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: В. В. Хрушкова
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: 18

27	$x^2y''-2xy'+(x^2+2)y=0,$ $y(\pi/2)=\pi/2,$ $y'(\pi/2)=1-\pi/2,$ $x\in[\pi/2,\pi/2+1],h=0.1$	$y=x\cos x+x\sin x$

Рис. 1: Входные данные

2 Результаты работы

```
1.570796 1.670796 1.770796 1.870796 1.970796 2.070796 2.170796 2.270796 2.370796 2.470796 2.570796
1.570796 1.513717 1.443947 1.360493 1.262468 1.149117 1.019839 0.874210 0.712001 0.533200 0.338023
1.570796 1.507202 1.429936 1.338131 1.231068 1.108198 0.969167 0.813832 0.642278 0.454835 0.252086
1.570796 1.507202 1.429936 1.338131 1.231059 1.108183 0.969147 0.813808 0.642253 0.454812 0.252067
0.000000 0.011554 0.046240 0.103749 0.183300 0.283681 0.403235 0.539894 0.691209 0.854382 1.026311
0.000000 0.000000 0.000000 0.000000 0.000000 0.000001
```

Рис. 2: Вывод программы в консоли

Рис. 3: Вывод программы в консоли

3 Исходный код

```cpp
#include <bits/stdc++.h>

using namespace std;

const double PI = 3.1415926535;

double y(double x){
    return x*cos(x) + x*sin(x);
}

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

double* method_eiler(double start_pos, double end_pos, double precision, double y_0,
    double z_0){
    int n = (end_pos - start_pos) / precision;
    double x[n + 1];
    double* y = new double[n + 1];
    double z[n + 1];
    x[0] = start_pos;
    y[0] = y_0;
    z[0] = z_0;
    for (int i = 1; i <= n; i++){
        y[i] = y[i - 1] + precision * z[i - 1];
        z[i] = z[i - 1] + precision * f(x[i - 1], y[i - 1], z[i - 1]);
        x[i] = x[i - 1] + precision;
    }
    return y;
}


double K(double x, double y, double z, double precision, int i);

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

double K(double x, double y, double z, double precision, int i){
    if (i == 0)
        return precision * f(x, y, z);
    else if (i == 3)
```

```
47          return precision * f(x + precision, y + L(x, y, z, precision, i - 1), z + K(x,
                y, z, precision, i - 1));
48      else
49          return precision * f(x + precision / 2, y + L(x, y, z, precision, i - 1) / 2, z
                 + K(x, y, z, precision, i - 1) / 2);
50  }
51
52  double dz(double x, double y, double z, double precision){
53      double d = 0;
54      for (int i = 0; i < 4; i++){
55          if (i == 0 || i == 3)
56              d += K(x, y, z, precision, i);
57          else
58              d += 2 * K(x, y, z, precision, i);
59      }
60      return d / 6;
61  }
62
63  double dy(double x, double y, double z, double precision){
64      double d = 0;
65      for (int i = 0; i < 4; i++)
66          if (i == 0 || i == 3)
67              d += L(x, y, z, precision, i);
68          else
69              d += 2 *L(x, y, z, precision, i);
70      return d / 6;
71  }
72
73  pair<double*, double*> RK_method(double start_pos, double end_pos, double precision,
        double y_0, double z_0){
74      int n = (end_pos - start_pos) / precision;
75      double x[n + 1];
76      double* y = new double[n + 1];
77      double* z = new double[n + 1];
78      x[0] = start_pos;
79      y[0] = y_0;
80      z[0] = z_0;
81      for (int i = 1; i <= n; i++){
82          y[i] = y[i - 1] + dy(x[i - 1], y[i - 1], z[i - 1], precision);
83          z[i] = z[i - 1] + dz(x[i - 1], y[i - 1], z[i - 1], precision);
84          x[i] = x[i - 1] + precision;
85      }
86      return pair<double*, double*>(y, z);
87  }
88
89  double* method_adams(double start_pos, double end_pos, double precision, double y_0,
        double z_0){
90      int n = (end_pos - start_pos) / precision;
91      double x[n + 1];
```

```
92      double* y = new double[n + 1];
93      double* z = new double[n + 1];
94      pair<double*, double*> yz = RK_method(start_pos, end_pos, precision, y_0, z_0);
95      for (int i = 0; i < 4; i++){
96          x[i] = start_pos + precision * i;
97          y[i] = yz.first[i];
98          z[i] = yz.second[i];
99      }
100     for (int i = 4; i <= n; i++){
101         y[i] = y[i - 1] + precision / 24 * (55 * z[i - 1] - 59 * z[i - 2] + 37 * z[i -
                3] - 9 * z[i - 4]);
102         z[i] = z[i - 1] + precision / 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]));
103         x[i] = x[i - 1] + precision;
104     }
105     return y;
106 }
107
108 double* RR_method(double y1[], double y2[], int n){
109     double* R = new double[n];
110     for (int i = 0; i < n; i++){
111         R[i] = (y1[i * 2] - y2[i]) / (pow(2, 4) - 1);
112     }
113     return R;
114 }
115
116 double* deviation(double yt[], double Y[], int n){
117     double* eps = new double[n];
118     for (int i = 0; i < n; i++)
119         eps[i] = abs(yt[i] - Y[i]);
120     return eps;
121 }
122
123 int main(){
124     double start_pos = PI/2, end_pos = PI/2 + 1, y_0 = PI/2, z_0 = 1 - PI/2, precision
            = 0.1;
125     double* answer[5];
126     int n = (end_pos - start_pos) / precision;
127     double* X = new double[n + 1];
128     for (int i = 0; i <= n; i++)
129         X[i] = start_pos + precision * i;
130     answer[0] = X;
131     double* ye = method_eiler(start_pos, end_pos, precision, y_0, z_0);
132     answer[1] = ye;
133     auto [yr, Z1] = RK_method(start_pos, end_pos, precision, y_0, z_0);
134     auto [yr2, Z2] = RK_method(start_pos, end_pos, precision * 2, y_0, z_0);
135     answer[2] = yr;
136     double* ya = method_adams(start_pos, end_pos, precision, y_0, z_0);
```

```
137     answer[3] = ya;
138     double yt[n + 1];
139     for (int i = 0; i <= n; i++)
140         yt[i] = y(X[i]);
141     double* eps = deviation(yt, ya, n + 1);
142     answer[4] = eps;
143     for (int i = 0; i < 5; i++){
144         for (int j = 0; j <= n; ++j)
145             cout << fixed << answer[i][j] << " ";
146         cout << "\n";
147     }
148     double* RR = RR_method(yr, yr2, n / 2 + 1);
149     for (int i = 0; i <= n / 2; i++)
150         cout << RR[i] << " ";
151 }
```

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: 27

27	$(2x+1)y''+(2x-1)y' -2y=x^2+x,$ $y'(0)=1,$ $y'(1)+y(1)=5$	$y(x) = 2x - 1 + e^{-x} + \frac{x^2+1}{2}$

Рис. 4: Входные данные

5 Результаты работы

```
17 4.194238 4.219282 4.244448 4.269737 4.295148 4.320681 4.346335 4.372112 4.398009 4.424028 4.4
.609536 4.636517 4.663618 4.690838 4.718178 4.745636 4.773214 4.800910 4.828725 4.856658 4.884709 4.912879 4.941166 4.969571 4.998094 5.026734 5.055
492 5.084366 5.113358 5.142467 5.171692 5.201034 5.230492 5.260067 5.289758 5.319565 5.349488 5.379527 5.409681 5.439951 5.470336 5.500836 5.531452
5.562183 5.593028 5.623988 5.655063 5.686253 5.717556 5.748974 5.780507 5.812153 5.843913 5.875787 5.907775 5.939876 5.972091 6.004420 6.036861 6.06
9416 6.102084 6.134864 6.167758 6.200764 6.233883 6.267115 6.300459 6.333915 6.367483 6.401164 6.434957 6.468862 6.502878 6.537007 6.571247 6.605598
 6.640062 6.674636 6.709322 6.744119 6.779028 6.814047 6.849177 6.884419 6.919771 6.955233 6.990807 7.026491 7.062285 7.098190 7.134205 7.170331 7.2
06566 7.242912 7.279368 7.315934 7.352609 7.389394 7.426290 7.463294 7.500409 7.537632 7.574966 7.612408 7.649960 7.687621 7.725391 7.763271 7.80125
9 7.839356 7.877563 7.915878 7.954301 7.992834 8.031475 8.070225 8.109083 8.148050 8.187125 8.226308 8.265600 8.305000 8.344508 8.384124 8.423848 8.
463680 8.503620 8.543668 8.583824 8.624087 8.664459 8.704938 8.745524 8.786218 8.827019 8.867928 8.908945 8.950068 8.991299 9.032637 9.074083 9.1156
35 9.157295 9.199061 9.240935 9.282916 9.325003 9.367197 9.409498 9.451906 9.494421 9.537042 9.579770 9.622604 9.665545 9.708593 9.751746 9.795007 9
.838373 9.881846 9.925426 9.969111 10.012903 10.056801 10.100805 10.144915 10.189131 10.233453 10.277881 10.322415 10.367055 10.411801 10.456652 10.
501609 10.546672 10.591841 10.637115 10.682495 10.727981 10.773572 10.819269 10.865071 10.910979 10.956992 11.003110 11.049334 11.095663 11.142097 1
1.188637 11.235282 11.282032 11.328887 11.375847 11.422913 11.470083 11.517358
```

Рис. 5: Вывод программы в консоли

6 Исходный код

```
#include <bits/stdc++.h>

using namespace std;


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

double y(double x){
    return 2*x - 1 + exp(-x) + (x*x + 1)/2;
}


class double_matrix
{
private:
   double **start_pos;
   int n, m;
public:
    int m_shape(){
        return n;
    }

   int n_shape(){
      return m;
   }

   double* operator [] (int ind){
      return get_n (ind);
   }

   double_matrix(){
      start_pos = 0;
      n = 0;
      m = 0;
   }

   double_matrix(int N, int M, bool E = 0){
        n = N;
        m = M;
        start_pos = new double *[n];
        for (int i = 0; i < n; ++ i){
            start_pos[i] = new double[m];
            for (int j = 0; j < m; ++ j)
                start_pos[i][j] = (i == j) * E;
     }
```

```
48      }
49
50      double* get_n(int ind){
51       if (ind >= 0 && ind < n)
52           return start_pos[ind];
53       return 0;
54      }
55
56      void swapRows (int ind1, int ind2){
57         if (ind1 < 0 || ind2 < 0 || ind1 >= n || ind2 >= n)
58           return ;
59         for (int i = 0; i < m; ++ i)
60            swap (start_pos[ind1][i], start_pos[ind2][i]);
61      }
62  };
63
64  void cout_matrix(double_matrix & start_pos){
65      for (int i = 0; i < start_pos.m_shape (); ++ i){
66         for (int j = 0; j < start_pos.n_shape (); ++ j)
67            printf ("%5.3lf ", start_pos[i][j]);
68         puts ("");
69      }
70  }
71
72  double ugol(double y, double z){
73      return z + y - 5.25;
74  }
75
76  double p(double x){
77      return (2 - x) / 2 / x / (x + 2);
78  }
79
80  double q(double x){
81      return 0.5 / x / (x + 2);
82  }
83
84  double K(double x, double y, double z, double precision, int i);
85
86  double L(double x, double y, double z, double precision, int i){
87      if (i == 0)
88          return precision * z;
89      else if (i == 3)
90          return precision * (z + K(x, y, z, precision, i - 1));
91      else
92          return precision * (z + K(x, y, z, precision, i - 1) / 2);
93  }
94
95  double K(double x, double y, double z, double precision, int i){
96      if (i == 0)
```

```
97          return precision * f(x, y, z);
98      else if (i == 3)
99          return precision * f(x + precision, y + L(x, y, z, precision, i - 1), z + K(x,
                y, z, precision, i - 1));
100     else
101         return precision * f(x + precision / 2, y + L(x, y, z, precision, i - 1) / 2, z
                + K(x, y, z, precision, i - 1) / 2);
102 }
103
104 double dz(double x, double y, double z, double precision){
105     double d = 0;
106     for (int i = 0; i < 4; ++i)
107         if (i == 0 || i == 3)
108             d += K(x, y, z, precision, i);
109         else
110             d += 2 * K(x, y, z, precision, i);
111     return d / 6;
112 }
113
114 double dy(double x, double y, double z, double precision){
115     double d = 0;
116     for (int i = 0; i < 4; ++i)
117         if (i == 0 || i == 3)
118             d += L(x, y, z, precision, i);
119         else
120             d += 2 *L(x, y, z, precision, i);
121     return d / 6;
122 }
123
124 pair<double*, double*> RK_method(double start_pos, double end_pos, double precision,
        double y0, double z0){
125     int n = (end_pos - start_pos) / precision;
126     double x[n + 1];
127     double* y = new double[n + 1];
128     double* z = new double[n + 1];
129     x[0] = start_pos;
130     y[0] = y0;
131     z[0] = z0;
132     for (int i = 1; i <= n; ++i){
133         y[i] = y[i - 1] + dy(x[i - 1], y[i - 1], z[i - 1], precision);
134         z[i] = z[i - 1] + dz(x[i - 1], y[i - 1], z[i - 1], precision);
135         x[i] = x[i - 1] + precision;
136     }
137     return pair<double*, double*>(y, z);
138 }
139
140 double* shooting(double start_pos, double end_pos, double precision, double y0, double
         eps){
141     double nu[3] = {1, 0.8, 0};
```

```
142     double phi[3] = {0, 0, 0};
143     int n = (end_pos - start_pos) / precision;
144     pair<double*, double*> R;
145     R = RK_method(start_pos, end_pos, precision, y0, nu[0]);
146     phi[0] = ugol(R.first[n], R.second[n]);
147     R = RK_method(start_pos, end_pos, precision, y0, nu[1]);
148     phi[1] = ugol(R.first[n], R.second[n]);
149     phi[2] = phi[1];
150     while (abs(phi[1]) > eps){
151         nu[2] = nu[1] - (nu[1] - nu[0]) / (phi[1] - phi[0]) * phi[1];
152         R = RK_method(start_pos, end_pos, precision, y0, nu[2]);
153         phi[2] = ugol(R.first[n], R.second[n]);
154         nu[0] = nu[1];
155         nu[1] = nu[2];
156         phi[0] = phi[1];
157         phi[1] = phi[2];
158     }
159     return R.first;
160 }
161
162 double* f_diff(double start_pos, double end_pos, double precision, double y0, double
        eps){
163     int n = (end_pos - start_pos) / precision;
164     double X[n + 1];
165     for (int i = 0; i <= n; ++i)
166         X[i] = start_pos + precision * i;
167     double_matrix A_matr = double_matrix(n, 3);
168     double B_vect[n];
169     A_matr[0][0] = 0;
170     A_matr[0][1] = -2 + pow(precision, 2) * q(X[1]);
171     A_matr[0][2] = 1 + p(X[1]) * precision / 2;
172     B_vect[0] = 0;
173     for (int i = 1; i < n - 1; ++i){
174         A_matr[i][0] = 1 - p(X[i]) * precision / 2;
175         A_matr[i][1] = -2 + pow(precision, 2) * q(X[i]);
176         A_matr[i][2] = 1 + p(X[i]) * precision / 2;
177         B_vect[i] = 0;
178     }
179     A_matr[n - 1][0] = -1;
180     A_matr[n - 1][1] = 1 + precision;
181     A_matr[n - 1][2] = 0;
182     B_vect[n - 1] = 5.25 * precision;
183     double P[n], Q[n];
184     double* Y = new double[n + 1];
185     P[0] = -A_matr[0][2] / A_matr[0][1];
186     Q[0] = B_vect[0] / A_matr[0][1];
187     for (int i = 1; i < n; ++i){
188         P[i] = -A_matr[i][2] / (A_matr[i][1] + A_matr[i][0] * P[i - 1]);
```

```
189         Q[i] = (B_vect[i] - A_matr[i][0] * Q[i - 1]) / (A_matr[i][1] + A_matr[i][0] * P
                [i - 1]);
190     }
191     Y[n] = Q[n - 1];
192     for (int i = n - 2; i > -1; --i)
193         Y[i + 1] = P[i] * Y[i + 2] + Q[i];
194     Y[0] = 0;
195     return Y;
196 }
197
198 double* deviation(double yt[], double Y[], int n){
199     double* eps = new double[n];
200     for (int i = 0; i < n; ++i)
201         eps[i] = abs(yt[i] - Y[i]);
202     return eps;
203 }
204
205 int main(){
206     double start_pos = 1, end_pos = 4, y0 = 1, precision = 0.01, eps = 0.001;
207     double* answer[5];
208     int n = (end_pos - start_pos) / precision;
209     double* X = new double[n + 1];
210     for (int i = 0; i <= n; ++i)
211         X[i] = start_pos + precision * i;
212     answer[0] = X;
213     double* Y_S = shooting(start_pos, end_pos, precision, y0, eps);
214     answer[1] = Y_S;
215     double* yfd = f_diff(start_pos, end_pos, precision, y0, eps);
216     answer[2] = yfd;
217     double yt[n + 1];
218     for (int i = 0; i <= n; ++i)
219         yt[i] = y(X[i]);
220     double* err = deviation(yt, Y_S, n + 1);
221     answer[3] = err;
222     err = deviation(yt, yfd, n + 1);
223     answer[4] = err;
224     for (int i = 0; i < 5; ++i){
225         for (int j = 0; j <= n; ++j)
226             cout << fixed << answer[i][j] << " ";
227         cout << "\n";
228     }
229 }
```