

**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-408Б-20
Дата:
Оценка:
Подпись:

Москва, 2024

1 Методы решения нелинейных уравнений и систем нелинейных уравнений

1 Постановка задачи

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант: 20

$$tgx - 5x^2 + 1 = 0 \tag{1}$$

2 Результаты работы

```
Ньютон: x=1.4690, f(x)=0.0000
Количество итераций: 4
Простые итерации: x=1.4690, f(x)=0.0000
Количество итераций: 5
```

Рис. 1: Вывод в консоли

3 Исходный код

matrix.h

```
1  #pragma once
2  #include <iostream>
3  #include <vector>
4  #include <complex>
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             ));
332     for (int i = 0; i < n; i++)
333         u[i][i] = 1;
334     u[cur_i][cur_j] = -sin(phi);
335     u[cur_j][cur_i] = cos(phi);
336     u[cur_j][cur_j] = sin(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 }

```

2-1.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <fstream>
5 #include <functional>
6 #include "matrix.h"

```

```

7
8 using namespace std;
9
10 using double_n = vector <double (*)(vector <double>)>;
11 using double_nn = vector <vector <double (*)(vector <double>)>>;
12 int iter = 0;
13
14 double dichotomy(function <double(double)> f, double l, double r, double eps)
15 {
16     while (abs(l - r) > eps)
17     {
18         double nl = l + 0.3 * (r - l);
19         double nr = r - 0.3 * (r - l);
20         if (f(nl) < f(nr))
21             l = nl;
22         else
23             r = nr;
24     }
25     return (l + r) / 2;
26 }
27
28 double newton(double (*f)(double), double (*df)(double), double x, double eps)
29 {
30     double xk = x - f(x) / df(x);
31     iter = 0;
32     while (abs(xk - x) > eps)
33     {
34         x = xk;
35         xk = x - f(x) / df(x);
36         iter++;
37     }
38     return xk;
39 }
40
41 double iteration(double (*phi)(double), double (*dphi)(double), double l, double r,
42     double eps)
43 {
44     double x = (l + r) / 2;
45     double xk = phi(x);
46     iter = 0;
47     double q = dphi(dichotomy(dphi, l, r, eps));
48     while ((q * abs(xk - x) / (1 - q)) > eps)
49     {
50         x = xk;
51         xk = phi(x);
52         iter++;
53     }
54     return xk;
55 }

```

```

55
56 double f(double x)
57 {
58     return tan(x) - 5. * pow(x, 2) + 1;
59 }
60
61 double df(double x)
62 {
63     return (1. / pow(cos(x), 2)) - 10. * x;
64 }
65
66 double phi(double x)
67 {
68     return atan(5. * pow(x, 2) - 1.);
69 }
70
71 double dphi(double x)
72 {
73     return (10*x) / (pow((5 * pow(x, 2) - 1), 2) + 1);
74 }
75
76 int main()
77 {
78     setlocale(LC_ALL, "Rus");
79     ofstream fout("answer2-1.txt");
80     fout.precision(4);
81     fout << fixed;
82     double ans = newton(f, df, 1.5, 0.00001);
83     fout << "Ньютон: x=" << ans << ", f(x)=" << f(ans) << '\n';
84     fout << "Количество итераций: " << iter << '\n';
85
86     ans = iteration(phi, dphi, 1, 2, 0.00001);
87     fout << "Простые итерации: x=" << ans << ", f(x)=" << f(ans) << '\n';
88     fout << "Количество итераций: " << iter << '\n';
89 }

```

4 Постановка задачи

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Вариант: 20

$$\begin{cases} x_1^2 - 2lgx_2 - 1 = 0 \\ x_1^2 - 2x_1 * x_2 + 2 = 0 \end{cases} \quad (2)$$

5 Результаты работы

```
Система методом Ньютона: x1=1.1488, x2=1.4449, f1(x)=0.0000, f2(x)=0.0000
Количество итераций: 4
Система методом простых итераций: x1=1.1488, x2=1.4449, f1(x)=0.0000, f2(x)=-0.0000
Количество итераций: 8
```

Рис. 2: Вывод в консоли

6 Исходный код

2-2.cpp

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <cmath>
4 | #include <fstream>
5 | #include <functional>
6 | #include "matrix.h"
7 |
8 | using namespace std;
9 |
10 | using double_n = vector <double (*) (vector <double>)>;
11 | using double_nn = vector <vector <double (*) (vector <double>)>>;
12 | int iter = 0;
13 |
14 | double dichotomy(function <double(double)> f, double l, double r, double eps)
15 | {
16 |     while (abs(l - r) > eps)
17 |     {
18 |         double nl = l + 0.3 * (r - l);
19 |         double nr = r - 0.3 * (r - l);
20 |         if (f(nl) < f(nr))
21 |             l = nl;
22 |         else
23 |             r = nr;
24 |     }
25 |     return (l + r) / 2;
26 | }
27 |
28 | vector <double> newton_system(double_n f, double_nn df, vector <double> x, double eps)
```

```

29 {
30     int n = f.size();
31     matrix x0(n, 1);
32     for (int i = 0; i < n; i++)
33         x0[i][0] = x[i];
34     matrix xk = x0;
35     iter = 0;
36     do
37     {
38         iter++;
39         x0 = xk;
40         for (int i = 0; i < n; i++)
41             x[i] = x0[i][0];
42         matrix a(n, n);
43         for (int i = 0; i < n; i++)
44         {
45             for (int j = 0; j < n; j++)
46                 a[i][j] = df[i][j](x);
47         }
48         matrix b(n, 1);
49         for (int i = 0; i < n; i++)
50             b[i][0] = -f[i](x);
51         xk = x0 + solve_gauss(lu_decomposition(a), b);
52     } while (abs(xk - x0) > eps);
53     vector <double> res(n);
54     for (int i = 0; i < n; i++)
55         res[i] = xk[i][0];
56     return res;
57 }
58
59 vector <double> iteration_system(double_n phi, double_nn dphi, vector <double> l,
60     vector <double> r, double eps)
61 {
62     double n = phi.size();
63     vector <double> xk(n), x(n);
64     for (int i = 0; i < n; i++)
65         xk[i] = (l[i] + r[i]) / 2;
66     vector <double> qn = xk;
67     auto dphinx = [&](vector <double> x)
68     {
69         double res = 0;
70         for (int j = 0; j < n; j++)
71         {
72             double tmp = 0;
73             for (int k = 0; k < n; k++)
74                 tmp += abs(dphi[j][k](x));
75             res = max(res, tmp);
76         }
77         return res;
78     };

```



```

77     };
78     for (int i = 0; i < n; i++)
79     {
80         auto dphix = [&](double x)
81         {
82             vector <double> xn = qn;
83             xn[i] = x;
84             return dphinx(xn);
85         };
86         qn[i] = dichotomy(dphix, l[i], r[i], eps);
87     }
88     double q = dphinx(qn);
89     double epsk = 0;
90     iter = 0;
91     do
92     {
93         iter++;
94         x = xk;
95         for (int i = 0; i < n; i++)
96             xk[i] = phi[i](x);
97         epsk = 0;
98         for (int i = 0; i < n; i++)
99         {
100             double s = abs(x[i] - xk[i]);
101             epsk = max(epsk, s);
102         }
103     } while (q * epsk / (1 - q) > eps);
104     return xk;
105 }
106
107 double f1(vector <double> x)
108 {
109     return x[0] * x[0] - 2. * log10(x[1]) - 1.;
110 }
111
112 double f2(vector <double> x)
113 {
114     return x[0] * x[0] - 2. * x[0] * x[1] + 2.;
115 }
116
117 double_n fn = { f1, f2 };
118
119 double df11(vector <double> x)
120 {
121     return 2. * x[0];
122 }
123
124 double df12(vector <double> x)
125 {
126     return - 2. / (x[1] * log(10));

```

```

126 }
127
128 double df21(vector <double> x)
129 {
130     return 2 * x[0] - 2 * x[1];
131 }
132
133 double df22(vector <double> x)
134 {
135     return - 2 * x[0];
136 }
137
138 double_nn dfn = { {df11, df12}, {df21, df22} };
139
140 double phi1(vector <double> x)
141 {
142     return sqrt(2. * log10(x[1]) + 1);
143 }
144
145 double phi2(vector <double> x)
146 {
147     return (pow(x[0], 2) + 2) / (2 * x[0]);
148 }
149
150 double_n phin = { phi1, phi2 };
151
152 double dphi11(vector <double> x)
153 {
154     return 0;
155 }
156
157 double dphi12(vector <double> x)
158 {
159     return pow((x[1] * sqrt(log(10))) * sqrt(2*log(x[1]) + log(10)), -1);
160 }
161
162 double dphi21(vector <double> x)
163 {
164     return pow(2, -1) - pow(x[0], -2);
165 }
166
167 double dphi22(vector <double> x)
168 {
169     return 0;
170 }
171
172 double_nn dphin = { {dphi11, dphi12}, {dphi21, dphi22} };
173
174 int main()

```

```

175 {
176     setlocale(LC_ALL, "Rus");
177     ofstream fout("answer2-2.txt");
178     fout.precision(4);
179     fout << fixed;
180     vector <double> x = newton_system(fn, dfn, { 1, 1 }, 0.00001);
181     fout << "Система методом Ньютона: x1=" << x[0] << ", x2=" << x[1] << ", f1(x)=" <<
        f1(x) << ", f2(x)=" << f2(x) << '\n';
182     fout << "Количество итераций: " << iter << '\n';
183
184     x = iteration_system(phin, dphin, { 1, 1 }, { 2, 2 }, 0.00001);
185     fout << "Система методом простых итераций: x1=" << x[0] << ", x2=" << x[1] << ", f1(x)
        )=" << f1(x) << ", f2(x)=" << f2(x) << '\n';
186     fout << "Количество итераций: " << iter << '\n';
187 }

```