

**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**

## 2.1 Методы простой итерации и Ньютона

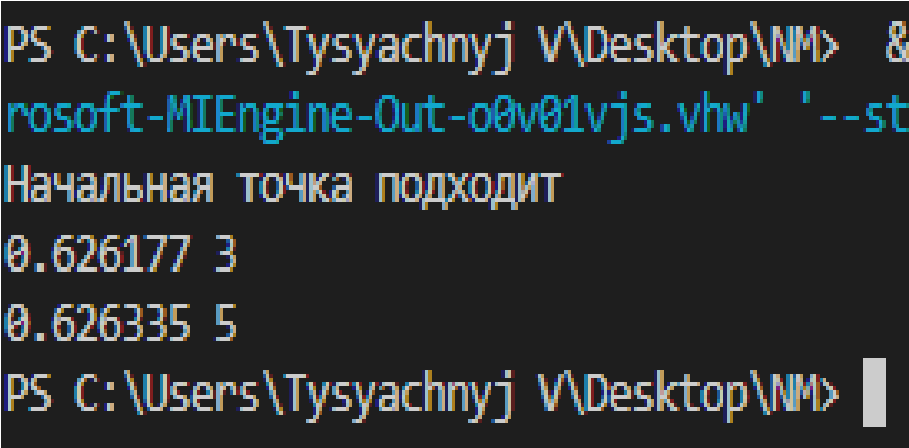
### 1 Постановка задачи

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: 25**

$$\sqrt{x+2} - 2\cos x = 0$$

### 2 Результаты работы



```
PS C:\Users\Tsyachnyj V\Desktop\NM> &
rossoft-MIEngine-Out-o0v01vjs.vhw' '--st
Начальная точка подходит
0.626177 3
0.626335 5
PS C:\Users\Tsyachnyj V\Desktop\NM>
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 | #include <string>
4 | #include <cmath>
5 |
6 | using namespace std;
7 |
8 | //
9 | double f(double x){
10 |     return pow(x + 2, 0.5) - 2 * cos(x);
11 | }
12 |
13 | //
14 | double df(double x){
15 |     return 0.5 * pow(x + 2, -0.5) + 2 * sin(x);
16 | }
17 |
18 | //
19 | double ddf(double x){
20 |     return -0.25 * pow(x + 2, -1.5) + 2 * cos(x);
21 | }
22 |
23 | //
24 | double phi(double x){
25 |     return acos(pow(x + 2, 0.5) / 2);
26 | }
27 |
28 | //
29 | double dphi(double x){
30 |     return -0.5 / pow(4 - x * x, 0.5);
31 | }
32 |
33 | int main(){
34 |
35 |     //
36 |     double x0 = 1;
37 |     double eps = 0.001;
38 |     if (f(x0) * ddf(x0) > 0){
39 |         cout << " " << "\n";
40 |
41 |         int k = 0;
42 |         double x[2] = {x0 - eps - 1, x0};
43 |         while (abs(x[1] - x[0]) >= eps){
44 |             x[0] = x[1];
45 |             x[1] -= f(x[1]) / df(x[1]);
46 |             k += 1;
47 |         }
```

```

48     cout << x[1] << " " << k << "\n";
49 }
50 else{
51     cout << " " << "\n";
52 }
53
54 //
55 // , q ,
56 int k = 0;
57 double q = abs(dphi(1));
58 double x[2] = {0., (0. + 1.) / 2};
59 while (q / (1 - q) * abs(x[1] - x[0]) > eps){
60     x[0] = x[1];
61     x[1] = phi(x[1]);
62     k += 1;
63 }
64 cout << x[1] << " " << k << "\n";
65
66 }

```

## 2.2 Методы простой итерации и Ньютона

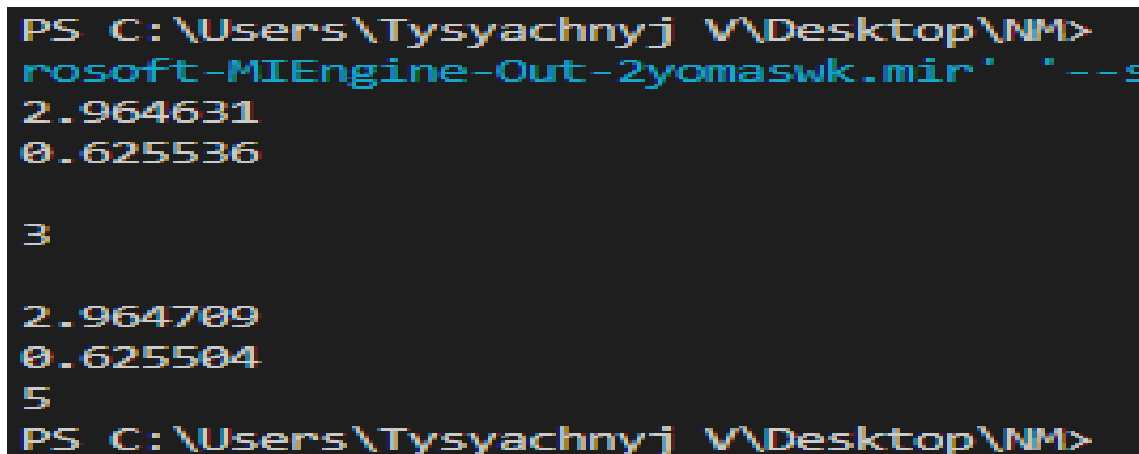
### 4 Постановка задачи

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: 25

$$\begin{cases} x_1^2 - x_2 + x_2^2 - 1 = 0 \\ x_1 - \sqrt{x_2 + 1} + 1 = 0 \end{cases}$$

### 5 Результаты работы



```
PS C:\Users\Tysyachnyj V\Desktop\NM>
rossoft-MIEngine-Out-2yomaswk.mir" '--s
2.964631
0.625536

3

2.964789
0.625584
5
PS C:\Users\Tysyachnyj V\Desktop\NM>
```

Рис. 2: Вывод программы в консоли

## 6 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <cmath>
5
6  using namespace std;
7
8
9
10 class matrix
11 {
12 private:
13     double **a;
14     int n, m;
15 public:
16     //
17     matrix(){
18         a = 0;
19         n = 0;
20         m = 0;
21     }
22
23     // NxM, E, ,
24     matrix (int N, int M, bool E = 0){
25         n = N;
26         m = M;
27         a = new double *[n];
28         for (int i = 0; i < n; ++ i){
29             a[i] = new double[m];
30             for (int j = 0; j < m; ++ j){
31                 a[i][j] = (i == j) * E;
32             }
33         }
34     }
35
36     //
37     int get_n_rows(){
38         return n;
39     }
40     int get_n_cols(){
41         return m;
42     }
43
44     double* operator [] (int index){
45         return getRow (index);
46     }
47 }
```

```

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

```

```

97         puts ("");
98     }
99 }
100
101 //
102 matrix mul (matrix & a, double k){
103     matrix c = matrix (a.get_n_rows (), a.get_n_cols ());
104     for (int i = 0; i < a.get_n_rows (); ++ i){
105         for (int j = 0; j < a.get_n_cols (); ++ j){
106             c[i][j] = a[i][j] * k;
107         }
108     }
109     return c;
110 }
111
112 //
113 matrix mul (matrix & a, matrix & b){
114     if (a.get_n_cols () != b.get_n_rows ()){
115         throw "Error";
116     }
117     matrix c = matrix (a.get_n_rows (), b.get_n_cols ());
118     for (int i = 0; i < a.get_n_rows (); ++ i){
119         for (int j = 0; j < b.get_n_cols (); ++ j){
120             for (int k = 0; k < a.get_n_cols (); ++ k){
121                 c[i][j] += a[i][k] * b[k][j];
122             }
123         }
124     }
125     return c;
126 }
127
128 matrix operator+ (matrix A, matrix B){
129     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
130         throw "Matrix size not matched";
131     }
132     matrix C = matrix(A.get_n_rows(), A.get_n_cols());
133     for (int i = 0; i < A.get_n_rows(); ++i){
134         for (int j = 0; j < A.get_n_cols(); ++j){
135             C[i][j] = A[i][j] + B[i][j];
136         }
137     }
138     return C;
139 }
140
141 matrix operator- (matrix A, matrix B){
142     if (A.get_n_rows() != B.get_n_rows() || A.get_n_cols() != B.get_n_cols()){
143         throw "Matrix size not matched";
144     }
145     matrix C = matrix(A.get_n_rows(), A.get_n_cols());

```



```

146     for (int i = 0; i < A.get_n_rows(); ++i){
147         for (int j = 0; j < A.get_n_cols(); ++j){
148             C[i][j] = A[i][j] - B[i][j];
149         }
150     }
151     return C;
152 }
153
154 //
155 matrix minor(matrix A, int a, int b){
156     int n = A.get_n_rows();
157     int m = A.get_n_cols();
158     matrix C = matrix (n - 1, m - 1);
159     int k = 0;
160     for (int i = 0; i < n; ++i){
161         if (i > a){
162             k = 1;
163         }
164         for (int j = 0; j < m; ++j){
165             if (i != a){
166                 if (j < b){
167                     C[i - k][j] = A[i][j];
168                 }
169                 else if (j > b){
170                     C[i - k][j - 1] = A[i][j];
171                 }
172             }
173         }
174     }
175     return C;
176 }
177
178 //
179 double det(matrix A){
180     double d = 0;
181     int n = A.get_n_rows();
182     if (n == 1){
183         return A[0][0];
184     }
185     else if (n == 2){
186         return A[0][0] * A[1][1] - A[0][1] * A[1][0];
187     }
188     else {
189         for (int k = 0; k < n; ++k) {
190             matrix M = matrix(n - 1, n - 1);
191             for (int i = 1; i < n; ++i) {
192                 int t = 0;
193                 for (int j = 0; j < n; ++j) {
194                     if (j == k)

```

```

195         continue;
196         M[i-1][t] = A[i][j];
197         t += 1;
198     }
199 }
200     d += pow(-1, k + 2) * A[0][k] * det(M);
201 }
202 }
203 return d;
204 }
205
206 //
207 matrix transpose(matrix A){
208     int n = A.get_n_rows();
209     int m = A.get_n_cols();
210     matrix C = matrix(m, n);
211     for (int i = 0; i < n; ++i){
212         for (int j = 0; j < m; ++j){
213             C[j][i] = A[i][j];
214         }
215     }
216     return C;
217 }
218
219 //
220 matrix inv(matrix A){
221     double d = det(A);
222     int n = A.get_n_rows();
223     matrix inv_A = matrix(n, n);
224     for(int i = 0; i < n; ++i){
225         for(int j = 0; j < n; ++j){
226             matrix M = matrix(n - 1, n - 1);
227             M = minor(A, i, j);
228             inv_A[i][j] = pow(-1.0, i + j + 2) * det(M) / d;
229         }
230     }
231     inv_A = transpose(inv_A);
232     return inv_A;
233 }
234
235 //
236 double Norm(matrix A){
237     double norm = 0;
238     for (int i = 0; i < A.get_n_rows(); ++i){
239         for (int j = 0; j < A.get_n_cols(); ++j){
240             norm += pow(A[i][j], 2);
241         }
242     }
243     return pow(norm, 0.5);

```

```

244 }
245
246 // ,      A
247 matrix Jacobian(matrix X, int k = -1){
248     matrix J = matrix(2, 2);
249     J[0][0] = 2 * X[0][0] * X[1][0];
250     J[0][1] = X[0][0] * X[0][0] + 4;
251     J[1][0] = 2 * X[0][0] - 2;
252     J[1][1] = 2 * X[1][0] - 2;
253     return J;
254 }
255
256 matrix F(matrix X){
257     matrix f = matrix(2,1);
258     f[0][0] = (pow(X[0][0], 2)+4)*X[1][0]-8;
259     f[1][0] = pow(X[0][0]-1, 2)+pow(X[1][0]-1, 2)-4;
260     return f;
261 }
262
263 //
264 matrix Phi(matrix X){
265     matrix phi = matrix(2,1);
266     phi[1][0] = 8 / (pow(X[0][0], 2) + 4);
267     phi[0][0] = pow(4 - pow(X[1][0]-1, 2), 0.5) + 1;
268     return phi;
269 }
270
271
272 int main()
273 {
274     //
275     double eps = 0.001;
276     matrix X[2] = {matrix(2, 1), matrix(2, 1)};
277     X[0][0][0] = 2.8; X[0][1][0] = 0.6;
278     X[1][0][0] = 3.0; X[1][1][0] = 0.8;
279     int k = 0;
280     matrix J = matrix(2, 1);
281     matrix f = matrix(2, 1);
282     while (Norm(X[1] - X[0]) > eps){
283         X[0] = X[1];
284         J = inv(Jacobian(X[1]));
285         f = F(X[1]);
286         X[1] = X[1] - mul(J, f);
287         k += 1;
288     }
289     printMatrix(X[1]);
290     cout << "\n" << k << "\n\n";
291
292     //

```

```

293 // , q ,
294 k = 0;
295 X[0][0][0] = 2.8; X[0][1][0] = 0.6;
296 X[1][0][0] = 3.0; X[1][1][0] = 0.8;
297 double q = 0.4;
298 matrix phi = matrix(2, 1);
299 while (q / (1 - q) * Norm(X[1] - X[0]) > eps and k < 10){
300     X[0] = X[1];
301     X[1] = Phi(X[1]);
302     printMatrix(X[1]);
303     cout << "\n";
304     k += 1;
305 }
306 printMatrix(X[1]);
307 cout << k << "\n";
308
309 }

```