
154         fout << x[i] << " ";
155     }
156
157     fout << endl << "  y:" << endl;
158     for (int i = 0; i < n; ++i){
159         fout << y[i] << " ";
160     }
161     fout << endl;
162
163     fout << endl << " shooting:" << endl;
164     for (int i = 0; i < n; ++i){
165         fout << shoot[i] << " ";
166     }
167
168     vector<double> diff = difference_method(a, b, h, x, n, alpha, beta, delta, gamma,
            y0, y1);
169
170     fout << endl << " difference:" << endl;
171     for (int i = 0; i < n; ++i){
172         fout << diff[i] << " ";
173     }
174     fout << endl;
175
176     fout << endl << "  --" << endl;
177     vector<double> r = RRR_inaccuracy(a, b, h, x, n, eps, y0, y1, alpha, beta, gamma,
            delta).first;
178
179     fout << endl << " shooting :" << endl;
180     for (int i = 0; i < n; ++i){
181         fout << r[i] << " ";
182     }
183
```

```
184     vector<double> rr = RRR_inaccuracy(a, b, h, x, n, eps, y0, y1, alpha, beta, gamma,
            delta).second;
185 
186     fout << endl << " difference :" << endl;
187     for (int i = 0; i < n; ++i){
188         fout << rr[i] << " ";
189     }
190     fout << endl;
191 
192     fout << endl << "    " << endl;
193 
194     fout << endl << " shooting :" << endl;
195     for (int i = 0; i < n; ++i){
196         fout << abs(shoot[i] - y[i]) << " ";
197     }
198 
199     fout << endl << " difference :" << endl;
200     for (int i = 0; i < n; ++i){
201         fout << abs(diff[i] - y[i]) << " ";
202     }
203 
204     return 0;
205 }
```