

**Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
ИТМО»**

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №2

Вариант 10

Студент: Крикунов Олег Евгеньевич

P3267

Преподаватель: Машина Екатерина Алексеевна

Оценка: _____

Подпись преподавателя: _____

1. Цели работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов

2. Описание метода, расчётные формулы

Метод Ньютона - функция $y = f(x)$ на отрезке $[a, b]$ заменяется касательной и в качестве приближенного значения корня $x^* = x_n$ принимается точка пересечения касательной с осью абсцисс.

$$x_1 = x_0 - h_0$$

$$h_0 = f(x_0) / \tan \alpha = f(x_0) / f'(x_0)$$

$$x_1 = x_0 - f(x_0) / f'(x_0)$$

Рабочая формула метода:

$$x_i = x_{i-1} - f(x_{i-1}) / f'(x_{i-1})$$

Критерий окончания итерационного процесса:

$$x_n - x_{n-1} \leq \varepsilon \text{ или } |f(x_n)| \leq \varepsilon$$

Приближенное значение корня: $x^* = x_n$

Метод половинного деления - начальный интервал изоляции корня делим пополам, получаем начальное приближение к корню: $x_0 = (a_0 + b_0) / 2$. Вычисляем $f(x_0)$. В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки: $[a_0, x_0]$ либо $[x_0, b_0]$. Другую половину отрезка $[a_0, b_0]$, на которой функция $f(x)$ знак не меняет, отбрасываем. Новый интервал вновь делим пополам, получаем очередное приближение к корню: $x_1 = (a_1 + b_1) / 2$. и т.д.

Рабочая формула метода: $x_i = (a_i + b_i) / 2$

Критерий окончания итерационного процесса: $|b_n - a_n| \leq \varepsilon$ или $|f(x_n)| \leq \varepsilon$.

Приближенное значение корня: $x^* = (a_n + b_n) / 2$ или $x^* = a_n$ или $x^* = b_n$

Метод секущих - упростим метод Ньютона, заменив $f'(x)$ разностным приближением:

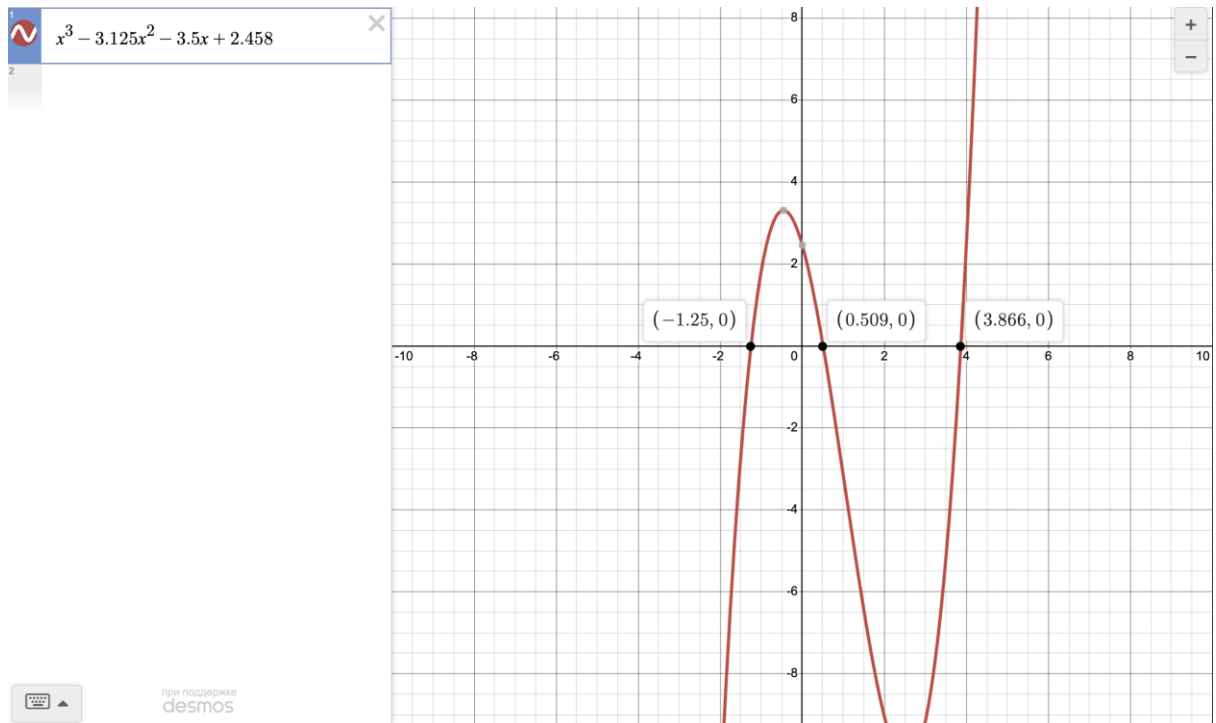
$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Рабочая формула метода: $x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$, $i = 1, 2, \dots$

Метод секущих является двухшаговым, т.е. новое приближение x_{i+1} определяется двумя предыдущими итерациями x_i и x_{i-1} . Выбор x_0 определяется как и в методе Ньютона, x_1 выбирается рядом с начальным самостоятельно. Критерий окончания итерационного процесса: $x_n - x_{n-1} \leq \varepsilon$ или $|f(x_n)| \leq \varepsilon$

3. Вычислительная часть

1) Графическое отделение корней уравнения:



2) Интервалы изоляции корней:

Крайний левый $[-1.5, -1]$, средний $[0, 1]$, крайний правый $[3.5, 4]$

3) Нахождение крайнего правого корня методом Ньютона:

- Проверим условие сходимости метода Ньютона:

1. Исходя из графика функции и ее типа (алгебраический, без делений на x) делаем вывод, что наша функция – всюду определена, непрерывна на отрезке $[3, 4]$

2. На концах отрезка функция имеет разные знаки:

$$f(3) = 3^3 - 3.125 \cdot 3^2 - 3.5 \cdot 3 + 2.458 = -9.167$$

$$f(4) = 4^3 - 3.125 \cdot 4^2 - 3.5 \cdot 4 + 2.458 = 2.458$$

3. По графику видно, что производные $f'(x)$ и $f''(x)$ сохраняют знак на отрезке $[a; b] \rightarrow$ функция возрастает на промежутке и выпукла вниз

4. По графику видно, что на выбранном промежутке нет точек экстремума $\rightarrow f'(x) \neq 0$

- Выбор начального приближения x_0 :

Выбор начального приближения $x_0 \in [a; b]$:

Метод обеспечивает быструю *сходимость*, если выполняется условие:

$$f(x_0) \cdot f''(x_0) > 0$$

(тот конец интервала, для которого знаки функции и второй производной совпадают)

$$x_0 = \begin{cases} a_0, & \text{если } f(a_0) \cdot f''(a_0) > 0 \\ b_0, & \text{если } f(b_0) \cdot f''(b_0) > 0 \end{cases}$$

$$f'(x) = 3x^2 - 6.25x - 3.5$$

$$f''(x) = 6x - 6.25$$

Нам подошел вариант, что $x_0 = b_0$

№ итерации	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}	$ x_{k+1} - x_k $
1	4	2.458	19.5	3.873	0.129
2	3.873	0.0708	17.243	3.869	0.001

Таким образом мы нашли приближительный корень за 2 итерации.

4) Нахождение крайнего левого корня методом половинного деления

№ шага	a	b	x	F(a)	F(b)	F(x)	a-b
1	-2	-1	-1.5	-11.042	1.833	-2.698	1
2	-1.5	-1	-1.25	-2.698	1.833	-0.003	0.5

5) Нахождение среднего корня методом секущих

№ итерации	x_{k-1}	x_k	x_{k+1}	$f(x_{k+1})$	$ x_{k+1} - x_k $
1	1	0	0.436	0.421	0.436
2	0	0.436	0.526	-0.102	0.09
3	0.436	0.526	0.508	0.001	0,018

4. Программная часть

Листинг программы:

Функции и их производные в header файле:

```

1 #ifndef FUNCTIONS_H
2 #define FUNCTIONS_H
3 #include <cmath>
4
5 inline double F1(const double x) {
6     return pow(x, 3) - 3.125 * pow(x, 2) - 3.5 * x + 2.458;
7 }
8
9 inline double DF1(const double x) {
10    return 3 * pow(x, 2) - 6.25 * x - 3.5;
11 }
12
13 inline double DDF1(const double x) {
14    return 6 * x - 6.25;
15 }
16
17 inline double F2(const double x) {
18    return 2 * pow(x, 3) - 1.89 * pow(x, 2) - 5 * x + 2.34;
19 }
20
21 inline double DF2(const double x) {
22    return 6 * pow(x, 2) - 3.78 * x - 5;

```

```

23 }
24
25 inline double DDF2(const double x) {
26     return 12 * x - 3.78;
27 }
28
29 inline double F3(const double x) {
30     return exp(x) - 3;
31 }
32
33 inline double DF3(const double x) {
34     return exp(x);
35 }
36
37 inline double DDF3(const double x) {
38     return exp(x);
39 }
40
41 #endif //FUNCTIONS_H

```

Метод Ньютона в header файле:

```

1  #ifndef NEWTON_METHOD_H
2  #define NEWTON_METHOD_H
3
4  #include <vector>
5  #include "Functions.h"
6
7  class NewtonMethod {
8  public:
9      static double FindRoot(const std::vector<double>&segment, short
10 functionChoice) {
11         double x0 = 0;
12
13         switch (functionChoice) {
14             case 1:
15                 if (F1(segment[0]) * DDF1(segment[0]) > 0) {
16                     x0 = segment[0];
17                 }
18                 else if (F1(segment[1]) * DDF1(segment[1]) > 0) {
19                     x0 = segment[1];
20                 }
21                 else if (F1(segment[0]) * DDF1(segment[0]) < 0 && F1(segment[1]) *
22 DDF1(segment[1]) < 0) {
23                     x0 = std::abs(segment[0] + segment[1]) / 2.0;
24                 }
25                 else {
26                     throw std::invalid_argument("Невозможно выбрать начальное
27 приближение!");
28                 }
29                 break;
30             case 2:
31                 if (F2(segment[0]) * DDF2(segment[0]) > 0) {
32                     x0 = segment[0];
33                 }
34                 else if (F2(segment[1]) * DDF2(segment[1]) > 0) {
35                     x0 = segment[1];
36                 }
37                 else if (F2(segment[0]) * DDF2(segment[0]) < 0 && F2(segment[1]) *
38 DDF2(segment[1]) < 0) {

```

```

39         x0 = std::abs(segment[0] + segment[1]) / 2.0;
40     }
41     else {
42         throw std::invalid_argument("Невозможно выбрать начальное
43 приближение!");
44     }
45     break;
46 case 3:
47     if (F3(segment[0]) * DDF3(segment[0]) > 0) {
48         x0 = segment[0];
49     }
50     else if (F3(segment[1]) * DDF3(segment[1]) > 0) {
51         x0 = segment[1];
52     }
53     else if (F3(segment[0]) * DDF3(segment[0]) < 0 && F3(segment[1]) *
54 DDF3(segment[1]) < 0) {
55         x0 = std::abs(segment[0] + segment[1]) / 2.0;
56     }
57     else {
58         throw std::invalid_argument("Невозможно выбрать начальное
59 приближение!");
60     }
61     break;
62     default: ;
63 }
64
65 return x0;
66 }
67
68 static double SolveEquation(const std::vector<double>&segment, const double
69 eps, const short functionChoice) {
70     double x = FindRoot(segment, functionChoice);
71     int counter = 0;
72
73     switch (functionChoice) {
74     case 1:
75         while (std::abs(F1(x)) > eps) {
76             x = x - (F1(x) / DF1(x));
77             counter++;
78         }
79         break;
80     case 2:
81         while (std::abs(F2(x)) > eps) {
82             x = x - (F2(x) / DF2(x));
83             counter++;
84         }
85         break;
86     case 3:
87         while (std::abs(F3(x)) > eps) {
88             x = x - (F3(x) / DF3(x));
89             counter++;
90         }
91         break;
92     default: ;
93 }
94
95 std::cout << "Полученное значение x: " << x << std::endl;
96 std::cout << "Количество итераций: " << counter << std::endl;
97 switch (functionChoice) {
98     case 1:
99         std::cout << "Значение функции в x:" << F1(x);
100        break;

```

```

101         case 2:
102             std::cout << "Значение функции в x:" << F2(x);
103             break;
104         case 3:
105             std::cout << "Значение функции в x:" << F3(x);
106             break;
            default: ;
        }

        return x;
    }
};

#endif // NEWTON_METHOD_H

```

Метод хорд в header файле:

```

1  #ifndef CHORDSMETHOD_H
2  #define CHORDSMETHOD_H
3
4  #include <vector>
5  #include "Functions.h"
6
7  class ChordsMethod {
8  public:
9      static double FindRoot(const std::vector<double>& segment, const short
10 functionChoice) {
11
12         double fa, fb;
13         switch (functionChoice) {
14             case 1:
15                 fa = F1(segment[0]);
16                 fb = F1(segment[1]);
17                 break;
18             case 2:
19                 fa = F2(segment[0]);
20                 fb = F2(segment[1]);
21                 break;
22             case 3:
23                 fa = F3(segment[0]);
24                 fb = F3(segment[1]);
25                 break;
26             default:
27                 throw std::invalid_argument("Невозможно посчитать начальное
28 приближение!");
29         }
30
31         if (fa * fb > 0) {
32             throw std::invalid_argument("Начальные точки на отрезке не обеспечивают
33 смены знака функции!");
34         }
35
36         return segment[0] - ((segment[1] - segment[0]) * fa / (fb - fa));
37     }
38
39     static double SolveEquation(std::vector<double>& segment, const double eps,
40 const short functionChoice) {
41         double x = FindRoot(segment, functionChoice);
42         int counter = 0;
43
44         switch (functionChoice) {

```

```

45     case 1:
46         while (std::abs(F1(x)) > eps) {
47             if (F1(segment[0]) * F1(x) < 0) {
48                 segment[1] = x;
49             } else if (F1(segment[1]) * F1(x) < 0) {
50                 segment[0] = x;
51             }
52             x = FindRoot(segment, functionChoice);
53             counter++;
54         }
55         break;
56     case 2:
57         while (std::abs(F2(x)) > eps) {
58             if (F2(segment[0]) * F2(x) < 0) {
59                 segment[1] = x;
60             } else if (F2(segment[1]) * F2(x) < 0) {
61                 segment[0] = x;
62             }
63             x = FindRoot(segment, functionChoice);
64             counter++;
65         }
66         break;
67     case 3:
68         while (std::abs(F3(x)) > eps) {
69             if (F3(segment[0]) * F3(x) < 0) {
70                 segment[1] = x;
71             } else if (F3(segment[1]) * F3(x) < 0) {
72                 segment[0] = x;
73             }
74             x = FindRoot(segment, functionChoice);
75             counter++;
76         }
77         break;
78     default: ;
79 }
80
81 std::cout << "Полученное значение x: " << x << std::endl;
82 std::cout << "Количество итераций: " << counter << std::endl;
83 switch (functionChoice) {
84     case 1:
85         std::cout << "Значение функции в x:" << F1(x);
86         break;
87     case 2:
88         std::cout << "Значение функции в x:" << F2(x);
89         break;
90     case 3:
91         std::cout << "Значение функции в x:" << F3(x);
92         break;
93     default: ;
94 }
95
96 return x;
97 }
98 };
99
100 #endif //CHORDSMETHOD_H

```

Метод простых итераций в header файле:

```

1 #ifndef SIMPLEITERATIONSMETHOD_H

```



```

2 #define SIMPLEITERATIONSMETHOD_H
3
4 #include <iostream>
5 #include <vector>
6 #include "Functions.h"
7
8
9 class SimpleIterationsMetgod {
10 public:
11     static double SolveEquation(const std::vector<double>&segment, const double
12 eps, const int functionChoice) {
13         double a = segment[0];
14         double b = segment[1];
15         double lm, prev_x, x;
16         int counter = 0;
17         switch (functionChoice) {
18             case 1:
19                 lm = -1 / std::max(DF1(a), DF1(b));
20                 if (1 + lm * DF1(a) > 1 || 1 + lm * DF1(b) > 1) {
21                     throw std::runtime_error("Условие сходимости не выполняется.");
22                 }
23                 prev_x = a;
24                 while (true) {
25                     x = prev_x + lm * F1(prev_x);
26                     counter++;
27                     if (std::abs(x - prev_x) < eps) {
28                         std::cout << "Корень уравнения: " << x << "\n";
29                         std::cout << "Количество итераций: " << counter <<
30 std::endl;
31                         return x;
32                     }
33                     prev_x = x;
34                 }
35             case 2:
36                 lm = -1 / std::max(DF2(a), DF2(b));
37                 if (1 + lm * DF2(a) > 1 || 1 + lm * DF2(b) > 1) {
38                     throw std::runtime_error("Условие сходимости не выполняется.");
39                 }
40                 prev_x = a;
41                 while (true) {
42                     x = prev_x + lm * F2(prev_x);
43                     counter++;
44                     if (std::abs(x - prev_x) < eps) {
45                         std::cout << "Корень уравнения: " << x << "\n";
46                         std::cout << "Количество итераций: " << counter <<
47 std::endl;
48                         return x;
49                     }
50                     prev_x = x;
51                 }
52             case 3:
53                 lm = -1 / std::max(DF3(a), DF3(b));
54                 if (1 + lm * DF3(a) > 1 || 1 + lm * DF3(b) > 1) {
55                     throw std::runtime_error("Условие сходимости не выполняется.");
56                 }
57                 prev_x = a;
58                 while (true) {
59                     x = prev_x + lm * F3(prev_x);
60                     counter++;
61                     if (std::abs(x - prev_x) < eps) {
62                         std::cout << "Корень уравнения: " << x << "\n";
63

```

```

64         std::cout << "Количество итераций: " << counter <<
65 std::endl;
66         return x;
67     }
68     prev_x = x;
69 }
70 default:
71     throw std::invalid_argument("Невозможно посчитать начальное
приближение!");
    }
};

#endif //SIMPLEITERATIONSMETHOD_H

```

Метод Ньютона для систем в header файле:

```

1  #ifndef MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H
2  #define MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H
3
4  #include <vector>
5  #include "Functions.h"
6
7  class SystemNewtonMethod {
8  public:
9      static std::vector<double> SolveEquation(double x0, double y0, double eps) {
10         double x = x0;
11         double y = y0;
12         double dx, dy;
13
14         short iterations = 0;
15         std::vector<double> errors;
16
17         dx = (-F4(x, y) * DF5_DY() + F5(x, y) * DF4_DY(y)) / (DF4_DX(x) * DF5_DY()
18 - DF5_DX(x) * DF4_DY(y));
19         dy = (-F5(x, y) * DF4_DX(x) + F4(x, y) * DF5_DX(x)) / (DF4_DX(x) * DF5_DY()
20 - DF5_DX(x) * DF4_DY(y));
21         x += dx;
22         y += dy;
23         iterations++;
24
25         while (abs(dx) > eps && abs(dy) > eps) {
26             dx = (-F4(x, y) * DF5_DY() + F5(x, y) * DF4_DY(y)) / (DF4_DX(x) *
27 DF5_DY() - DF5_DX(x) * DF4_DY(y));
28             dy = (-F5(x, y) * DF4_DX(x) + F4(x, y) * DF5_DX(x)) / (DF4_DX(x) *
29 DF5_DY() - DF5_DX(x) * DF4_DY(y));
30             x += dx;
31             y += dy;
32             iterations++;
33             errors.push_back(sqrt(dx * dx + dy * dy));
34         }
35
36         std::cout << "Количество итераций: " << iterations << std::endl;
37         std::cout << "Вектор погрешностей: ";
38         for (const auto& error : errors) {
39             std::cout << error << " ";
40         }
41         std::cout << std::endl;
42
43         std::vector<double> result = {x, y};

```

```

44         return result;
    }
};

#endif //MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H

```

Тело программы в main.cpp:

```

1  #include <iostream>
2  #include <fstream>
3  #include "NewtonMethod.h"
4  #include "ChordsMethod.h"
5  #include "SimpleIterationsMethod.h"
6
7  int main() {
8      short functionChoice;
9      std::cout << "Введите номер функции, которую хотите решить:" << "\n" <<
10         "1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458" << "\n" <<
11         "2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34" << "\n" <<
12         "3) F3(x) = e^x - 3" << "\n";
13      std::cin >> functionChoice;
14      if (functionChoice > 3 or functionChoice < 1 or typeid(functionChoice) !=
15          typeid(short)) {
16          throw std::invalid_argument("Неверный выбор функции!");
17      }
18
19      short choiceMethod;
20      std::cout << "Выберите способ решения:" << "\n" << "1) Метод Ньютона" << "\n"
21 << "2) Метод хорд" << "\n" <<
22         "3) Метод простой итерации" << "\n" << "4) Метод Ньютона" << "\n";
23      std::cin >> choiceMethod;
24      if (choiceMethod > 4 or choiceMethod < 1 or typeid(choiceMethod) !=
25          typeid(short)) {
26          throw std::invalid_argument("Неверный выбор метода!");
27      }
28
29      short dataInput;
30      std::cout << "Как бы вы хотели получить значения границы интервала, "
31         "начальное приближение к корню(в случае, где оно не высчитывается) и
32 погрешность вычисления?" << "\n" <<
33         "1 - вручную" << "\n" << "2 - из файла" << "\n";
34      std::cin >> dataInput;
35      if (dataInput > 2 or dataInput < 1 or typeid(dataInput) != typeid(short)) {
36          throw std::invalid_argument("Неверный выбор способа получения данных!");
37      }
38
39      std::vector<double> segment(2);
40      double EPS;
41      switch (dataInput) {
42          case 1:
43              std::cout << "Введите границу отрезка через пробел:" << "\n";
44              std::cin >> segment[0] >> segment[1];
45              if (segment[0] >= segment[1]) {
46                  throw std::invalid_argument("Начальное значение интервала должно
47 быть меньше конечного значения!");
48              }
49              switch (functionChoice) {
50                  case 1:
51                      if (F1(segment[0]) * F1(segment[1]) > 0) {
52

```

```

53         throw std::invalid_argument("На заданном интервале нет
54 корня!");
55     }
56     break;
57 case 2:
58     if (F2(segment[0]) * F2(segment[1]) > 0) {
59         throw std::invalid_argument("На заданном интервале нет
60 корня!");
61     }
62     break;
63 case 3:
64     if (F3(segment[0]) * F3(segment[1]) > 0) {
65         throw std::invalid_argument("На заданном интервале нет
66 корня!");
67     }
68     break;
69 default: throw std::invalid_argument("Невозможно проанализировать
70 функцию!");
71 }
72 std::cout << "Введите погрешность:" << "\n";
73 std::cin >> EPS;
74 if (EPS < 0 or typeid(EPS) != typeid(double)) {
75     throw std::invalid_argument("Некорректно введена погрешность!");
76 }
77 }
78
79 case 2:
80 {
81     std::ifstream inputFile("test.txt");
82     if (!inputFile.is_open()) {
83         throw std::runtime_error("Не удалось открыть файл!");
84     }
85     inputFile >> segment[0] >> segment[1] >> EPS;
86     inputFile.close();
87 }
88 break;
89 default: ;
90 }
91
92 switch (choiceMethod) {
93 case 1:
94     NewtonMethod::SolveEquation(segment, EPS, functionChoice);
95 break;
96 case 2:
97     ChordsMethod::SolveEquation(segment, EPS, functionChoice);
98 break;
99 case 3:
100    SimpleIterationsMetgod::SolveEquation(segment, EPS, functionChoice);
101 break;
102 default: ;
103 }
104 return

```

5. Примеры и результаты работы программы

Пример удачного выполнения программы при решении методом Ньютона:

The screenshot shows a C++ IDE with a source file `main.cpp` and a terminal window. The source code defines three functions: $F_1(x) = x^3 - 3.125x^2 - 3.5x + 2.458$, $F_2(x) = 2x^3 - 1.89x^2 - 5x + 2.34$, and $F_3(x) = e^x - 3$. The `main` function prompts the user to select a function and a method. In the terminal, the user selects function 1 and method 1 (Newton's method). The program successfully calculates the root of $F_1(x)$ as $x \approx -1.25065$ after 2 iterations.

```
#include <iostream>
#include <fstream>
#include "NewtonMethod.h"
#include "ChordsMethod.h"
#include "SimpleIterationsMethod.h"

int main() {
    short functionChoice;
    std::cout << "Введите номер функции, которую хотите решить:" << "\n" <<
        "1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458" << "\n" <<
        "2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34" << "\n" <<
        "3) F3(x) = e^x - 3" << "\n";
    std::cin >> functionChoice;
    if (functionChoice > 3 or functionChoice < 1 or typeid(functionChoice) != typeid(short))
        throw std::invalid_argument("Неверный выбор функции!");
    }

    short choiceMethod;
    std::cout << "Выберите способ решения:" << "\n" << "1) Метод Ньютона" << "\n" << "2) Метод хорд" << "\n" <<
        "3) Метод простой итерации" << "\n" << "4) Метод Ньютона" << "\n";
    std::cin >> choiceMethod;
    if (choiceMethod > 4 or choiceMethod < 1 or typeid(choiceMethod) != typeid(short)) {
        throw std::invalid_argument("Неверный выбор метода!");
    }

    short dataInput;
    std::cout << "Как бы вы хотели получить значения границы интервала, "
```

```
/Users/krknv/CLionProjects/untitled/lab_2/cmake-build...
Введите номер функции, которую хотите решить:
1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458
2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34
3) F3(x) = e^x - 3
1
Выберите способ решения:
1) Метод Ньютона
2) Метод хорд
3) Метод простой итерации
4) Метод Ньютона
1
Как бы вы хотели получить значения границы интервала:
1 - вручную
2 - из файла
1
Введите границу отрезка через пробел:
-1.5 -1
Введите погрешность:
0.01
Полученное значение x: -1.25065
Количество итераций: 2
Значение функции в x: -0.00881785
Process finished with exit code 0
```

Пример неудачного выполнения программы при решении методом Ньютона (ввели неправильный отрезок):

The screenshot shows the same C++ IDE as before, but the terminal output indicates a crash. The user entered `-1.5 -1` for the interval boundaries. The program terminated with exit code 134 due to an uncaught exception: `libc++abi: terminating due to uncaught exception of type std::invalid_argument: Неверный выбор метода!`. This occurs because the user selected method 4 (Newton's method) in the terminal, which is not a valid choice according to the program's logic.

```
#include <iostream>
#include <fstream>
#include "NewtonMethod.h"
#include "ChordsMethod.h"
#include "SimpleIterationsMethod.h"

int main() {
    short functionChoice;
    std::cout << "Введите номер функции, которую хотите решить:" << "\n" <<
        "1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458" << "\n" <<
        "2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34" << "\n" <<
        "3) F3(x) = e^x - 3" << "\n";
    std::cin >> functionChoice;
    if (functionChoice > 3 or functionChoice < 1 or typeid(functionChoice) != typeid(short))
        throw std::invalid_argument("Неверный выбор функции!");
    }

    short choiceMethod;
    std::cout << "Выберите способ решения:" << "\n" << "1) Метод Ньютона" << "\n" << "2) Метод хорд" << "\n" <<
        "3) Метод простой итерации" << "\n" << "4) Метод Ньютона" << "\n";
    std::cin >> choiceMethod;
    if (choiceMethod > 4 or choiceMethod < 1 or typeid(choiceMethod) != typeid(short)) {
        throw std::invalid_argument("Неверный выбор метода!");
    }

    short dataInput;
    std::cout << "Как бы вы хотели получить значения границы интервала, "
```

```
/Users/krknv/CLionProjects/untitled/lab_2/cmake-build...
Введите номер функции, которую хотите решить:
1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458
2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34
3) F3(x) = e^x - 3
1
Выберите способ решения:
1) Метод Ньютона
2) Метод хорд
3) Метод простой итерации
4) Метод Ньютона
1
Как бы вы хотели получить значения границы интервала:
1 - вручную
2 - из файла
1
Введите границу отрезка через пробел:
-1.5 -1
Введите погрешность:
0.01
libc++abi: terminating due to uncaught exception of type std::invalid_argument: Неверный выбор метода!
Process finished with exit code 134 (interrupted by keyboard signal)
```

Пример удачного выполнения программы при решении методом хорд:

```
#include <iostream>
#include <fstream>
#include "NewtonMethod.h"
#include "ChordsMethod.h"
#include "SimpleIterationsMethod.h"

int main() {
    short functionChoice;
    std::cout << "Введите номер функции, которую хотите решить:" << "\n" <<
        "1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458" << "\n" <<
        "2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34" << "\n" <<
        "3) F3(x) = e^x - 3" << "\n";
    std::cin >> functionChoice;
    if (functionChoice > 3 or functionChoice < 1 or typeid(functionChoice) != typeid(short))
        throw std::invalid_argument("Неверный выбор функции!");
    }

    short choiceMethod;
    std::cout << "Выберите способ решения:" << "\n" << "1) Метод Ньютона" << "\n" << "2) Метод хорд" << "\n" << "3) Метод простой итерации" << "\n" << "4) Метод Ньютона" << "\n";
    std::cin >> choiceMethod;
    if (choiceMethod > 4 or choiceMethod < 1 or typeid(choiceMethod) != typeid(short)) {
        throw std::invalid_argument("Неверный выбор метода!");
    }

    short dataInput;
    std::cout << "Как бы вы хотели получить значения границы интервала, "
```

```
/Users/krknv/CLionProjects/untitled/lab_2/cmake-bu
Введите номер функции, которую хотите решить:
1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458
2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34
3) F3(x) = e^x - 3
1
Выберите способ решения:
1) Метод Ньютона
2) Метод хорд
3) Метод простой итерации
4) Метод Ньютона
2
Как бы вы хотели получить значения границы интервала:
1 - вручную
2 - из файла
1
Введите границу отрезка через пробел:
3 4
Введите погрешность:
0.01
Полученное значение x: 3.86557
Количество итераций: 2
Значение функции в x:-0.00542373
Process finished with exit code 0
```

Пример неудачного выполнения программы методом хорд (отрезок – одна точка):

```
#include <iostream>
#include <fstream>
#include "NewtonMethod.h"
#include "ChordsMethod.h"
#include "SimpleIterationsMethod.h"

int main() {
    short functionChoice;
    std::cout << "Введите номер функции, которую хотите решить:" << "\n" <<
        "1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458" << "\n" <<
        "2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34" << "\n" <<
        "3) F3(x) = e^x - 3" << "\n";
    std::cin >> functionChoice;
    if (functionChoice > 3 or functionChoice < 1 or typeid(functionChoice) != typeid(short))
        throw std::invalid_argument("Неверный выбор функции!");
    }

    short choiceMethod;
    std::cout << "Выберите способ решения:" << "\n" << "1) Метод Ньютона" << "\n" << "2) Метод хорд" << "\n" << "3) Метод простой итерации" << "\n" << "4) Метод Ньютона" << "\n";
    std::cin >> choiceMethod;
    if (choiceMethod > 4 or choiceMethod < 1 or typeid(choiceMethod) != typeid(short)) {
        throw std::invalid_argument("Неверный выбор метода!");
    }

    short dataInput;
    std::cout << "Как бы вы хотели получить значения границы интервала, "
```

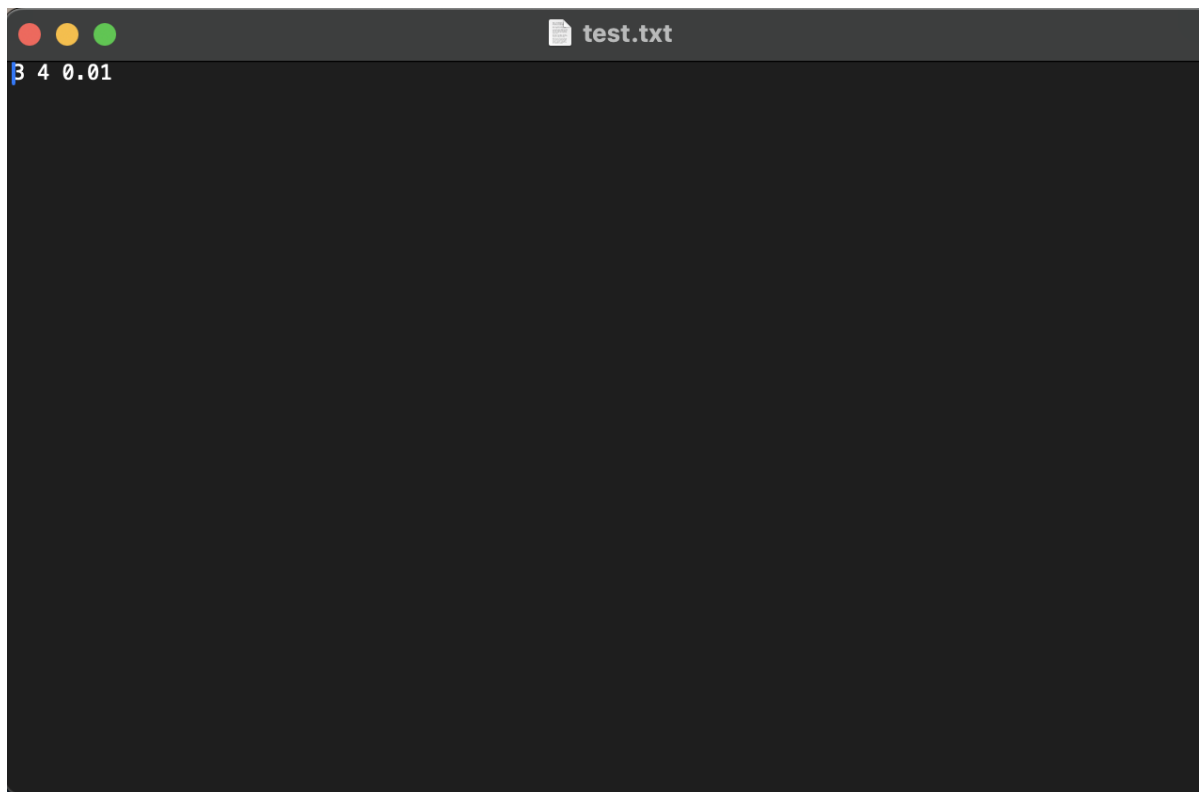
```
/Users/krknv/CLionProjects/untitled/lab_2/cmake-bu
Введите номер функции, которую хотите решить:
1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458
2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34
3) F3(x) = e^x - 3
1
Выберите способ решения:
1) Метод Ньютона
2) Метод хорд
3) Метод простой итерации
4) Метод Ньютона
2
Как бы вы хотели получить значения границы интервала:
1 - вручную
2 - из файла
1
Введите границу отрезка через пробел:
1 1
libc++abi: terminating due to uncaught exception o
Process finished with exit code 134 (interrupted by
```

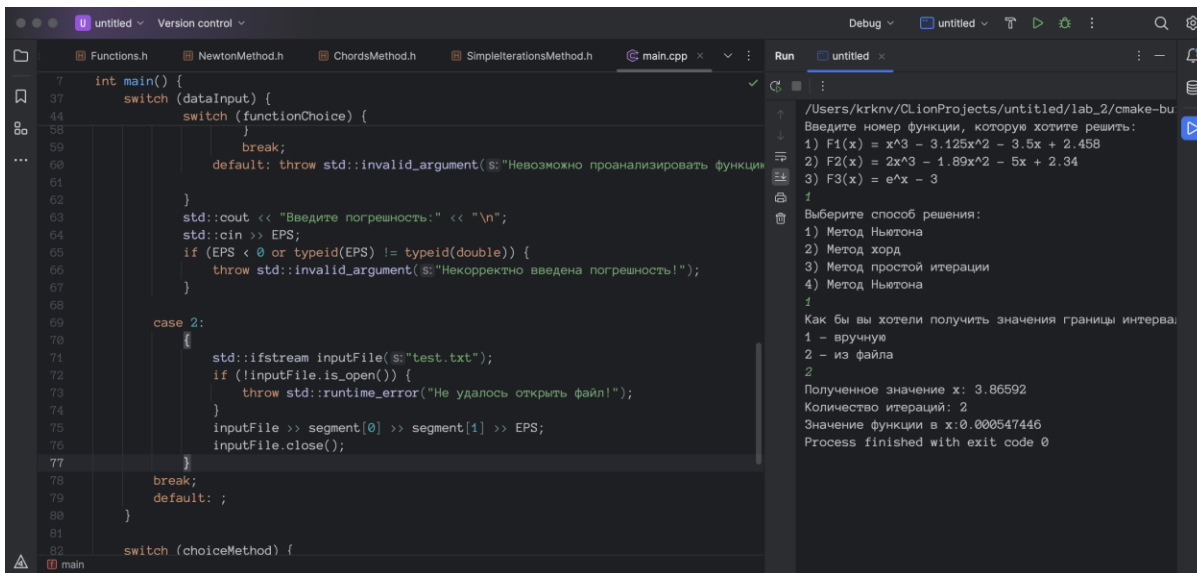
Пример удачного выполнения программы при решении методом простых итераций:

```
Functions.h NewtonMethod.h ChordsMethod.h SimpleIterationsMethod.h main.cpp Run untitled
1 #ifndef SIMPLEITERATIONSMETHOD_H
2 #define SIMPLEITERATIONSMETHOD_H
3
4 #include <iostream>
5 #include <vector>
6 #include "Functions.h"
7
8 class SimpleIterationsMetgod {
9 public:
10     static double SolveEquation(const std::vector<double>&segment, const double eps, const
11         double a = segment[0];
12         double b = segment[1];
13         double lm, prev_x, x;
14         int counter = 0;
15         switch (functionChoice) {
16             case 1:
17                 lm = -1 / std::max(abs(DF1(a)), abs(DF1(b)));
18                 if (1 + lm * DF1(a) > 1 || 1 + lm * DF1(b) > 1) {
19                     throw std::runtime_error("Условие сходимости не выполняется.");
20                 }
21                 prev_x = a;
22                 while (true) {
23                     x = prev_x + lm * F1(prev_x);
24                     counter++;
25                     if (std::abs(x - prev_x) < eps) {
26                         std::cout << "Корень уравнения: " << x << "\n";
27                         std::cout << "Значение функции в x: " << F1(x) << "\n";
28                     }
29                 }
30             case 2:
31                 // ...
32             case 3:
33                 // ...
34             case 4:
35                 // ...
36         }
37     }
38 };
39
40 #endif
```

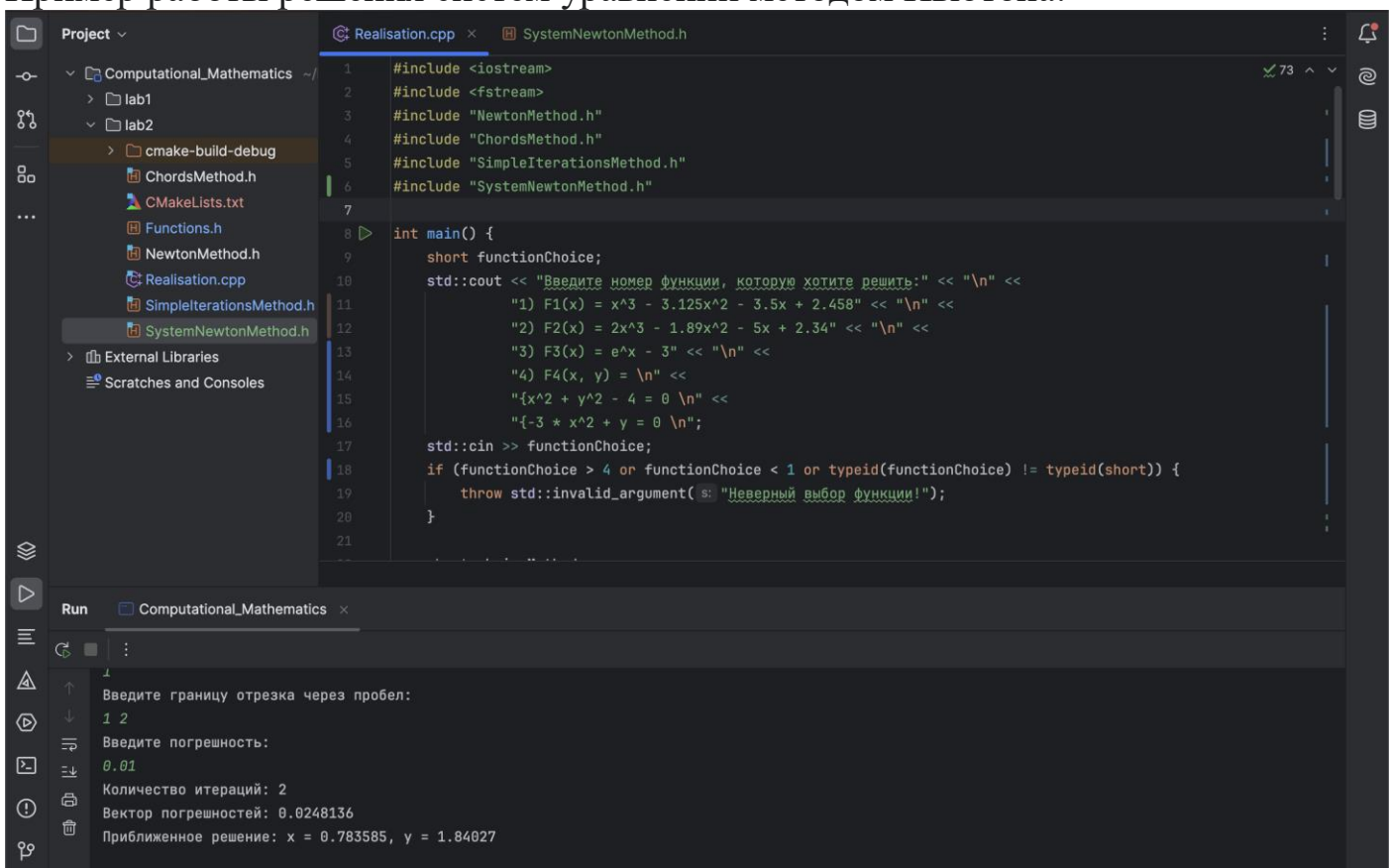
```
/Users/krknv/CLionProjects/untitled/lab_2/cmake-build...
Введите номер функции, которую хотите решить:
1) F1(x) = x^3 - 3.125x^2 - 3.5x + 2.458
2) F2(x) = 2x^3 - 1.89x^2 - 5x + 2.34
3) F3(x) = e^x - 3
4
Выберите способ решения:
1) Метод Ньютона
2) Метод хорд
3) Метод простой итерации
4) Метод Ньютона
3
Как бы вы хотели получить значения границы интервала:
1 - вручную
2 - из файла
1
Введите границу отрезка через пробел:
3 4
Введите погрешность:
0.01
Корень уравнения: 3.8656
Значение функции в x: -0.00494164
Количество итераций: 5
Process finished with exit code 0
```

Пример работы, в котором значения берутся из txt файла:





Пример работы решения систем уравнений методом Ньютона:



6. Вывод:

В ходе выполнения работы мы познакомились и научились решать нелинейные уравнения и их системы некоторыми численными способами