

**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: И. С. Своеволин
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

1 Методы решения начальных и краевых задач для обыкновенных дифференциальных уравнений (ОДУ) и систем ОДУ

1 Постановка задачи

4.1. 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: 20

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

$$y = |x|^{3/2} \quad (2)$$

2 Результаты работы

Явный Эйлер	Улучшенный	Рунге-Кутта	Адамс	Точный ответ
2.82842712	2.82842712	2.82842712	2.82842712	2.82842712
3.04055916	3.04321081	3.04966758	3.04966758	3.04318912
3.25799449	3.26316620	3.27607947	3.27607947	3.26312733
3.48059538	3.48817496	3.50758831	3.50758831	3.48812270
3.70823601	3.71812643	3.74411656	3.73675304	3.71806401
3.94080053	3.95291687	3.98558593	3.97112540	3.95284708
4.17818158	4.19244878	4.23191880	4.20994369	4.19237403
4.42027915	4.43663030	4.48303904	4.45363257	4.43655272
4.66699967	4.68537469	4.73887247	4.70185715	4.68529615
4.91825529	4.93859987	4.99934714	4.95462338	4.93852205

Погрешности методом Рунге-Ромберга

Явный Эйлер	Улучшенный	Рунге-Кутта	Адамс
-0.00000000	-0.00000000	-0.00000000	-0.00000000
0.00176777	-0.00001762	-0.00353478	-0.00353478
0.00344613	-0.00003157	-0.00705750	-0.00887126
0.00504883	-0.00004246	-0.01059461	-0.01599983
0.00658649	-0.00005073	-0.01416599	-0.01534794
0.00806751	-0.00005675	-0.01778683	-0.01507458
0.00949867	-0.00006082	-0.02146899	-0.01447272
0.01088553	-0.00006317	-0.02522180	-0.01409484
0.01223272	-0.00006400	-0.02905272	-0.01367768
0.01354414	-0.00006349	-0.03296774	-0.01331641

Рис. 1: Вывод в консоли

3 Исходный код

matrix.h

```
1 | #pragma once
2 | #include <iostream>
3 | #include <vector>
4 | #include <ccomplex>
5 | #include <fstream>
6 |
7 | using namespace std;
8 |
9 | using cmd = complex <double>;
10 | const double pi = acos(-1);
11 |
12 | struct matrix
13 | {
```

```

14     int rows = 0, cols = 0;
15     vector <vector <double>> v;
16
17     matrix() {}
18     matrix(int _rows, int _cols)
19     {
20         rows = _rows;
21         cols = _cols;
22         v = vector <vector <double>>(rows, vector <double>(cols));
23     }
24
25     vector <double>& operator[](int row)
26     {
27         return v[row];
28     }
29
30     operator double()
31     {
32         return v[0][0];
33     }
34 };
35
36 matrix operator*(matrix lhs, matrix rhs)
37 {
38     if (lhs.cols != rhs.rows)
39         return matrix(0, 0);
40     matrix res(lhs.rows, rhs.cols);
41     for (int i = 0; i < res.rows; i++)
42     {
43         for (int j = 0; j < res.cols; j++)
44         {
45             res[i][j] = 0;
46             for (int k = 0; k < lhs.cols; k++)
47                 res[i][j] += lhs[i][k] * rhs[k][j];
48         }
49     }
50     return res;
51 }
52
53 matrix operator*(double lhs, matrix rhs)
54 {
55     for (int i = 0; i < rhs.rows; i++)
56     {
57         for (int j = 0; j < rhs.cols; j++)
58             rhs[i][j] *= lhs;
59     }
60     return rhs;
61 }
62

```

```

63 matrix operator+(matrix lhs, matrix rhs)
64 {
65     if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
66         return matrix(0, 0);
67     matrix res(lhs.rows, lhs.cols);
68     for (int i = 0; i < rhs.rows; i++)
69     {
70         for (int j = 0; j < res.cols; j++)
71             res[i][j] = lhs[i][j] + rhs[i][j];
72     }
73     return res;
74 }
75
76 matrix operator-(matrix lhs, matrix rhs)
77 {
78     if (lhs.rows != rhs.rows || rhs.cols != lhs.cols)
79         return matrix(0, 0);
80     matrix res(lhs.rows, lhs.cols);
81     for (int i = 0; i < rhs.rows; i++)
82     {
83         for (int j = 0; j < res.cols; j++)
84             res[i][j] = lhs[i][j] - rhs[i][j];
85     }
86     return res;
87 }
88
89 ostream& operator<<(ostream& stream, matrix a)
90 {
91     for (int i = 0; i < a.rows; i++)
92     {
93         for (int j = 0; j < a.cols; j++)
94             stream << a[i][j] << ' ';
95         stream << '\n';
96     }
97     return stream;
98 }
99
100 istream& operator>>(istream& stream, matrix& a)
101 {
102     for (int i = 0; i < a.rows; i++)
103     {
104         for (int j = 0; j < a.cols; j++)
105             stream >> a[i][j];
106     }
107     return stream;
108 }
109
110 matrix transposition(matrix a)
111 {

```

```

112     matrix res(a.cols, a.rows);
113     for (int i = 0; i < a.rows; i++)
114     {
115         for (int j = 0; j < a.cols; j++)
116             res[j][i] = a[i][j];
117     }
118     return res;
119 }
120
121 vector <int> swp;
122
123 pair <matrix, matrix> lu_decomposition(matrix a)
124 {
125     int n = a.rows;
126     matrix l(n, n);
127     swp = vector <int>(0);
128     for (int k = 0; k < n; k++)
129     {
130         matrix prev = a;
131         int idx = k;
132         for (int i = k + 1; i < n; i++)
133         {
134             if (abs(prev[idx][k]) < abs(prev[i][k]))
135                 idx = i;
136         }
137         swap(prev[k], prev[idx]);
138         swap(a[k], a[idx]);
139         swap(l[k], l[idx]);
140         swp.push_back(idx);
141         for (int i = k + 1; i < n; i++)
142         {
143             double h = prev[i][k] / prev[k][k];
144             l[i][k] = h;
145             for (int j = k; j < n; j++)
146                 a[i][j] = prev[i][j] - h * prev[k][j];
147         }
148     }
149 }
150 for (int i = 0; i < n; i++)
151     l[i][i] = 1;
152 return { l, a };
153 }
154
155 matrix solve_triag(matrix a, matrix b, bool up)
156 {
157     int n = a.rows;
158     matrix res(n, 1);
159     int d = up ? -1 : 1;
160     int first = up ? n - 1 : 0;

```

```

161     for (int i = first; i < n && i >= 0; i += d)
162     {
163         res[i][0] = b[i][0];
164         for (int j = 0; j < n; j++)
165         {
166             if (i != j)
167                 res[i][0] -= a[i][j] * res[j][0];
168         }
169         res[i][0] = res[i][0] / a[i][i];
170     }
171     return res;
172 }
173
174 matrix solve_gauss(pair <matrix, matrix> lu, matrix b)
175 {
176     for (int i = 0; i < swp.size(); i++)
177         swap(b[i], b[swp[i]]);
178     matrix z = solve_triag(lu.first, b, false);
179     matrix x = solve_triag(lu.second, z, true);
180     //for (int i = 0; i < swp.size(); i++)
181         //swap(x[i], x[swp[i]]);
182     return x;
183 }
184
185 matrix inverse(matrix a)
186 {
187     int n = a.rows;
188     matrix b(n, 1);
189     pair <matrix, matrix> lu = lu_decomposition(a);
190     matrix res(n, n);
191     for (int i = 0; i < n; i++)
192     {
193         b[max(i - 1, 0)][0] = 0;
194         b[i][0] = 1;
195         matrix col = solve_gauss(lu, b);
196         for (int j = 0; j < n; j++)
197             res[j][i] = col[j][0];
198     }
199     return res;
200 }
201
202 double determinant(matrix a)
203 {
204     int n = a.rows;
205     pair <matrix, matrix> lu = lu_decomposition(a);
206     double det = 1;
207     for (int i = 0; i < n; i++)
208         det *= lu.second[i][i];
209     return det;

```

```

210 }
211
212 matrix solve_tridiagonal(matrix& a, matrix& b)
213 {
214     int n = a.rows;
215     vector <double> p(n), q(n);
216     p[0] = -a[0][1] / a[0][0];
217     q[0] = b[0][0] / a[0][0];
218     for (int i = 1; i < n; i++)
219     {
220         if (i != n - 1)
221             p[i] = -a[i][i + 1] / (a[i][i] + a[i][i - 1] * p[i - 1]);
222         else
223             p[i] = 0;
224         q[i] = (b[i][0] - a[i][i - 1] * q[i - 1]) / (a[i][i] + a[i][i - 1] * p[i - 1]);
225     }
226     matrix res(n, 1);
227     res[n - 1][0] = q[n - 1];
228     for (int i = n - 2; i >= 0; i--)
229         res[i][0] = p[i] * res[i + 1][0] + q[i];
230     return res;
231 }
232
233 double abs(matrix a)
234 {
235     double mx = 0;
236     for (int i = 0; i < a.rows; i++)
237     {
238         double s = 0;
239         for (int j = 0; j < a.cols; j++)
240             s += abs(a[i][j]);
241         mx = max(mx, s);
242     }
243     return mx;
244 }
245
246 matrix solve_iteration(matrix a, matrix b, double eps)
247 {
248     int n = a.rows;
249     matrix alpha(n, n), beta(n, 1);
250     for (int i = 0; i < n; i++)
251     {
252         for (int j = 0; j < n; j++)
253             alpha[i][j] = -a[i][j] / a[i][i];
254         alpha[i][i] = 0;
255     }
256     for (int i = 0; i < n; i++)
257         beta[i][0] = b[i][0] / a[i][i];
258     matrix x = beta;

```



```

259     double m = abs(a);
260     double epsk = 2 * eps;
261     while (epsk > eps)
262     {
263         matrix prev = x;
264         x = beta + alpha * x;
265         if (m < 1)
266             epsk = m / (1 - m) * abs(x - prev);
267         else
268             epsk = abs(x - prev);
269     }
270     return x;
271 }
272
273 matrix solve_seidel(matrix a, matrix b, double eps)
274 {
275     int n = a.rows;
276     matrix alpha(n, n), beta(n, 1);
277     for (int i = 0; i < n; i++)
278     {
279         for (int j = 0; j < n; j++)
280             alpha[i][j] = -a[i][j] / a[i][i];
281         alpha[i][i] = 0;
282     }
283     for (int i = 0; i < n; i++)
284         beta[i][0] = b[i][0] / a[i][i];
285     matrix x = beta;
286     double m = abs(alpha);
287     double epsk = 2 * eps;
288     while (epsk > eps)
289     {
290         matrix prev = x;
291         for (int i = 0; i < n; i++)
292         {
293             double cur = beta[i][0];
294             for (int j = 0; j < n; j++)
295                 cur += alpha[i][j] * x[j][0];
296             x[i][0] = cur;
297         }
298         if (m < 1)
299             epsk = m / (1 - m) * abs(x - prev);
300         else
301             epsk = abs(x - prev);
302     }
303     return x;
304 }
305
306 pair <matrix, matrix> method_jacobi(matrix a, double eps)
307 {

```

```

308     int n = a.rows;
309     double epsk = 2 * eps;
310     matrix vec(n, n);
311     for (int i = 0; i < n; i++)
312         vec[i][i] = 1;
313     while (epsk > eps)
314     {
315         int cur_i = 1, cur_j = 0;
316         for (int i = 0; i < n; i++)
317         {
318             for (int j = 0; j < i; j++)
319             {
320                 if (abs(a[cur_i][cur_j]) < abs(a[i][j]))
321                 {
322                     cur_i = i;
323                     cur_j = j;
324                 }
325             }
326         }
327         matrix u(n, n);
328         double phi = pi / 4;
329         if (abs(a[cur_i][cur_i] - a[cur_j][cur_j]) > 1e-7)
330             phi = 0.5 * atan((2 * a[cur_i][cur_j]) / (a[cur_i][cur_i] - a[cur_j][cur_j]));
331         for (int i = 0; i < n; i++)
332             u[i][i] = 1;
333         u[cur_i][cur_j] = -sin(phi);
334         u[cur_i][cur_i] = cos(phi);
335         u[cur_j][cur_i] = sin(phi);
336         u[cur_j][cur_j] = cos(phi);
337         vec = vec * u;
338         a = transposition(u) * a * u;
339         epsk = 0;
340         for (int i = 0; i < n; i++)
341         {
342             for (int j = 0; j < i; j++)
343                 epsk += a[i][j] * a[i][j];
344         }
345         epsk = sqrt(epsk);
346     }
347     matrix val(n, 1);
348     for (int i = 0; i < n; i++)
349         val[i][0] = a[i][i];
350     return { val, vec };
351 }
352
353 double sign(double x)
354 {
355     return x > 0 ? 1 : -1;

```

```

356 }
357
358 pair <matrix, matrix> qr_decomposition(matrix a)
359 {
360     int n = a.rows;
361     matrix e(n, n);
362     for (int i = 0; i < n; i++)
363         e[i][i] = 1;
364     matrix q = e;
365     for (int i = 0; i < n - 1; i++)
366     {
367         matrix v(n, 1);
368         double s = 0;
369         for (int j = i; j < n; j++)
370             s += a[j][i] * a[j][i];
371         v[i][0] = a[i][i] + sign(a[i][i]) * sqrt(s);
372         for (int j = i + 1; j < n; j++)
373             v[j][0] = a[j][i];
374         matrix h = e - (2.0 / double(transposition(v) * v)) * (v * transposition(v));
375         q = q * h;
376         a = h * a;
377     }
378     return { q, a };
379 }
380
381 vector <cmd> qr_eigenvalues(matrix a, double eps)
382 {
383     int n = a.rows;
384     vector <cmd> prev(n);
385     while (true)
386     {
387         pair <matrix, matrix> p = qr_decomposition(a);
388         a = p.second * p.first;
389         vector <cmd> cur;
390         for (int i = 0; i < n; i++)
391         {
392             if (i < n - 1 && abs(a[i + 1][i]) > 1e-7)
393             {
394                 double b = -(a[i][i] + a[i + 1][i + 1]);
395                 double c = a[i][i] * a[i + 1][i + 1] - a[i][i + 1] * a[i + 1][i];
396                 double d = b * b - 4 * c;
397                 cmd sgn = (d > 0) ? cmd(1, 0) : cmd(0, 1);
398                 d = sqrt(abs(d));
399                 cur.push_back(0.5 * (-b - sgn * d));
400                 cur.push_back(0.5 * (-b + sgn * d));
401                 i++;
402             }
403             else
404                 cur.push_back(a[i][i]);

```

```

405     }
406     bool ok = true;
407     for (int i = 0; i < n; i++)
408         ok = ok && abs(cur[i] - prev[i]) < eps;
409     if (ok)
410         break;
411     prev = cur;
412 }
413 return prev;
414 }

```

4-1.cpp

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <fstream>
5  #include <functional>
6  #include <algorithm>
7  #include "matrix.h"
8
9  using namespace std;
10 using equation = function <double(vector <double>>>;
11 using func = function <double(double)>;
12
13 double runge_romberg(double i1, double i2, double h1, double h2, double p)
14 {
15     double k = h2 / h1;
16     return (i1 - i2) / (pow(k, p) - 1);
17 }
18
19 vector <equation> make_system(equation dy, int k)
20 {
21     vector <equation> res(k);
22     for (int i = 0; i < k - 1; i++)
23     {
24         auto f = [=](vector <double> x) mutable
25         {
26             return x[i + 2];
27         };
28         res[i] = f;
29     }
30     res[k - 1] = dy;
31     return res;
32 }
33
34 vector <vector <double>> explicit_euler(vector <equation> dy, vector <double> yk,
35     double a, double b, double h)
36 {

```

```

36     int n = dy.size();
37     vector <vector <double>> res(n);
38     for (int i = 0; i < n; i++)
39         res[i].push_back(yk[i]);
40     for (double x = a; x <= b - h; x += h)
41     {
42         vector <double> args;
43         args.push_back(x);
44         for (int i = 0; i < n; i++)
45             args.push_back(res[i].back());
46         for (int i = 0; i < n; i++)
47             res[i].push_back(res[i].back() + h * dy[i](args));
48     }
49     return res;
50 }
51
52 vector <vector <double>> improved_euler(vector <equation> dy, vector <double> yk,
53     double a, double b, double h)
54 {
55     int n = dy.size();
56     vector <vector <double>> tmp(n);
57     vector <vector <double>> res(n);
58     for (int i = 0; i < n; i++)
59     {
60         tmp[i].push_back(yk[i]);
61         res[i].push_back(yk[i]);
62     }
63     int k = 0;
64     for (double x = a; x <= b - h / 2; x += h / 2)
65     {
66         vector <double> args;
67         args.push_back(x);
68         for (int i = 0; i < n; i++)
69             args.push_back(tmp[i].back());
70         for (int i = 0; i < n; i++)
71         {
72             tmp[i].push_back(tmp[i][tmp[i].size() - 1 - k % 2] + 0.5 * (k % 2 + 1) * h
73                 * dy[i](args));
74             if (k % 2)
75                 res[i].push_back(tmp[i].back());
76         }
77         k++;
78     }
79     return res;
80 }
81
82 vector <double> make_args(double x, const vector <double>& y, double add_x, double
83     add_y)
84 {

```

```

82     vector <double> res;
83     res.push_back(x + add_x);
84     for (int i = 0; i < y.size(); i++)
85         res.push_back(y[i] + add_y);
86     return res;
87 }
88
89 vector <vector <double>> runge_kutta(vector <equation> dy, vector <double> yk, double
    1, double r, double h)
90 {
91     // для4 порядкаточности
92     int p = 4;
93     vector <double> a = { 0, 0, 0.5, 0.5, 1 };
94     vector <vector <double>> b = { {}, {0}, {0, 0.5}, {0, 0, 0.5}, {0, 0, 0, 0.5} };
95     vector <double> c = { 0, 1. / 6, 1. / 3, 1. / 3, 1. / 6 };
96     //
97     int n = dy.size();
98     vector <vector <double>> res(n);
99     for (int i = 0; i < n; i++)
100         res[i].push_back(yk[i]);
101     vector <double> K(p + 1);
102     for (double x = 1; x <= r - h; x += h)
103     {
104         vector <double> y;
105         for (int idx = 0; idx < n; idx++)
106             y.push_back(res[idx].back());
107         for (int idx = 0; idx < n; idx++)
108         {
109             K[1] = h * dy[idx](make_args(x, y, 0, 0));
110             for (int i = 2; i <= p; i++)
111             {
112                 double add = 0;
113                 for (int j = 1; j <= i - 1; j++)
114                     add += b[i][j] * K[j];
115                 K[i] = h * dy[idx](make_args(x, y, a[i] * h, add));
116             }
117             double delta = 0;
118             for (int i = 1; i <= p; i++)
119                 delta += c[i] * K[i];
120             res[idx].push_back(res[idx].back() + delta);
121         }
122     }
123     return res;
124 }
125
126 vector <vector <double>> adams(vector <equation> dy, vector <double> yk, double a,
    double b, double h)
127 {
128     int n = dy.size();

```

```

129 vector <vector <double>> y = runge_kutta(dy, yk, a, b, h);
130 int m = y[0].size();
131 vector <vector <double>> res(n);
132 for (int i = 0; i < n; i++)
133 {
134     for (int j = 0; j < 4; j++)
135         res[i].push_back(y[i][j]);
136 }
137 for (int k = 4; k < m; k++)
138 {
139     vector <vector <double>> args(4);
140     for (int j = 0; j < 4; j++)
141     {
142         args[j].push_back(a + h * (k - j - 1));
143         for (int i = 0; i < n; i++)
144             args[j].push_back(res[i][k - j - 1]);
145     }
146     for (int i = 0; i < n; i++)
147     {
148         double delta = 55 * dy[i](args[0]) - 59 * dy[i](args[1]) + 37 * dy[i](args
149             [2]) - 9 * dy[i](args[3]);
150         res[i].push_back(res[i].back() + (h / 24) * delta);
151     }
152 }
153 return res;
154 }
155 int main()
156 {
157     setlocale(LC_ALL, "Rus");
158     ofstream fout("answer4-1.txt");
159     fout.precision(8);
160     fout << fixed;
161
162     auto ddy = [](vector <double> x)
163     {
164         return ( -1./2 * x[2] + 3./4 * x[1] ) / ( x[0] * (x[0] - 1) );
165     };
166     double h1 = 0.1, h2 = 0.05;
167     vector <vector <double>> y1 = explicit_euler(make_system(ddy, 2), { 2 * sqrt(2), 3.
168         / 2 * sqrt(2) }, 2, 3, h1);
169     vector <vector <double>> y2 = improved_euler(make_system(ddy, 2), { 2 * sqrt(2), 3.
170         / 2 * sqrt(2) }, 2, 3, h1);
171     vector <vector <double>> y3 = runge_kutta(make_system(ddy, 2), { 2 * sqrt(2), 3. / 2
172         * sqrt(2) }, 2, 3, h1);
173     vector <vector <double>> y4 = adams(make_system(ddy, 2), { 2 * sqrt(2), 3. / 2 *
174         sqrt(2) }, 2, 3, h1);
175     fout << "Явный Эйлер" << "\t" << "Улучшенный" << "\t" << "РунгеКутта-" << "\t" << "
176         Адамс " << "\t" << "Точный ответ" << endl;

```

```

172     for (int i = 0; i < y1[0].size(); i++)
173     {
174         double x = 2 + h1 * i;
175         fout << y1[0][i] << '\t' << y2[0][i] << '\t' << y3[0][i] << '\t' << y4[0][i] <<
            '\t' << pow(abs(x), 3./2) << endl;
176     }
177     vector <vector <double>> y12 = explicit_euler(make_system(ddy, 2), { 2 * sqrt(2), 3.
        / 2 * sqrt(2) }, 2, 3, h2);
178     vector <vector <double>> y22 = improved_euler(make_system(ddy, 2), { 2 * sqrt(2), 3.
        / 2 * sqrt(2) }, 2, 3, h2);
179     vector <vector <double>> y32 = runge_kutta(make_system(ddy, 2), { 2 * sqrt(2), 3. /
        2 * sqrt(2) }, 2, 3, h2);
180     vector <vector <double>> y42 = adams(make_system(ddy, 2), { 2 * sqrt(2), 3. / 2 *
        sqrt(2) }, 2, 3, h2);
181     fout << "\Погрешностиn методомРунгеРомберга-\n";
182     fout << "Явный Эйлер" << "\t" << "Улучшенный" << "\t" << "РунгеКутта-" << "\t" << "
        Адамс " << endl;
183     for (int i = 0; i < y1[0].size(); i++)
184     {
185         fout << runge_romberg(y1[0][i], y12[0][2 * i], h1, h2, 2) << '\t'
186             << runge_romberg(y2[0][i], y22[0][2 * i], h1, h2, 4) << '\t'
187             << runge_romberg(y3[0][i], y32[0][2 * i], h1, h2, 4) << '\t'
188             << runge_romberg(y4[0][i], y42[0][2 * i], h1, h2, 4) << endl;
189     }
190 }

```


4 Постановка задачи

4.2. 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: 20 (Сделан 21, поскольку 20 вариант прописан с ошибкой в файле с ТЗ)

$$x(2x + 1)y'' + 2(x + 1)y' - 2y = 0, \quad y'(1) = 0, \quad y(3) - y'(3) = \frac{31}{9} \quad (3)$$

$$y(x) = x + 1 + \frac{1}{x} \quad (4)$$

5 Результаты работы

Стрельба	Разности	Точное
3.05909138	3.05755377	3.00000000
3.05909138	3.05755377	3.00909091
3.07819541	3.07394816	3.03333333
3.11159727	3.10296242	3.06923077
3.15598466	3.14189780	3.11428571
3.20899988	3.18877364	3.16666667
3.26891863	3.24210348	3.22500000
3.33445037	3.30075003	3.28823529
3.40460984	3.36382830	3.35555556
3.47863168	3.43063937	3.42631579
3.55591222	3.50062390	3.50000000
3.63596885	3.57332900	3.57619048
3.71841103	3.64838400	3.65454545
3.80291934	3.72548265	3.73478261
3.88922989	3.80436968	3.81666667
3.97712278	3.88483063	3.90000000
4.06641323	3.96668403	3.98461538
4.15694483	4.04977525	4.07037037
4.24858426	4.13397173	4.15714286
4.34121719	4.21915921	4.24482759
Погрешности методом Рунге-Ромберга		
Стрельба	Разности	
-0.07136471	-0.03817839	
-0.06622140	-0.03216416	
-0.06357545	-0.02695730	
-0.06260307	-0.02238041	
-0.06279289	-0.01830404	
-0.06381482	-0.01463170	
-0.06544857	-0.01129005	
-0.06754294	-0.00822237	
-0.06999159	-0.00538417	
-0.07271807	-0.00274004	
-0.07566640	-0.00026147	
-0.07879487	0.00207475	
-0.08207193	0.00428769	
-0.08547337	0.00639318	
-0.08898038	0.00840446	
-0.09257818	0.01033267	
-0.09625501	0.01218729	
-0.10000144	0.01397639	
-0.10380980	0.01570692	
-0.10767384	0.01738487	

Рис. 2: Вывод в консоли

6 Исходный код

4-2.cpp

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <fstream>
5  #include <functional>
6  #include <algorithm>
7  #include "matrix.h"
8
9  using namespace std;
10 using equation = function <double(vector <double>>>;
11 using func = function <double(double)>;
12
13 double runge_romberg(double i1, double i2, double h1, double h2, double p)
14 {
15     double k = h2 / h1;
16     return (i1 - i2) / (pow(k, p) - 1);
17 }
18
19 vector <equation> make_system(equation dy, int k)
20 {
21     vector <equation> res(k);
22     for (int i = 0; i < k - 1; i++)
23     {
24         auto f = [=](vector <double> x) mutable
25         {
26             return x[i + 2];
27         };
28         res[i] = f;
29     }
30     res[k - 1] = dy;
31     return res;
32 }
33
34 vector <vector <double>> explicit_euler(vector <equation> dy, vector <double> yk,
35     double a, double b, double h)
36 {
37     int n = dy.size();
38     vector <vector <double>> res(n);
39     for (int i = 0; i < n; i++)
40         res[i].push_back(yk[i]);
41     for (double x = a; x <= b - h; x += h)
42     {
43         vector <double> args;
44         args.push_back(x);
45         for (int i = 0; i < n; i++)
```

```

45         args.push_back(res[i].back());
46         for (int i = 0; i < n; i++)
47             res[i].push_back(res[i].back() + h * dy[i](args));
48     }
49     return res;
50 }
51
52 vector <vector <double>> improved_euler(vector <equation> dy, vector <double> yk,
53     double a, double b, double h)
54 {
55     int n = dy.size();
56     vector <vector <double>> tmp(n);
57     vector <vector <double>> res(n);
58     for (int i = 0; i < n; i++)
59     {
60         tmp[i].push_back(yk[i]);
61         res[i].push_back(yk[i]);
62     }
63     int k = 0;
64     for (double x = a; x <= b - h / 2; x += h / 2)
65     {
66         vector <double> args;
67         args.push_back(x);
68         for (int i = 0; i < n; i++)
69             args.push_back(tmp[i].back());
70         for (int i = 0; i < n; i++)
71         {
72             tmp[i].push_back(tmp[i][tmp[i].size() - 1 - k % 2] + 0.5 * (k % 2 + 1) * h
73                 * dy[i](args));
74             if (k % 2)
75                 res[i].push_back(tmp[i].back());
76         }
77         k++;
78     }
79     return res;
80 }
81
82 vector <double> make_args(double x, const vector <double>& y, double add_x, double
83     add_y)
84 {
85     vector <double> res;
86     res.push_back(x + add_x);
87     for (int i = 0; i < y.size(); i++)
88         res.push_back(y[i] + add_y);
89     return res;
90 }
91
92 vector <vector <double>> runge_kutta(vector <equation> dy, vector <double> yk, double
93     l, double r, double h)

```

```

90 {
91     // для порядкаточности
92     int p = 4;
93     vector <double> a = { 0, 0, 0.5, 0.5, 1 };
94     vector <vector <double>> b = { {}, {0}, {0, 0.5}, {0, 0, 0.5}, {0, 0, 0, 0.5} };
95     vector <double> c = { 0, 1. / 6, 1. / 3, 1. / 3, 1. / 6 };
96     //
97     int n = dy.size();
98     vector <vector <double>> res(n);
99     for (int i = 0; i < n; i++)
100         res[i].push_back(yk[i]);
101     vector <double> K(p + 1);
102     for (double x = l; x <= r - h; x += h)
103     {
104         vector <double> y;
105         for (int idx = 0; idx < n; idx++)
106             y.push_back(res[idx].back());
107         for (int idx = 0; idx < n; idx++)
108         {
109             K[1] = h * dy[idx](make_args(x, y, 0, 0));
110             for (int i = 2; i <= p; i++)
111             {
112                 double add = 0;
113                 for (int j = 1; j <= i - 1; j++)
114                     add += b[i][j] * K[j];
115                 K[i] = h * dy[idx](make_args(x, y, a[i] * h, add));
116             }
117             double delta = 0;
118             for (int i = 1; i <= p; i++)
119                 delta += c[i] * K[i];
120             res[idx].push_back(res[idx].back() + delta);
121         }
122     }
123     return res;
124 }
125
126 vector <vector <double>> adams(vector <equation> dy, vector <double> yk, double a,
127     double b, double h)
128 {
129     int n = dy.size();
130     vector <vector <double>> y = runge_kutta(dy, yk, a, b, h);
131     int m = y[0].size();
132     vector <vector <double>> res(n);
133     for (int i = 0; i < n; i++)
134     {
135         for (int j = 0; j < 4; j++)
136             res[i].push_back(y[i][j]);
137     }
138     for (int k = 4; k < m; k++)

```

```

138 {
139     vector <vector <double>> args(4);
140     for (int j = 0; j < 4; j++)
141     {
142         args[j].push_back(a + h * (k - j - 1));
143         for (int i = 0; i < n; i++)
144             args[j].push_back(res[i][k - j - 1]);
145     }
146     for (int i = 0; i < n; i++)
147     {
148         double delta = 55 * dy[i](args[0]) - 59 * dy[i](args[1]) + 37 * dy[i](args
149             [2]) - 9 * dy[i](args[3]);
150         res[i].push_back(res[i].back() + (h / 24) * delta);
151     }
152     return res;
153 }
154
155 vector <double> shooting(equation ddy, double a, double b, vector <double> alpha,
156     vector <double> beta, double ya, double yb, double h)
157 {
158     double eps = 0.00000001;
159     vector <equation> v = make_system(ddy, 2);
160     auto phi = [&](double n)
161     {
162         vector <double> args;
163         if (abs(beta[0]) > eps)
164             args = { n, (ya - alpha[0] * n) / beta[0] };
165         else
166             args = { ya / alpha[0], n };
167         vector <vector <double>> yk = runge_kutta(v, args, a, b, h);
168         return alpha[1] * yk[0].back() + beta[1] * yk[1].back() - yb;
169     };
170     double n0 = 10, n1 = -1;
171     double phi0 = phi(n0);
172     double phi1 = phi(n1);
173     double n;
174     while (true)
175     {
176         n = n1 - ((n1 - n0) / (phi1 - phi0)) * phi1;
177         double phij = phi(n);
178         if (abs(phij) < eps)
179             break;
180         n0 = n1;
181         n1 = n;
182         phi0 = phi1;
183         phi1 = phij;
184     }
185     vector <double> args;

```

```

185     if (abs(beta[0]) > eps)
186         args = { n, (ya - alpha[0] * n) / beta[0] };
187     else
188         args = { ya / alpha[0], n };
189     vector <vector <double>> res = runge_kutta(v, args, a, b, h);
190     return res[0];
191 }
192
193 vector <double> finite_difference(func f, func p, func q, double a, double b, vector <
    double> alpha, vector <double> beta, double ya, double yb, double h)
194 {
195     vector <double> x;
196     for (int i = 0; a + i * h < b; i++)
197         x.push_back(a + i * h);
198     int n = x.size();
199     matrix A(n + 1, n + 1);
200     matrix B(n + 1, 1);
201     A[0][0] = alpha[0] * h - beta[0];
202     A[0][1] = beta[0];
203     B[0][0] = h * ya;
204     for (int i = 1; i <= n - 1; i++)
205     {
206         A[i][i + 1] = 1 + p(x[i]) * h / 2;
207         A[i][i] = -2 + h * h * q(x[i]);
208         A[i][i - 1] = 1 - p(x[i]) * h / 2;
209         B[i][0] = h * h * f(x[i]);
210     }
211     A[n][n - 1] = -beta[1];
212     A[n][n] = alpha[1] * h + beta[1];
213     B[n][0] = h * yb;
214     matrix sol = solve_tridiagonal(A, B);
215     vector <double> res;
216     for (int i = 0; i < n + 1; i++)
217         res.push_back(sol[i][0]);
218     return res;
219 }
220
221 int main()
222 {
223     setlocale(LC_ALL, "Rus");
224     ofstream fout("answer4-2.txt");
225     fout.precision(8);
226     fout << fixed;
227
228     auto ddy = [] (vector <double> x)
229     {
230         return (2 * x[1] - 2 * (x[0] + 1) * x[2]) / (x[0] * (2 * x[0] + 1));
231     };
232     double h1 = 0.1;

```

```

233 vector <double> y1 = shooting(ddy, 1, 3, { 0, 1 }, { 1, -1 }, 0, 31. / 9, h1);
234
235 auto f = [](double x) { return 0; };
236 auto p = [](double x) { return 2 * (x + 1) / (x * (2 * x + 1)); };
237 auto q = [](double x) { return -2 / (x * (2 * x + 1)); };
238 vector <double> y2 = finite_difference(f, p, q, 1, 3, { 0, 1 }, { 1, -1 }, 0, 31. /
    9, h1);
239 fout << "Стрельба" << "\t" << "Разности" << '\t' << "Точное" << endl;
240 for (int i = 0; i < y1.size(); i++)
241 {
242     double x = 1 + i * h1;
243     fout << y1[i] << '\t' << y2[i] << '\t' << (x + 1 + 1 / x) << endl;
244 }
245 double h2 = 0.05;
246 vector <double> y12 = shooting(ddy, 1, 3, { 0, 1 }, { 1, -1 }, 0, 31. / 9, h2);
247 vector <double> y22 = finite_difference(f, p, q, 1, 3, { 0, 1 }, { 1, -1 }, 0, 31.
    / 9, h2);
248 fout << "\Погрешности методом Рунге-Ромберга-\n";
249 fout << "Стрельба" << "\t" << "Разности" << endl;
250 for (int i = 0; i < y1.size(); i++)
251 {
252     fout << runge_romberg(y1[i], y12[2 * i], h1, h2, 4) << '\t'
253         << runge_romberg(y2[i], y22[2 * i], h1, h2, 2) << endl;
254 }
255 }

```