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: М. Р. Жалялетдинов
Преподаватель: Д. Е. Пивоваров
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: 8

$$y''-4xy'+(4x^2-3)y-e^{x^2}=0,$$
$$y(0)=1,$$
$$y'(0)=0,$$
$$x\in[0,1], h=0.1$$

$$y=(e^x+e^{-x}-1)e^{x^2}$$

Рис. 1: Входные данные

2 Результаты работы

Euler:

X	Y	Yprec	Error	
0	1	1	0	
0.1	1	1.02016	0.0201591	
0.2	1.04	1.08258	0.0425822	
0.3	1.1213	1.19339	0.0720903	
0.4	1.24905	1.36379	0.11474	
0.5	1.43267	1.61178	0.179104	
0.6	1.68689	1.96499	0.278108	
0.7	2.03344	2.46535	0.431911	
0.8	2.50381	3.17636	0.672548	
0.9	3.14336	4.19498	1.05163	
1	4.01754	5.67077	1.65323	

Runge:

X	Y	Yprec	Error	
0	1	1	0	
0.1	1.02016	1.02016	3.69835e-07	
0.2	1.08258	1.08258	2.46406e-06	
0.3	1.19338	1.19339	7.14653e-06	
0.4	1.36377	1.36379	1.63087e-05	
0.5	1.61174	1.61178	3.35668e-05	
0.6	1.96493	1.96499	6.56222e-05	
0.7	2.46522	2.46535	0.00012483	
0.8	3.17612	3.17636	0.000234037	
0.9	4.19455	4.19498	0.000435728	
1	5.66996	5.67077	0.000809489	

Adam:

X	Y	Yprec	Error
0	1	1	0
0.1	1.02016	1.02016	3.69835e-07
0.2	1.08258	1.08258	2.46406e-06
0.3	1.19338	1.19339	7.14653e-06
0.4	1.36336	1.36379	0.000434318
0.5	1.60998	1.61178	0.00179953
0.6	1.96086	1.96499	0.00413489
0.7	2.45693	2.46535	0.0084145
0.8	3.16035	3.17636	0.016007
0.9	4.16557	4.19498	0.0294193
1	5.61769	5.67077	0.0530816

Euler error by Romberg:

X	Y err	Y' err	
0	0	0	
0.1	0.000666667	0.000108344	
0.2	0.00137088	0.0010626	
0.3	0.00226314	0.00301627	
0.4	0.00351998	0.00636803	
0.5	0.00537956	0.0118437	
0.6	0.00818899	0.0206594	
0.7	0.0124745	0.0348073	
0.8	0.0190509	0.0575374	
0.9	0.0291965	0.0941673	
1	0.0449373	0.153444	

Runge error by Romberg:

X	Y err	Y' err	
0	0	0	
0.1	2.22429e-08	1.96323e-08	
0.2	1.51837e-07	1.54135e-07	
0.3	4.4219e-07	5.88298e-07	
0.4	1.01048e-06	1.66106e-06	
0.5	2.08073e-06	4.0333e-06	
0.6	4.06796e-06	9.00423e-06	
0.7	7.73696e-06	1.91254e-05	
0.8	1.45011e-05	3.94051e-05	
0.9	2.69877e-05	7.96887e-05	
1	5.01152e-05	0.000159391	

Adams error by Romberg:

X	Y err	Y' err	
0	0	0	
0.1	2.22429e-08	1.96323e-08	
0.2	-2.39804e-07	-4.16526e-06	
0.3	-2.50624e-06	-1.55901e-05	
0.4	2.07612e-05	0.000145111	
0.5	0.000102224	0.000448712	
0.6	0.000241138	0.000981858	
0.7	0.000497323	0.0019368	
0.8	0.000952972	0.00367901	
0.9	0.00175911	0.00685524	
1	0.00318218	0.0126603	

Рис. 2: Вывод программы в консоли

3 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <functional>
#include <cmath>
using namespace std;

double F(double x, double y1, double y2)
{
    return 4 * x * y2 - y1 * (4 * x * x - 3) + pow(M_E, pow(x, 2));
}

double Fprec(double x)
{
    return (pow(M_E, x) + pow(M_E, -x) - 1) * pow(M_E, pow(x, 2));
}

vector<vector<double>> Euler(double x0, double y1_0, double y2_0, double h, int n)
{
    vector<double> x(n);
    x[0] = x0;

    vector<double> y1(n);
    y1[0] = y1_0;

    vector<double> y2(n);
    y2[0] = y2_0;

    for (int i = 1; i < n; ++i)
    {
        x[i] = x[i - 1] + h;
        y1[i] = y1[i - 1] + h * y2[i - 1];
        y2[i] = y2[i - 1] + h * F(x[i - 1], y1[i - 1], y2[i - 1]);
    }

    return vector<vector<double>>{x, y1, y2};
}


vector<vector<double>> Runge(double x0, double y1_0, double y2_0, double h, int n)
{
    vector<double> x(n);
    x[0] = x0;

    vector<double> y1(n);
    y1[0] = y1_0;

    vector<double> y2(n);
```

```
48      y2[0] = y2_0;
49
50
51      for (int i = 1; i < n; ++i)
52      {
53          x[i] = x[i - 1] + h;
54
55          double k1_y1 = h * y2[i - 1];
56          double k1_y2 = h * F(x[i - 1], y1[i - 1], y2[i - 1]);
57
58          double k2_y1 = h * (y2[i - 1] + 0.5 * k1_y2);
59          double k2_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k1_y1, y2[i - 1] +
                0.5 * k1_y2);
60
61          double k3_y1 = h * (y2[i - 1] + 0.5 * k2_y2);
62          double k3_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k2_y1, y2[i - 1] +
                0.5 * k2_y2);
63
64          double k4_y1 = h * (y2[i - 1] + k3_y2);
65          double k4_y2 = h * F(x[i - 1] + h, y1[i - 1] + k3_y1, y2[i - 1] + k3_y2);
66
67          y1[i] = y1[i - 1] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
68          y2[i] = y2[i - 1] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
69      }
70
71      return vector<vector<double>>{x, y1, y2};
72  }
73
74  vector<vector<double>> Adam(double x0, double y1_0, double y2_0, double h, int n)
75  {
76      vector<double> x(n);
77      x[0] = x0;
78
79      vector<double> y1(n);
80      y1[0] = y1_0;
81
82      vector<double> y2(n);
83      y2[0] = y2_0;
84
85      for (int i = 1; i < 4; ++i)
86      {
87          x[i] = x[i - 1] + h;
88
89          double k1_y1 = h * y2[i - 1];
90          double k1_y2 = h * F(x[i - 1], y1[i - 1], y2[i - 1]);
91
92          double k2_y1 = h * (y2[i - 1] + 0.5 * k1_y2);
93          double k2_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k1_y1, y2[i - 1] +
                0.5 * k1_y2);
```

```
94
95          double k3_y1 = h * (y2[i - 1] + 0.5 * k2_y2);
96          double k3_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k2_y1, y2[i - 1] +
                0.5 * k2_y2);
97
98          double k4_y1 = h * (y2[i - 1] + k3_y2);
99          double k4_y2 = h * F(x[i - 1] + h, y1[i - 1] + k3_y1, y2[i - 1] + k3_y2);
100
101         y1[i] = y1[i - 1] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
102         y2[i] = y2[i - 1] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
103     }
104
105     for (int i = 4; i < n; ++i)
106     {
107         x[i] = x[i - 1] + h;
108
109         y1[i] = y1[i - 1] + h / 24 * (55 * y2[i - 1] - 59 * y2[i - 2] + 37 * y2[i - 3]
                - 9 * y2[i - 4]);
110         y2[i] = y2[i - 1] + h / 24 * (55 * F(x[i - 1], y1[i - 1], y2[i - 1]) - 59 * F(x
                [i - 2], y1[i - 2], y2[i - 2]) + 37 * F(x[i - 3], y1[i - 3], y2[i - 3]) - 9
                 * F(x[i - 4], y1[i - 4], y2[i - 4]));
111     }
112
113     return vector<vector<double>>{x, y1, y2};
114 }
115
116 void Romberg(const function<vector<vector<double>>(double, double, double, double, int
        )> &Func, double x0, double y1_0, double y2_0, double h, int n)
117 {
118     vector<vector<double>> dat1 = Func(x0, y1_0, y2_0, h, n);
119     vector<vector<double>> dat2 = Func(x0, y1_0, y2_0, h / 2, 2 * n);
120
121     vector<double> y1_h, y2_h, y1_h2, y2_h2;
122
123     y1_h = dat1[1];
124     y2_h = dat1[2];
125
126     y1_h2 = dat2[1];
127     y2_h2 = dat2[2];
128
129     cout << "X\tY err\tY' err" << endl;
130     for (int i = 0; i < n; ++i)
131     {
132         double x_i = x0 + i * h;
133         double error_y1 = (y1_h2[2 * i] - y1_h[i]) / (pow(2, 4) - 1);
134         double error_y2 = (y2_h2[2 * i] - y2_h[i]) / (pow(2, 4) - 1);
135         cout << x_i << "\t" << error_y1 << "\t" << error_y2 << endl;
136     }
137 }
```

```
138 
139 void Error(vector<vector<double>>& data)
140 {
141     cout << "X \t Y \t Yprec \t Error" << endl;
142 
143     for (size_t i = 0; i < data[0].size(); ++i)
144     {
145         double Yprec = Fprec(data[0][i]);
146         double err = abs(data[1][i] - Yprec);
147         cout << data[0][i] << " \t " << data[1][i] << " \t " << Yprec << " \t" << err
                << endl;
148     }
149 }
150 
151 int main() {
152     double x = 0;
153     double y1 = 1;
154     double y2 = 0;
155     double h = 0.1;
156 
157     int n = 11;
158 
159     vector<vector<double>> euler = Euler(x, y1, y2, h, n);
160     vector<vector<double>> runge = Runge(x, y1, y2, h, n);
161     vector<vector<double>> adam = Adam(x, y1, y2, h, n);
162 
163     cout << "\nEuler:" << "\n";
164     Error(euler);
165 
166     cout << "\nEuler error by Romberg:" << "\n";
167     Romberg(Euler, x, y1, y2, h, n);
168 
169     cout << "\nRunge:" << "\n";
170     Error(runge);
171 
172     cout << "\nRunge error by Romberg:" << "\n";
173     Romberg(Runge, x, y1, y2, h, n);
174 
175     cout << "\nAdam:" << "\n";
176     Error(adam);
177 
178     cout << "\nAdams error by Romberg:" << "\n";
179     Romberg(Adam, x, y1, y2, h, n);
180 
181     return 0;
182 }
```

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: 8

(2x+1) y″+4xy′-4y=0, y′ (-2)+2y(-2)= -9, y′(0)=1	$y(x) = 3x + e^{-2x}$

Рис. 3: Входные данные

5 Результаты работы

```
x       y
-2      48.5982
-1.8    31.1982
-1.6    19.7325
-1.4    12.2446
-1.2    7.42318
-1      4.38906
-0.8    2.55303
-0.6    1.52012
-0.4    1.02554
-0.2    0.891825
-2.77556e-16    1

Shooting:
s               f(b, y, s)      P(s)
1               30.4482         29.4482
-120.158        1       4.52971e-14

x       y
-2      48.5982
-1.8    29.0656
-1.6    16.4754
-1.4    8.53955
-1.2    3.72388
-1      1

Errorr:
X       Y err           Y' err
-2      0       0
-1.8    8.17842e-05     -0.000163568
-1.6    0.00010965      -0.0002193
-1.4    0.000110258     -0.000220516
-1.2    9.85502e-05     -0.0001971
-1      8.25804e-05     -0.000165161
```

```
EndDiff:
x       y
-2      48.5982
-2      176.655
-1.8    276.446
-1.6    328.604
-1.4    318.626
-1.2    240.674
-1      102.21
-0.8    -68.9612
-0.6    -207.031
-0.4    -80.1012
-0.2    -42.8846
0       -19.391

EndDiff h/2:
x       y
-2      48.5982
-2      44.108
-1.9    37.8535
-1.8    30.0307
-1.7    20.9144
-1.6    10.8568
-1.5    0.286094
-1.4    -10.299
-1.3    -20.3347
-1.2    -29.2013
-1.1    -36.2323
-1      -40.727
-0.9    -41.9636
-0.8    -39.2137
-0.7    -31.7581
-0.6    -18.9036
-0.5    0
-0.4    0.095599
-0.3    0.217966
-0.2    0.368668
-0.1    0.548864
0       0.759247

Error:
x       y
-2      128.056
-1.8    245.248
-1.6    308.872
-1.4    306.381
-1.2    233.251
-1      97.821
-0.8    71.5143
-0.6    208.551
-0.4    81.1267
-0.2    43.7764
0       20.391
```

Рис. 4: Вывод программы в консоли

6 Исходный код

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <functional>
using namespace std;

std::vector<double> triade(const vector<vector<double>>& matrix, const std::vector<
    double>& b) {
    size_t n = matrix.size();

    std::vector<double> C(n, 0);
    std::vector<double> D(n, 0);
    std::vector<double> x(n);

    C[0] = matrix[0][1] / matrix[0][0];
    D[0] = b[0] / matrix[0][0];

    for (size_t i = 1; i < n; ++i) {
        double m = 1 / (matrix[i][i] - matrix[i][i - 1] * C[i - 1]);
        C[i] = i < n - 1 ? matrix[i][i + 1] * m : 0;
        D[i] = (b[i] - matrix[i][i - 1] * D[i - 1]) * m;
    }

    x[n - 1] = D[n - 1];

    for (int i = n - 2; i >= 0; --i) {
        x[i] = D[i] - C[i] * x[i + 1];
    }

    return x;
}

double F(double x, double y1, double y2)
{
    return (4 * y1 - 4 * x * y2) / (2 * x + 1);
}

double Fprec(double x)
{
    return 3 * x + pow(M_E, (-2) * x);
}

vector<vector<double>> Runge(double x0, double y1_0, double y2_0, double h, int n)
{
    vector<double> x(n);
    x[0] = x0;
```

```
47
48      vector<double> y1(n);
49      y1[0] = y1_0;
50
51      vector<double> y2(n);
52      y2[0] = y2_0;
53
54
55      for (int i = 1; i < n; ++i)
56      {
57          x[i] = x[i - 1] + h;
58
59          double k1_y1 = h * y2[i - 1];
60          double k1_y2 = h * F(x[i - 1], y1[i - 1], y2[i - 1]);
61
62          double k2_y1 = h * (y2[i - 1] + 0.5 * k1_y2);
63          double k2_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k1_y1, y2[i - 1] +
                0.5 * k1_y2);
64
65          double k3_y1 = h * (y2[i - 1] + 0.5 * k2_y2);
66          double k3_y2 = h * F(x[i - 1] + 0.5 * h, y1[i - 1] + 0.5 * k2_y1, y2[i - 1] +
                0.5 * k2_y2);
67
68          double k4_y1 = h * (y2[i - 1] + k3_y2);
69          double k4_y2 = h * F(x[i - 1] + h, y1[i - 1] + k3_y1, y2[i - 1] + k3_y2);
70
71          y1[i] = y1[i - 1] + (k1_y1 + 2 * k2_y1 + 2 * k3_y1 + k4_y1) / 6;
72          y2[i] = y2[i - 1] + (k1_y2 + 2 * k2_y2 + 2 * k3_y2 + k4_y2) / 6;
73      }
74
75      return vector<vector<double>>{x, y1, y2};
76  }
77
78  void Romberg(const function<vector<vector<double>>(double, double, double, double, int
        )> &Func, double x0, double y1_0, double y2_0, double h, int n)
79  {
80      vector<vector<double>> dat1 = Func(x0, y1_0, y2_0, h, n);
81      vector<vector<double>> dat2 = Func(x0, y1_0, y2_0, h / 2, 2 * n);
82
83      vector<double> y1_h, y2_h, y1_h2, y2_h2;
84
85      y1_h = dat1[1];
86      y2_h = dat1[2];
87
88      y1_h2 = dat2[1];
89      y2_h2 = dat2[2];
90
91      cout << "X\tY err\t\tY' err" << endl;
92      for (int i = 0; i < n; ++i)
```

```
93      {
94          double x_i = x0 + i * h;
95          double error_y1 = (y1_h2[2 * i] - y1_h[i]) / (pow(2, 4) - 1);
96          double error_y2 = (y2_h2[2 * i] - y2_h[i]) / (pow(2, 4) - 1);
97          cout << x_i << "\t" << error_y1 << "\t" << error_y2 << endl;
98      }
99  }
100
101 void BangBang(const function<double(double, double, double)>& f, double a, double b,
        double alpha, double beta, double eps, double h, int n)
102 {
103     vector<vector<double>> result;
104     vector<double> y_trial(n);
105     vector<double> s_values;
106     double s0 = 0;
107     double s1 = 1.0;
108     double y_b0, y_b1, s_new;
109
110     auto result0 = Runge(a, alpha, s0, h, n);
111     y_b0 = result0[1].back();
112     auto result1 = Runge(a, alpha, s1, h, n);
113     y_b1 = result1[1].back();
114
115
116     cout << "s\t\tf(b, y, s)\tP(s)" << endl;
117
118     while (abs(y_b1 - beta) > eps)
119     {
120         s_new = s1 - (y_b1 - beta) * (s1 - s0) / (y_b1 - y_b0);
121         s_values.push_back(s_new);
122         cout << s1 << " \t\t" << y_b1 << "\t\t" << abs(y_b1 - beta) << endl;
123         s0 = s1;
124         y_b0 = y_b1;
125         s1 = s_new;
126         result = Runge(a, alpha, s1, h, n);
127         y_b1 = result[1].back();
128     }
129     cout << s1 << "\t" << y_b1 << "\t" << abs(y_b1 - beta) << endl;
130
131     cout << "\nx\ty" << endl;
132
133     for (int i = 0; i < n; ++i)
134     {
135         cout << result[0][i] << "\t" << result[1][i] << endl;
136     }
137 }
138
139
140 double p(double x)
```

```
{
    return 0;
}

double q(double x)
{
    return (-4 + 4 * x) / (2 * x + 1);
}

double f(double x)
{
    return 0.0;
};


void EndDiff(double a, double b, double y0, double y1, double h, vector<double>& x,
    vector<double>& y)
{
    int n = static_cast<int>((b - a) / h) + 1;

    x.resize(n);

    y.resize(n);

    vector<double> rhs(n);

    for (int i = 0; i < n; ++i) {
        x[i] = a + i * h;
    }

    rhs[0] = h * h * f(x[0]) - (1 - p(x[0]) * h / 2) * y0;

    rhs[n - 1] = h * h * f(x[n - 1]) - (1 + p(x[n - 1]) * h / 2) * y1;

    for (int i = 1; i < n - 1; ++i)
    {
        rhs[i] = h * h * f(x[i]);
    }

    vector<vector<double>> A(n, vector<double>(n));
    for (int i = 0; i < n; ++i) {
        A[i][i] = -2 + h * h * q(x[i]);
        if (i > 0) A[i][i - 1] = 1 - p(x[i]) * h / 2;
        if (i < n - 1) A[i][i + 1] = (1 + p(x[i]) * h / 2);
    }
    y = triade(A, rhs);
    cout << "x\ty" << endl;
    cout << a << "\t" << y0 << endl;

```

```
189     for (int i = 0; i < n; ++i)
190     {
191         cout << x[i] << "\t" << y[i] << endl;
192     }
193 }
194 
195 void PrecF(double a, double b, double h)
196 {
197     vector<double> x, y;
198 
199     for (double xi = a; xi <= b; xi += h)
200     {
201         x.push_back(xi);
202         y.push_back(Fprec(xi));
203     }
204 
205     cout << "x\ty" << endl;
206 
207     for (size_t i = 0; i < x.size(); ++i)
208     {
209         cout << x[i] << "\t" << y[i] << endl;
210     }
211 }
212 
213 double RungeEr(const vector<double>& y_h, const vector<double>& y_h2, double r) {
214     double error = 0.0;
215     for (size_t i = 0; i < y_h.size(); ++i) {
216         error = max(error, abs(y_h2[2 * i] - y_h[i]) / (pow(2, r) - 1));
217     }
218     return error;
219 }
220 
221 int main()
222 {
223     PrecF(-2, 0, 0.2);
224 
225     double x1 = -2, x2 = 0, y1 = 48.5982, y2 = 1;
226     double e = 0.00001;
227     double h = 0.2;
228 
229 
230     cout << "\nShooting:" << "\n";
231     BangBang(F, x1, x2, y1, y2, e, h, 6);
232 
233 
234     cout << "\nErrorr:" << "\n";
235     Romberg(Runge, x1, y1, 0, h, 6);
236 
237     vector<double> x_h, y_h, x_h2, y_h2;
```

```
238
239     cout << "\nEndDiff:" << "\n";
240     EndDiff(x1, x2, y1, y2, h, x_h, y_h);
241
242     cout << "\nEndDiff h/2:" << "\n";
243     EndDiff(x1, x2, y1, y2, h / 2, x_h2, y_h2);
244
245     double error = RungeEr(y_h, y_h2, 2);
246
247     cout << "\nError:" << endl;
248
249     cout << "x\ty" << endl;
250
251     double m = 0.0;
252     for (size_t i = 0; i < x_h.size(); ++i)
253     {
254         double ex = Fprec(x_h[i]);
255         double err = abs(y_h[i] - ex);
256         m = max(m, err);
257         cout << x_h[i] << "\t" << err << endl;
258     }
259
260     return 0;
261 }
```