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: Тысячный В.В.
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: 25

25	$(x-2)^2 y''-(x-2)y'-3y=0,$ $y(3)=2,$ $y'(3)=2,$ $x\in[3,4], h=0.1$	$y=(x-2)^3+\frac{1}{x-2}$

Рис. 1: Входные данные

2 Результаты работы

```
PS C:\Users\Tysyachnyj V\Desktop\NM> & 'c:\Users\Tysyachnyj V\.vscode\extensions\ms-vscode.cpptools
rosoft-MIEngine-Out-uh45cflr.yre' '--stderr=Microsoft-MIEngine-Error-ly4idbvp.vxa' '--pid=Microsoft-
3.000000 3.100000 3.200000 3.300000 3.400000 3.500000 3.600000 3.700000 3.800000 3.900000 4.000000
2.000000 2.200000 2.480000 2.840000 3.281667 3.807722 4.421582 5.127136 5.928602 6.830437 7.837267
2.000000 2.240091 2.561332 2.966227 3.458279 4.041656 4.720985 5.501215 6.387529 7.385283 8.499960
2.000000 2.240091 2.561332 2.966227 3.458430 4.041726 4.721012 5.501106 6.387253 7.384801 8.499252
0.000000 0.000000 0.000002 0.000004 0.000145 0.000059 0.000012 0.000129 0.000303 0.000515 0.000748
0.000000 0.000001 0.000005 0.000011 0.000020 0.000031
PS C:\Users\Tysyachnyj V\Desktop\NM>
```

Рис. 2: Вывод программы в консоли

3 Исходный код

```cpp
#include <iostream>
#include<iomanip>
#include <cmath>

using namespace std;

//
double f(double x, double y, double z){
    return z / (x - 2) + 3 * y / pow(x - 2, 2);
}

double y(double x){
    return pow(x - 2, 3) + 1 / (x - 2);
}

//
double* Euler(double a, double b, double h, double y0, double z0){
    int n = (b - a) / h;
    double x[n + 1];
    double* y = new double[n + 1];
    double z[n + 1];
    x[0] = a;
    y[0] = y0;
    z[0] = z0;
    for (int i = 1; i <= n; ++i){
        y[i] = y[i - 1] + h * z[i - 1];
        z[i] = z[i - 1] + h * f(x[i - 1], y[i - 1], z[i - 1]);
        x[i] = x[i - 1] + h;
    }
    return y;
}

//     -
double K(double x, double y, double z, double h, int i);

double L(double x, double y, double z, double h, int i){
    if (i == 0){
        return h * z;
    }
    else if (i == 3){
        return h * (z + K(x, y, z, h, i - 1));
    }
    else{
        return h * (z + K(x, y, z, h, i - 1) / 2);
    }
}

```

```
48 double K(double x, double y, double z, double h, int i){
49     if (i == 0){
50         return h * f(x, y, z);
51     }
52     else if (i == 3){
53         return h * f(x + h, y + L(x, y, z, h, i - 1), z + K(x, y, z, h, i - 1));
54     }
55     else{
56         return h * f(x + h / 2, y + L(x, y, z, h, i - 1) / 2, z + K(x, y, z, h, i - 1)
               / 2);
57     }
58 }
59
60 double delta_z(double x, double y, double z, double h){
61     double d = 0;
62     for (int i = 0; i < 4; ++i){
63         if (i == 0 or i == 3){
64             d += K(x, y, z, h, i);
65         }
66         else{
67             d += 2 * K(x, y, z, h, i);
68         }
69     }
70     return d / 6;
71 }
72
73 double delta_y(double x, double y, double z, double h){
74     double d = 0;
75     for (int i = 0; i < 4; ++i){
76         if (i == 0 or i == 3){
77             d += L(x, y, z, h, i);
78         }
79         else{
80             d += 2 *L(x, y, z, h, i);
81         }
82     }
83     return d / 6;
84 }
85
86 //  - 4
87 pair<double*, double*> Runge_Kutta_4(double a, double b, double h, double y0, double
       z0){
88     int n = (b - a) / h;
89     double x[n + 1];
90     double* y = new double[n + 1];
91     double* z = new double[n + 1];
92     x[0] = a;
93     y[0] = y0;
94     z[0] = z0;
```

```
95      for (int i = 1; i <= n; ++i){
96          y[i] = y[i - 1] + delta_y(x[i - 1], y[i - 1], z[i - 1], h);
97          z[i] = z[i - 1] + delta_z(x[i - 1], y[i - 1], z[i - 1], h);
98          x[i] = x[i - 1] + h;
99      }
100     return pair<double*, double*>(y, z);
101 }
102
103
104 //
105 double* Adams(double a, double b, double h, double y0, double z0){
106     int n = (b - a) / h;
107     double x[n + 1];
108     double* y = new double[n + 1];
109     double* z = new double[n + 1];
110     pair<double*, double*> yz = Runge_Kutta_4(a, b, h, y0, z0);
111     for (int i = 0; i < 4; ++i){
112         x[i] = a + h * i;
113         y[i] = yz.first[i];
114         z[i] = yz.second[i];
115     }
116     for (int i = 4; i <= n; ++i){
117         y[i] = y[i - 1] + h / 24 * (55 * z[i - 1] - 59 * z[i - 2] + 37 * z[i - 3] - 9 *
                z[i - 4]);
118         z[i] = z[i - 1] + h / 24 * (55 * f(x[i - 1], y[i - 1], z[i - 1]) - 59 * f(x[i -
                2], y[i - 2], z[i - 2]) + 37 * f(x[i - 3], y[i - 3], z[i - 3]) - 9 * f(x[i
                - 4], y[i - 4], z[i - 4]));
119         x[i] = x[i - 1] + h;
120     }
121     return y;
122 }
123
124 double* Runge_Romberg(double Y_1[], double Y_2[], int n){
125     double* R = new double[n];
126     for (int i = 0; i < n; ++i){
127         R[i] = (Y_1[i * 2] - Y_2[i]) / (pow(2, 4) - 1);
128     }
129     return R;
130 }
131
132 double* Error(double Y_t[], double Y[], int n){
133     double* eps = new double[n];
134     for (int i = 0; i < n; ++i){
135         eps[i] = abs(Y_t[i] - Y[i]);
136     }
137     return eps;
138 }
139
140
```

```cpp
int main(){
    double a = 3;
    double b = 4;
    double y0 = 2;
    double z0 = 2;
    double h = 0.1;

    double* Ans[5];
    int n = (b - a) / h;

    double* X = new double[n + 1];
    for (int i = 0; i <= n; ++i){
        X[i] = a + h * i;
    }
    Ans[0] = X;

    //
    double* Y_E = Euler(a, b, h, y0, z0);
    Ans[1] = Y_E;

    //  - 4
    auto [Y_R, Z1] = Runge_Kutta_4(a, b, h, y0, z0);
    auto [Y_R2, Z2] = Runge_Kutta_4(a, b, h * 2, y0, z0);
    Ans[2] = Y_R;

    //
    double* Y_A = Adams(a, b, h, y0, z0);
    Ans[3] = Y_A;

    double Y_t[n + 1];
    for (int i = 0; i <= n; ++i){
        Y_t[i] = y(X[i]);
    }

    //
    double* eps = Error(Y_t, Y_A, n + 1);
    Ans[4] = eps;

    for (int i = 0; i < 5; ++i){
        for (int j = 0; j <= n; ++j){
            cout << fixed << Ans[i][j] << " ";
        }
        cout << "\n";
    }

    //  -
    double* RR = Runge_Romberg(Y_R, Y_R2, n / 2 + 1);
    for (int i = 0; i <= n / 2; ++i){
        cout << RR[i] << " ";
```

```
190 ||      }
191 ||
192 || }
```

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: 25

$$\begin{cases} 2x(x+2)y'' + (2-x)y' + 6xy = 0 \\ y(1) = 0 \\ y'(4) + y(4) = 21/4 \end{cases}$$

$y(x) = \sqrt{x} + x - 2$

5 Результаты работы

```
PS C:\Users\Tysyachnyj V\Desktop\NM>  & 'c:\Users\Tysyachnyj V\.vscode\extensions\ms-vscode.cpptool
rosoft-MIEngine-Out-aehorrpo.2pw' '--stderr=Microsoft-MIEngine-Error-debwvcc3.zuy' '--pid=Microsoft
1.000000 1.300000 1.600000 1.900000 2.200000 2.500000 2.800000 3.100000 3.400000 3.700000 4.000000
0.000000 0.440172 0.864906 1.278400 1.683235 2.081134 2.473316 2.860678 3.243906 3.623536 3.999998
0.000000 0.461741 0.907533 1.333142 1.743978 2.143404 2.533673 2.916383 3.292718 3.663580 4.029677
0.000000 0.000004 0.000005 0.000005 0.000005 0.000005 0.000004 0.000004 0.000003 0.000003 0.000002
0.000000 0.021566 0.042622 0.054737 0.060739 0.062265 0.060353 0.055702 0.048809 0.040041 0.029677
PS C:\Users\Tysyachnyj V\Desktop\NM>
```

Рис. 3: Вывод программы в консоли

6 Исходный код

```
#include <iostream>
#include<iomanip>
#include <cmath>

using namespace std;

class matrix
{
private:
   double **a;
   int n, m;
public:
   //
   matrix (){
      a = 0;
      n = 0;
      m = 0;
   }

   //  NxM,  E,  ,
   matrix (int N, int M, bool E = 0){
      n = N;
      m = M;
      a = new double *[n];
      for (int i = 0; i < n; ++ i){
         a[i] = new double[m];
         for (int j = 0; j < m; ++ j){
            a[i][j] = (i == j) * E;
         }
      }
   }

   //
   int get_n_rows(){
      return n;
   }
   int get_n_cols(){
      return m;
   }

   double* operator [] (int index){
      return getRow (index);
   }


   //
   double* getRow(int index){
```

```
48      if (index >= 0 && index < n){
49         return a[index];
50      }
51      return 0;
52   }
53
54   //
55   double* getColumn(int index){
56      if (index < 0 || index >= m){
57         return 0;
58      }
59      double * c = new double [n];
60      for (int i = 0; i < n; ++ i){
61         c[i] = a[i][index];
62      }
63      return c;
64   }
65
66   //
67   void swapRows (int index1, int index2){
68      if (index1 < 0 || index2 < 0 || index1 >= n || index2 >= n){
69         return ;
70      }
71      for (int i = 0; i < m; ++ i){
72         swap (a[index1][i], a[index2][i]);
73      }
74   }
75};
76
77//
78void printMatrix (matrix & a){
79   for (int i = 0; i < a.get_n_rows (); ++ i){
80      for (int j = 0; j < a.get_n_cols (); ++ j){
81         printf ("%5.3lf ", a[i][j]);
82      }
83      puts ("");
84   }
85}
86
87//
88double f(double x, double y, double z){
89    return (- y - (2 - x) * z) / (2 * x * (x + 2));
90}
91
92double y(double x){
93    return pow(abs(x), 0.5) + x - 2;
94}
95
96double Phi(double y, double z){
```

```
97      return z + y - 5.25;
98  }
99
100 double p(double x){
101     return (2 - x) / 2 / x / (x + 2);
102 }
103
104 double q(double x){
105     return 0.5 / x / (x + 2);
106 }
107
108 //      -
109 double K(double x, double y, double z, double h, int i);
110
111 double L(double x, double y, double z, double h, int i){
112     if (i == 0){
113         return h * z;
114     }
115     else if (i == 3){
116         return h * (z + K(x, y, z, h, i - 1));
117     }
118     else{
119         return h * (z + K(x, y, z, h, i - 1) / 2);
120     }
121 }
122
123 double K(double x, double y, double z, double h, int i){
124     if (i == 0){
125         return h * f(x, y, z);
126     }
127     else if (i == 3){
128         return h * f(x + h, y + L(x, y, z, h, i - 1), z + K(x, y, z, h, i - 1));
129     }
130     else{
131         return h * f(x + h / 2, y + L(x, y, z, h, i - 1) / 2, z + K(x, y, z, h, i - 1)
                / 2);
132     }
133 }
134
135 double delta_z(double x, double y, double z, double h){
136     double d = 0;
137     for (int i = 0; i < 4; ++i){
138         if (i == 0 or i == 3){
139             d += K(x, y, z, h, i);
140         }
141         else{
142             d += 2 * K(x, y, z, h, i);
143         }
144     }
```

```
145     return d / 6;
146 }
147 
148 double delta_y(double x, double y, double z, double h){
149     double d = 0;
150     for (int i = 0; i < 4; ++i){
151         if (i == 0 or i == 3){
152             d += L(x, y, z, h, i);
153         }
154         else{
155             d += 2 *L(x, y, z, h, i);
156         }
157     }
158     return d / 6;
159 }
160 
161 //  - 4
162 pair<double*, double*> Runge_Kutta_4(double a, double b, double h, double y0, double
        z0){
163     int n = (b - a) / h;
164     double x[n + 1];
165     double* y = new double[n + 1];
166     double* z = new double[n + 1];
167     x[0] = a;
168     y[0] = y0;
169     z[0] = z0;
170     for (int i = 1; i <= n; ++i){
171         y[i] = y[i - 1] + delta_y(x[i - 1], y[i - 1], z[i - 1], h);
172         z[i] = z[i - 1] + delta_z(x[i - 1], y[i - 1], z[i - 1], h);
173         x[i] = x[i - 1] + h;
174     }
175     return pair<double*, double*>(y, z);
176 }
177 
178 //
179 double* shooting(double a, double b, double h, double y0, double eps){
180     double nu[3] = {1, 0.8, 0};
181     double phi[3] = {0, 0, 0};
182     int n = (b - a) / h;
183 
184     pair<double*, double*> R;
185     R = Runge_Kutta_4(a, b, h, y0, nu[0]);
186     phi[0] = Phi(R.first[n], R.second[n]);
187     R = Runge_Kutta_4(a, b, h, y0, nu[1]);
188     phi[1] = Phi(R.first[n], R.second[n]);
189     phi[2] = phi[1];
190     while (abs(phi[1]) > eps){
191         nu[2] = nu[1] - (nu[1] - nu[0]) / (phi[1] - phi[0]) * phi[1];
192         R = Runge_Kutta_4(a, b, h, y0, nu[2]);
```

```
193         phi[2] = Phi(R.first[n], R.second[n]);
194         nu[0] = nu[1];
195         nu[1] = nu[2];
196         phi[0] = phi[1];
197         phi[1] = phi[2];
198     }
199     return R.first;
200 }
201 
202 // -
203 double* finite_difference(double a, double b, double h, double y0, double eps){
204     int n = (b - a) / h;
205     double X[n + 1];
206     for (int i = 0; i <= n; ++i){
207         X[i] = a + h * i;
208     }
209     matrix A = matrix(n, 3);
210     double B[n];
211     A[0][0] = 0;
212     A[0][1] = -2 + pow(h, 2) * q(X[1]);
213     A[0][2] = 1 + p(X[1]) * h / 2;
214     B[0] = 0;
215     for (int i = 1; i < n - 1; ++i){
216         A[i][0] = 1 - p(X[i]) * h / 2;
217         A[i][1] = -2 + pow(h, 2) * q(X[i]);
218         A[i][2] = 1 + p(X[i]) * h / 2;
219         B[i] = 0;
220     }
221     A[n - 1][0] = -1;
222     A[n - 1][1] = 1 + h;
223     A[n - 1][2] = 0;
224     B[n - 1] = 5.25 * h;
225 
226     double P[n], Q[n];
227     double* Y = new double[n + 1];
228     P[0] = -A[0][2] / A[0][1];
229     Q[0] = B[0] / A[0][1];
230     for (int i = 1; i < n; ++i){
231         P[i] = -A[i][2] / (A[i][1] + A[i][0] * P[i - 1]);
232         Q[i] = (B[i] - A[i][0] * Q[i - 1]) / (A[i][1] + A[i][0] * P[i - 1]);
233     }
234 
235     Y[n] = Q[n - 1];
236     for (int i = n - 2; i > -1; --i){
237         Y[i + 1] = P[i] * Y[i + 2] + Q[i];
238     }
239     Y[0] = 0;
240     return Y;
241 }
```

```
242
243 double* Error(double Y_t[], double Y[], int n){
244     double* eps = new double[n];
245     for (int i = 0; i < n; ++i){
246         eps[i] = abs(Y_t[i] - Y[i]);
247     }
248     return eps;
249 }
250
251
252 int main(){
253     double a = 1;
254     double b = 4;
255     double y0 = 0;
256     double h = 0.3;
257     double eps = 0.001;
258
259     double* Ans[5];
260     int n = (b - a) / h;
261
262     double* X = new double[n + 1];
263     for (int i = 0; i <= n; ++i){
264         X[i] = a + h * i;
265     }
266     Ans[0] = X;
267
268     //
269     double* Y_S = shooting(a, b, h, y0, eps);
270     Ans[1] = Y_S;
271
272     // -
273     double* Y_FD = finite_difference(a, b, h, y0, eps);
274     Ans[2] = Y_FD;
275
276     double Y_t[n + 1];
277     for (int i = 0; i <= n; ++i){
278         Y_t[i] = y(X[i]);
279     }
280
281     //
282     double* err = Error(Y_t, Y_S, n + 1);
283     Ans[3] = err;
284
285     //
286     err = Error(Y_t, Y_FD, n + 1);
287     Ans[4] = err;
288
289     for (int i = 0; i < 5; ++i){
290         for (int j = 0; j <= n; ++j){
```

```
291             cout << fixed << Ans[i][j] << " ";
292         }
293         cout << "\n";
294     }
295 }
```