

**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: Т. В. Мохнач
Преподаватель: Д. Е. Пивоваров
Группа: М8О-303Б-21
Дата:
Оценка:
Подпись:

Москва, 2024

1 Методы простой итерации и Ньютона

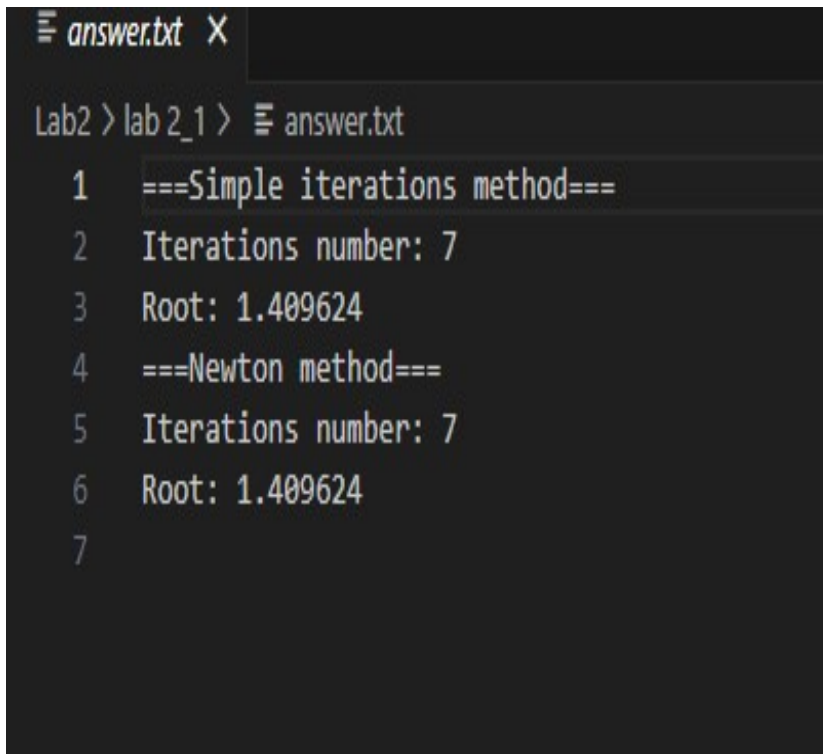
1 Постановка задачи

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: 15

$$\sin x - 2x^2 + 1 = 0$$

2 Результаты работы



```
Lab2 > lab 2_1 > answer.txt
1  ===Simple iterations method===
2  Iterations number: 7
3  Root: 1.409624
4  ===Newton method===
5  Iterations number: 7
6  Root: 1.409624
7
```

Рис. 1: Вывод программы

3 Исходный код

```
1 #include <cmath>
2 #include <algorithm>
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 using namespace std;
7
8
9 double F(double x) {
10     return sin(x) - pow(x,2) + 1; //sin x - x^2 + 1
11 }
12
13 double Diff_F(double x) {
14     return cos(x) - 2*x; //cos x - 2x
15 }
16
17 double Diff2_F(double x) {
18     return -sin(x) - 2; //-sin x - 2
19 }
20
21 double Phi(double x) {
22     return pow(1+sin(x), 0.5); //sqrt(1+sin x)
23 }
24
25 double Diff_Phi(double x) {
26     return cos(x)/pow(1+sin(x), 0.5); //cos(x)/sqrt(1+sin x)
27 }
28
29 double Iterations_method(double x0, double eps, int &i) {
30     int k = 0;
31     double q = abs(Diff_Phi(0.75)), x1 = x0-eps-1;
32     do{
33         x0 = x1;
34         x1 = Phi(x1);
35         i += 1;
36     }while (q / (1 - q) * abs(x1 - x0) > eps);
37     return x1;
38 }
39
40 double Newton_method(double x0, double eps, int &i) {
41     int k = 0;
42     double x1 = x0-eps-1;
43     do{
44         x0 = x1;
45         x1 = x0 - F(x1) / Diff_F(x1);
46         i += 1;
47     }while (abs(x1 - x0) >= eps);
48     return x1;
49 }
50
51 int main() {
52     //Начальные приближения подобраны графически исходя из условий выполнения методов
53     cout.precision(9);
54     ofstream fout("answer.txt");
55     ifstream fin("input.txt");
56     double eps;
57     fin >> eps;
58     double X_iterations, x0_iterations = 1.375;
59     int iterator_iterations = 0;
60     X_iterations = Iterations_method(x0_iterations, eps, iterator_iterations);
```

```

60 |     fout << "===Simple iterations method===\nIterations number: " << iterator_iterations << "\nRoot: " <<
    |         to_string(X_iterations) << '\n';
61 |     double X_Newton, x0_Newton = 1.7;
62 |     int iterator_Newton = 0;
63 |     X_Newton = Newton_method(x0_Newton, eps, iterator_Newton);
64 |     fout << "===Newton method===\nIterations number: " << iterator_iterations << "\nRoot: " << to_string(
    |         X_iterations) << '\n';
65 | }

```

2 Методы простой итерации и Ньютона решения систем нелинейных уравнений

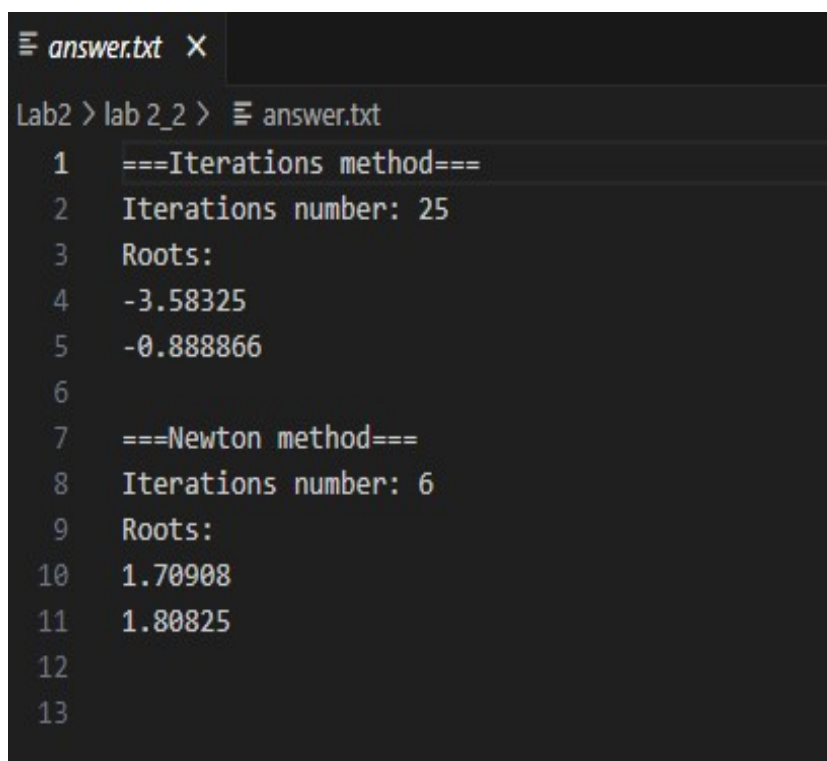
1 Постановка задачи

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: 15

$$\begin{cases} \frac{x_1^2}{16} + \frac{x_2^2}{4} - 1 = 0 \\ 4x_2 - e^{x_1} - x_1 = 0 \end{cases}$$

2 Результаты работы



```
≡ answer.txt X
Lab2 > lab 2_2 > ≡ answer.txt
1  ===Iterations method===
2  Iterations number: 25
3  Roots:
4  -3.58325
5  -0.888866
6
7  ===Newton method===
8  Iterations number: 6
9  Roots:
10 1.70908
11 1.80825
12
13
```

Рис. 2: Вывод программы

3 Исходный код

```
1  #include <cmath>
2  #include <algorithm>
3  #include <iostream>
4  #include <fstream>
5  #include <string>
6  #include <vector>
7  using namespace std;
8  class matrix
9  {
10     private:
11         vector <vector <double>> obj;
12     public:
13         int cols = 0, rows = 0;
14
15         matrix() {}
16         matrix(int _rows, int _cols)
17         {
18             rows = _rows;
19             cols = _cols;
20             obj = vector <vector <double>>(rows, vector <double>(cols));
21         }
22
23         operator double()
24         {
25             return obj[0][0];
26         }
27
28         vector <double>& operator[](int i)
29         {
30             return obj[i];
31         }
32
33         vector<double> getRow(int i){
34             return obj[i];
35         }
36         void swap (int I, int J){
37             for (int i = 0; i < cols; ++i){
38                 swap (obj[I][i], obj[J][i]);
39             }
40         }
41
42         matrix delete_column_row(int row,int coll) {
43             int d_r = 0;
44             matrix result = matrix (rows - 1, cols - 1);
45             for (int i = 0; i < rows; ++i){
46                 if (i > row) d_r = 1;
47                 int d_c = 0;
48                 for (int j = 0; j < cols; ++j){
49                     if (j > coll) d_c = 1;
50                     if (i != row && j != coll){
51                         result[i - d_r][j - d_c] = obj[i][j];
52                     }
53                 }
54             }
55             return result;
56         }
57
58         double get_determinant() {
59             double result = 0;
60             if (rows == 1){
```

```

61         return obj[0][0];
62     }
63     for (int k = 0; k < rows; ++k) {
64         matrix M = matrix(rows - 1, rows - 1);
65         for (int i = 1; i < rows; ++i) {
66             int t = 0;
67             for (int j = 0; j < rows; ++j) {
68                 if (j == k)
69                     continue;
70                 M[i-1][t] = obj[i][j];
71                 t += 1;
72             }
73         }
74         result += pow(-1, k + 2) * obj[0][k] * M.get_determinant();
75     }
76     return result;
77 }
78 double get_norm() {
79     double result = 0;
80     for (int i = 0; i < rows; ++i){
81         for (int j = 0; j < cols; ++j){
82             result += pow(obj[i][j], 2);
83         }
84     }
85     return pow(result, 0.5);
86 }
87 };
88
89
90
91 //Определение**** операторов****//
92 istream& operator>>(istream& stream, matrix& m)
93 {
94     for (int i = 0; i < m.rows; i++)
95     {
96         for (int j = 0; j < m.cols; j++)
97             stream >> m[i][j];
98     }
99     return stream;
100 }
101
102 ostream& operator<<(ostream& stream, matrix m)
103 {
104     for (int i = 0; i < m.rows; i++)
105     {
106         for (int j = 0; j < m.cols; j++)
107             stream << m[i][j] << ' ';
108         stream << '\n';
109     }
110     return stream;
111 }
112
113 matrix operator*(double a, matrix b)
114 {
115     for (int i = 0; i < b.rows; i++)
116     {
117         for (int j = 0; j < b.cols; j++)
118             b[i][j] *= a;
119     }
120     return b;
121 }
122

```

```

123 matrix operator+(matrix a, matrix b)
124 {
125     if (a.rows != b.rows || b.cols != a.cols)
126         return matrix(0, 0);
127     matrix res(a.rows, a.cols);
128     for (int i = 0; i < b.rows; i++)
129     {
130         for (int j = 0; j < res.cols; j++)
131             res[i][j] = a[i][j] + b[i][j];
132     }
133     return res;
134 }
135
136 matrix operator-(matrix a, matrix b)
137 {
138     if (a.rows != b.rows || b.cols != a.cols)
139         return matrix(0, 0);
140     matrix res(a.rows, a.cols);
141     for (int i = 0; i < b.rows; i++)
142     {
143         for (int j = 0; j < res.cols; j++)
144             res[i][j] = a[i][j] - b[i][j];
145     }
146     return res;
147 }
148
149 matrix operator*(matrix a, matrix b)
150 {
151     if (a.cols != b.rows)
152         return matrix(0, 0);
153     matrix res(a.rows, b.cols);
154     for (int i = 0; i < res.rows; i++)
155     {
156         for (int j = 0; j < res.cols; j++)
157         {
158             res[i][j] = 0;
159             for (int k = 0; k < a.cols; k++)
160                 res[i][j] += a[i][k] * b[k][j];
161         }
162     }
163     return res;
164 }
165
166 matrix get_J(matrix X){
167     matrix J = matrix(2, 2);
168     J[0][0] = X[0][0]/8;
169     J[0][1] = X[1][0]/2;
170     J[1][0] = -exp(X[0][0]) - 1;
171     J[1][1] = 4;
172     return J;
173 }
174
175 matrix get_A1(matrix X){
176     matrix A = matrix(2, 2);
177     A[0][0] = pow(X[0][0], 2)/16 + pow(X[1][0], 2)/4 - 1;
178     A[0][1] = X[1][0]/2;
179     A[1][0] = 4*X[1][0] - exp(X[0][0]) - X[0][0];
180     A[1][1] = 4;
181     return A;
182 }
183
184 matrix get_A2(matrix X){

```



```

185     matrix A = matrix(2, 2);
186     A[0][0] = X[0][0]/8;
187     A[0][1] = pow(X[0][0], 2)/16 + pow(X[1][0], 2)/4 - 1;
188     A[1][0] = -exp(X[0][0]) - 1;
189     A[1][1] = 4*X[1][0] - exp(X[0][0]) - X[0][0];
190     return A;
191 }
192
193 matrix Newton_method(matrix X0, double eps, int &iterations) {
194     matrix X1 = matrix(2, 1);
195     X1[0][0] = 1; X1[1][0] = 1;
196     matrix dets_div_result = matrix(2, 1);
197     while ((X1 - X0).get_norm() > eps){
198         X0 = X1;
199         dets_div_result[0][0] = get_A1(X0).get_determinant()/get_J(X0).get_determinant();
200         dets_div_result[1][0] = get_A2(X0).get_determinant()/ get_J(X0).get_determinant();
201         X1 = X0 - dets_div_result;
202         iterations += 1;
203     }
204     return X1;
205 }
206
207 matrix Phi(matrix X){
208     matrix phi = matrix(2,1);
209     phi[0][0] = -pow (16 - X[1][0]*X[1][0]*4, 0.5);
210     phi[1][0] = (exp(X[0][0]) + X[0][0])/4;
211     return phi;
212 }
213
214 matrix Iterations_method(matrix X0, double eps, int &iterations) {
215     matrix X1 = matrix(2, 1);
216     X1[0][0] = -3; X1[1][0] = -0.1;
217     double q = 0.1;
218     matrix phi = matrix(2, 1);
219     matrix diff = X1 - X0;
220     do{
221         X0 = X1;
222         X1 = Phi(X0);
223         iterations += 1;
224         diff = X1 - X0;
225         if (iterations > 1000) break;
226     } while (q / (1 - q) * diff.get_norm() > eps);
227     return X1;
228 }
229
230 int main() {
231     cout.precision(9);
232     ofstream fout("answer.txt");
233     ifstream fin("input.txt");
234     double eps;
235     fin >> eps;
236     matrix X0_iterations(2,1);
237     X0_iterations[0][0] = -3.5; X0_iterations[1][0] = -1;
238     int iterations_iterations = 0;
239     matrix X_iterations = Iterations_method(X0_iterations, eps, iterations_iterations);
240     fout << "===Iterations method===\nIterations number: " << iterations_iterations << "\nRoots:\n" <<
        X_iterations << '\n';
241
242     int iterations_Newton = 0;
243     matrix X0_Newton(2,1);
244     X0_Newton[0][0] = 1.5; X0_Newton[1][0] = 1.5;
245     matrix X_Newton = Newton_method(X0_Newton, eps, iterations_Newton);

```

```
246 || fout << "===Newton method===\nIterations number: " << iterations_Newton << "\nRoots:\n" << X_Newton <<  
247 || '\n';  
247 || }
```