

**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: Батулин Е.А.
Преподаватель: Пивоваров Д.Е.
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

2.1 Методы простой итерации и Ньютона

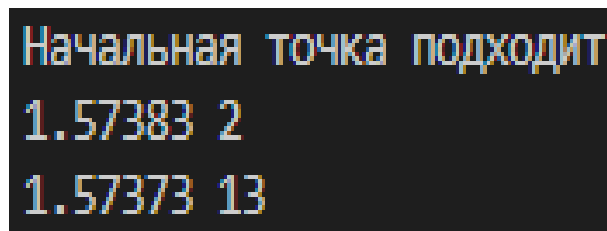
1 Постановка задачи

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: 1

$$2^x - x^2 - 0.5 = 0$$

2 Результаты работы



```
Начальная точка подходит  
1.57383 2  
1.57373 13
```

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  double initfunc(double x){
9      return pow(2, x) - pow(x, 2) - 0.5;
10 }
11
12 double dif(double x){
13     return pow(2, x) * log(2) - (2 * x);
14 }
15
16 double ddif(double x){
17     return pow(log(2), 2) * pow(2, x) - 2;
18 }
19
20 double eqf(double x){
21     return pow(pow(2, x) - 0.5, 0.5);
22 }
23
24 double deqf(double x){
25     return log(2) * pow(2, x - 1/2) / sqrt(pow(2, x + 1) - 1);
26 }
27
28 int main(){
29
30     //
31     double x0 = 1.6;
32     double eps = 0.001;
33     if (initfunc(x0) * ddif(x0) > 0){
34         cout << " " << "\n";
35
36         int k = 0;
37         double x[2] = {x0 - eps - 1, x0};
38         while (abs(x[1] - x[0]) >= eps){
39             x[0] = x[1];
40             x[1] -= initfunc(x[1]) / dif(x[1]);
41             k += 1;
42         }
43         cout << x[1] << " " << k << "\n";
44     }
45     else{
46         cout << " " << "\n";
47     }
```

```

48 ||
49 || //
50 || int k = 0;
51 || double q = abs(deqf(1.6));
52 || double x[2] = {1.5, (1.5 + 1.6) / 2};
53 || while (q / (1 - q) * abs(x[1] - x[0]) > eps){
54 ||     x[0] = x[1];
55 ||     x[1] = eqf(x[1]);
56 ||     k += 1;
57 || }
58 || cout << x[1] << " " << k << "\n";
59 ||
60 || }

```

2.2 Методы простой итерации и Ньютона

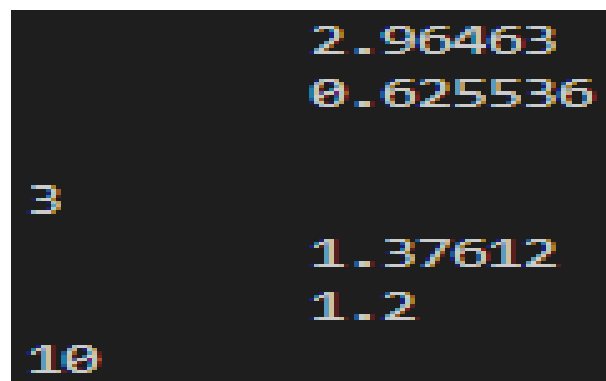
4 Постановка задачи

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: 1

$$\begin{cases} (x_1^2 + 4)x_2 - 8 = 0 \\ (x_1 - 1)^2 + (x_2 - 1)^2 - 4 = 0 \end{cases}$$

5 Результаты работы



```
2.96463
0.625536
3
1.37612
1.2
10
```

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 public:
12     int rows, cols;
13     double **a;
14
15     Matrix (){
16         rows = 0;
17         cols = 0;
18         a = new double*[rows];
19         for(int i = 0; i < rows; i++)
20             a[i] = new double[cols];
21     }
22
23     Matrix(int n, int m, bool identity = 0)
24     {
25         rows = n;
26         cols = m;
27         a = new double *[rows];
28         for (int i = 0; i < rows; i++){
29             a[i] = new double[cols];
30             for (int j = 0; j < cols; j++){
31                 a[i][j] = identity * (i == j);
32             }
33         }
34     }
35
36     double* operator[](int i)
37     {
38         return a[i];
39     }
40
41     double norm(){
42         double norm = 0;
43         for (int i = 0; i < rows; ++i){
44             for (int j = 0; j < cols; ++j){
45                 norm += pow(a[i][j], 2);
46             }
47         }
```

```

48     return pow(norm, 0.5);
49 }
50
51 void print()
52 {
53     for(int i = 0; i < rows; i++)
54     {
55         for(int j = 0; j < cols; j++)
56         {
57             cout << "\t" << a[i][j] << "\t";
58         }
59         cout << endl;
60     }
61 }
62 };
63
64 Matrix operator* (Matrix & A, double k){
65     Matrix C = Matrix (A.rows, A.cols);
66     for (int i = 0; i < A.rows; ++ i){
67         for (int j = 0; j < A.cols; ++ i){
68             C[i][j] = A[i][j] * k;
69         }
70     }
71     return C;
72 }
73
74 Matrix operator* (Matrix & A, Matrix & B){
75     Matrix C = Matrix (A.rows, B.cols);
76     for (int i = 0; i < A.rows; ++ i){
77         for (int j = 0; j < B.cols; ++ j){
78             for (int k = 0; k < A.cols; ++ k){
79                 C[i][j] += A[i][k] * B[k][j];
80             }
81         }
82     }
83     return C;
84 }
85
86 Matrix operator+ (Matrix A, Matrix B){
87     Matrix C = Matrix(A.rows, A.cols);
88     for (int i = 0; i < A.rows; ++i){
89         for (int j = 0; j < A.cols; ++j){
90             C[i][j] = A[i][j] + B[i][j];
91         }
92     }
93     return C;
94 }
95
96 Matrix operator- (Matrix A, Matrix B){

```

```

97     Matrix C = Matrix(A.rows, A.cols);
98     for (int i = 0; i < A.rows; ++i){
99         for (int j = 0; j < A.cols; ++j){
100             C[i][j] = A[i][j] - B[i][j];
101         }
102     }
103     return C;
104 }
105
106 double det(Matrix A){
107     double d = 0;
108     int n = A.rows;
109     if (n == 1){
110         return A[0][0];
111     }
112     else if (n == 2){
113         return A[0][0] * A[1][1] - A[0][1] * A[1][0];
114     }
115     else {
116         for (int k = 0; k < n; ++k) {
117             Matrix M = Matrix(n - 1, n - 1);
118             for (int i = 1; i < n; ++i) {
119                 int t = 0;
120                 for (int j = 0; j < n; ++j) {
121                     if (j == k)
122                         continue;
123                     M[i-1][t] = A[i][j];
124                     t += 1;
125                 }
126             }
127             d += pow(-1, k + 2) * A[0][k] * det(M);
128         }
129     }
130     return d;
131 }
132
133 double Norm(Matrix A){
134     double norm = 0;
135     for (int i = 0; i < A.rows; ++i){
136         for (int j = 0; j < A.cols; ++j){
137             norm += pow(A[i][j], 2);
138         }
139     }
140     return pow(norm, 0.5);
141 }
142
143 Matrix Minor(Matrix A, int a, int b){
144     int n = A.rows;
145     int m = A.cols;

```



```

146     Matrix C = Matrix(n - 1, m - 1);
147     int k = 0;
148     for (int i = 0; i < n; ++i){
149         if (i > a){
150             k = 1;
151         }
152         for (int j = 0; j < m; ++j){
153             if (i != a){
154                 if (j < b){
155                     C[i - k][j] = A[i][j];
156                 }
157                 else if (j > b){
158                     C[i - k][j - 1] = A[i][j];
159                 }
160             }
161         }
162     }
163     return C;
164 }
165
166 Matrix Transpose(Matrix A){
167     int n = A.rows;
168     int m = A.cols;
169     Matrix C = Matrix (m, n);
170     for (int i = 0; i < n; ++ i){
171         for (int j = 0; j < m; ++ j){
172             C[j][i] = A[i][j];
173         }
174     }
175     return C;
176 }
177
178 Matrix Inverse(Matrix A){
179     double d = det(A);
180     int n = A.rows;
181     Matrix inv_A = Matrix(n, n);
182     for(int i = 0; i < n; ++i){
183         for(int j = 0; j < n; ++j){
184             Matrix M = Matrix(n - 1, n - 1);
185             M = Minor(A, i, j);
186             inv_A[i][j] = pow(-1.0, i + j + 2) * det(M) / d;
187         }
188     }
189     inv_A = Transpose(inv_A);
190     return inv_A;
191 }
192
193 Matrix Jacobian(Matrix X, int k = -1)
194 {

```

```

195     Matrix J = Matrix(2, 2);
196     J[0][0] = 2 * X[0][0] * X[1][0];
197     J[0][1] = X[0][0] * X[0][0] + 4;
198     J[1][0] = 2 * X[0][0] - 2;
199     J[1][1] = 2 * X[1][0] - 2;
200     if (k != -1){
201         J[k][0] = (pow(X[0][0], 2) + 4) * X[1][0] - 8;
202         J[k][1] = pow((X[0][0] - 1), 2) + pow((X[1][0] - 1), 2) - 4;
203     }
204     return J;
205 }
206
207 Matrix F(Matrix X){
208     Matrix f = Matrix(2,1);
209     f[0][0] = (pow(X[0][0], 2)+4)*X[1][0]-8;
210     f[1][0] = pow(X[0][0]-1, 2)+pow(X[1][0]-1, 2)-4;
211     return f;
212 }
213
214 Matrix EqF(Matrix X){
215     Matrix eqf = Matrix(2,1);
216     //double x1 = X[0][0];
217     //double x2 = X[1][0];
218     eqf[0][0] = 8 / (pow(X[0][0], 2) + 4);
219     eqf[1][0] = pow(4 - pow(X[1][0]-1, 2), 0.5) + 1;
220     return eqf;
221 }
222
223 int main()
224 {
225     //
226     double eps = 0.001;
227     Matrix X[2] = {Matrix(2, 1), Matrix(2, 1)};
228     X[0][0][0] = 2.8; X[0][1][0] = 0.6;
229     X[1][0][0] = 3.0; X[1][1][0] = 0.8;
230     int k = 0;
231     Matrix J = Matrix(2, 1);
232     Matrix f = Matrix(2, 1);
233     while (Norm(X[1] - X[0]) > eps){
234         X[0] = X[1];
235         J = Inverse(Jacobian(X[1]));
236         f = F(X[1]);
237         X[1] = X[1] - (J * f);
238         k += 1;
239     }
240     X[1].print();
241     cout << "\n" << k << "\n";
242
243     //

```

```

244     k = 0;
245     X[0][0][0] = 2.8; X[0][1][0] = 0.6;
246     X[1][0][0] = 3.0; X[1][1][0] = 0.8;
247     double q = 0.4;
248     Matrix eqf = Matrix(2, 1);
249     while (q / (1 - q) * Norm(X[1] - X[0]) > eps and k < 10){
250         X[0] = X[1];
251         X[1] = EqF(X[1]);
252         //X[1].print();
253         //cout << "\n";
254         k += 1;
255     }
256     X[1].print();
257     cout << k << "\n";
258
259 }

```