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.
$x1 = x0 - h0$

$h0 = f(x0) / \tan\alpha = f(x0) / f'(x0)$

$x1 = x0 - f(x0) / f'(x0)$

Рабочая формула метода:

$$\boldsymbol{xi = xi-1 - f\ xi-1\ /\ f'(xi-1)}$$

Критерий окончания итерационного процесса:

$x_n - x_{n-1} \le \varepsilon$ или $|\boldsymbol{f\ xn}| \le \boldsymbol{\varepsilon}$

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: [a0,x0] либо [b0,x0]. Другую половину отрезка [a0, b0], на которой функция f(x) знак не меняет, отбрасываем. Новый интервал вновь делим пополам, получаем очередное приближение к корню: x1=(a1+b1)/2. и т.д.

Рабочая формула метода: $\boldsymbol{xi = ai + bi\ /\ 2}$

Критерий окончания итерационного процесса: |bn – an|≤ ε или $\left|f(xn)\right| \le$ ε. Приближенное значение корня: $x* = an + bn\ /\ 2$ или $x* = an$ или $x* = bn$

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

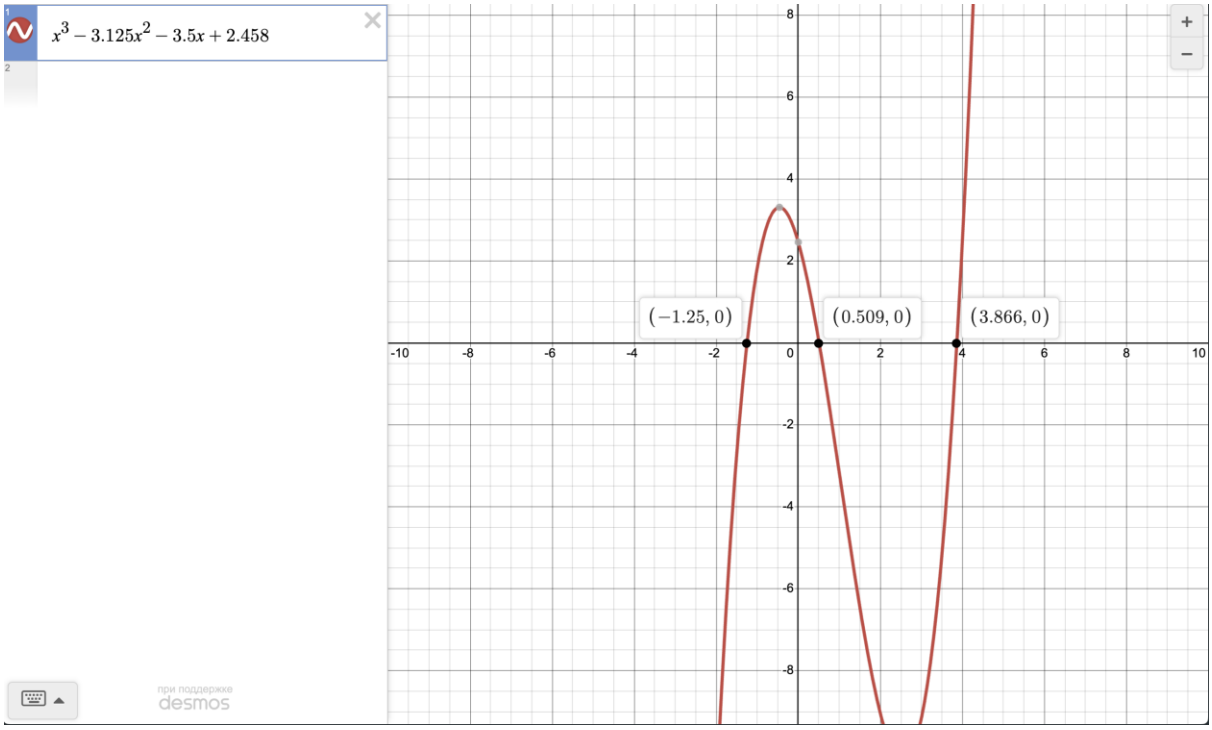
$f'(xi) \approx f\ xi - f(xi-1)\ /\ xi - xi-1$

Рабочая формула метода: $\boldsymbol{xi+1 = xi - xi - xi-1\ /(\ f(xi) - f(xi-1))\ *\ f\ xi},\ i = 1,2...$

Метод секущих является двухшаговым, т.е. новое приближение $xi+1$ определяется двумя предыдущими итерациями xi и $xi-1$. Выбор $x0$ определяется как и в методе Ньютона, $x1$- выбирается рядом с начальным самостоятельно. Критерий окончания итерационного процесса: $xn - xn-1 \le \varepsilon$ или $|\boldsymbol{f\ xn}| \le \boldsymbol{\varepsilon}$

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:

$f(3) = 3^3 - 3{,}125 * 3^2 - 3{,}5 * 3 + 2{,}458 = -9{,}167$

$f(4) = 4^3 - 3{,}125 * 4^2 - 3{,}5 * 4 + 2{,}458 = 2{,}458$

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

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

- 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}	$\lvert x_{k+1} - x_k \rvert$
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 приблизительный 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}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
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:

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H
#include <cmath>

inline double F1(const double x) {
    return pow(x, 3) - 3.125 * pow(x, 2) - 3.5 * x + 2.458;
}

inline double DF1(const double x) {
    return 3 * pow(x, 2) - 6.25 * x - 3.5;
}

inline double DDF1(const double x) {
    return 6 * x - 6.25;
}

inline double F2(const double x) {
    return 2 * pow(x, 3) - 1.89 * pow(x, 2) - 5 * x + 2.34;
}

inline double DF2(const double x) {
    return 6 * pow(x, 2) - 3.78 * x - 5;
```

```
}

inline double DDF2(const double x) {
    return 12 * x - 3.78;
}

inline double F3(const double x) {
    return exp(x) - 3;
}

inline double DF3(const double x) {
    return exp(x);
}

inline double DDF3(const double x) {
    return exp(x);
}

#endif //FUNCTIONS_H
```

Метод Ньютона в header файле:

```
 #ifndef NEWTON_METHOD_H
#define NEWTON_METHOD_H

#include <vector>
#include "Functions.h"

class NewtonMethod {
public:
    static double FindRoot(const std::vector<double>&segment, short
functionChoice) {
        double x0 = 0;

        switch (functionChoice) {
            case 1:
                if (F1(segment[0]) * DDF1(segment[0]) > 0) {
                    x0 = segment[0];
                }
                else if (F1(segment[1]) * DDF1(segment[1]) > 0) {
                    x0 = segment[1];
                }
                else if (F1(segment[0]) * DDF1(segment[0]) < 0 && F1(segment[1]) *
DDF1(segment[1]) < 0) {
                    x0 = std::abs(segment[0] + segment[1]) / 2.0;
                }
                else {
                    throw std::invalid_argument("Невозможно выбрать начальное
приближение!");
                }
                break;
            case 2:
                if (F2(segment[0]) * DDF2(segment[0]) > 0) {
                    x0 = segment[0];
                }
                else if (F2(segment[1]) * DDF2(segment[1]) > 0) {
                    x0 = segment[1];
                }
                else if (F2(segment[0]) * DDF2(segment[0]) < 0 && F2(segment[1]) *
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;
```

```
            case 2:
                std::cout << "Значение функции в x:" << F2(x);
                break;
            case 3:
                std::cout << "Значение функции в x:" << F3(x);
                break;
            default: ;
        }

        return x;
    }
};

#endif // NEWTON_METHOD_H
```

Метод хорд в header файле:

```
#ifndef CHORDSMETHOD_H
#define CHORDSMETHOD_H

#include <vector>
#include "Functions.h"

class ChordsMethod {
public:
    static double FindRoot(const std::vector<double>& segment, const short
functionChoice) {

        double fa, fb;
        switch (functionChoice) {
            case 1:
                fa = F1(segment[0]);
                fb = F1(segment[1]);
                break;
            case 2:
                fa = F2(segment[0]);
                fb = F2(segment[1]);
                break;
            case 3:
                fa = F3(segment[0]);
                fb = F3(segment[1]);
                break;
            default:
                throw std::invalid_argument("Невозможно посчитать начальное
приближение!");
        }

        if (fa * fb > 0) {
            throw std::invalid_argument("Начальные точки на отрезке не обеспечивают
смены знака функции!");
        }

        return segment[0] - ((segment[1] - segment[0]) * fa / (fb - fa));
    }

    static double SolveEquation(std::vector<double>& segment, const double eps,
const short functionChoice) {
        double x = FindRoot(segment, functionChoice);
        int counter = 0;

        switch (functionChoice) {
```

```
            case 1:
                while (std::abs(F1(x)) > eps) {
                    if (F1(segment[0]) * F1(x) < 0) {
                        segment[1] = x;
                    } else if (F1(segment[1]) * F1(x) < 0) {
                        segment[0] = x;
                    }
                    x = FindRoot(segment, functionChoice);
                    counter++;
                }
                break;
            case 2:
                while (std::abs(F2(x)) > eps) {
                    if (F2(segment[0]) * F2(x) < 0) {
                        segment[1] = x;
                    } else if (F2(segment[1]) * F2(x) < 0) {
                        segment[0] = x;
                    }
                    x = FindRoot(segment, functionChoice);
                    counter++;
                }
                break;
            case 3:
                while (std::abs(F3(x)) > eps) {
                    if (F3(segment[0]) * F3(x) < 0) {
                        segment[1] = x;
                    } else if (F3(segment[1]) * F3(x) < 0) {
                        segment[0] = x;
                    }
                    x = FindRoot(segment, functionChoice);
                    counter++;
                }
                break;
            default: ;
        }

        std::cout << "Полученное значение x: " << x << std::endl;
        std::cout << "Количество итераций: " << counter << std::endl;
        switch (functionChoice) {
            case 1:
                std::cout << "Значение функции в x:" << F1(x);
            break;
            case 2:
                std::cout << "Значение функции в x:" << F2(x);
            break;
            case 3:
                std::cout << "Значение функции в x:" << F3(x);
            break;
            default: ;
        }

        return x;
    }
};

#endif //CHORDSMETHOD_H
```

Метод простых итераций в header файле:

```
#ifndef SIMPLEITERATIONSMETHOD_H
```

```
#define SIMPLEITERATIONSMETHOD_H

#include <iostream>
#include <vector>
#include "Functions.h"


class SimpleIterationsMetgod {
public:
    static double SolveEquation(const std::vector<double>&segment, const double
eps, const int functionChoice) {
        double a = segment[0];
        double b = segment[1];
        double lm, prev_x, x;
        int counter = 0;
        switch (functionChoice) {
            case 1:
                lm = -1 / std::max(DF1(a), DF1(b));
                if (1 + lm * DF1(a) > 1 || 1 + lm * DF1(b) > 1) {
                    throw std::runtime_error("Условие сходимости не выполняется.");
                }
                prev_x = a;
                while (true) {
                    x = prev_x + lm * F1(prev_x);
                    counter++;
                    if (std::abs(x - prev_x) < eps) {
                        std::cout << "Корень уравнения: " << x << "\n";
                        std::cout << "Количество итераций: " << counter <<
std::endl;
                        return x;
                    }
                    prev_x = x;
                }
            case 2:
                lm = -1 / std::max(DF2(a), DF2(b));
                if (1 + lm * DF2(a) > 1 || 1 + lm * DF2(b) > 1) {
                    throw std::runtime_error("Условие сходимости не выполняется.");
                }
                prev_x = a;
                while (true) {
                    x = prev_x + lm * F2(prev_x);
                    counter++;
                    if (std::abs(x - prev_x) < eps) {
                        std::cout << "Корень уравнения: " << x << "\n";
                        std::cout << "Количество итераций: " << counter <<
std::endl;
                        return x;
                    }
                    prev_x = x;
                }
            case 3:
                lm = -1 / std::max(DF3(a), DF3(b));
                if (1 + lm * DF3(a) > 1 || 1 + lm * DF3(b) > 1) {
                    throw std::runtime_error("Условие сходимости не выполняется.");
                }
                prev_x = a;
                while (true) {
                    x = prev_x + lm * F3(prev_x);
                    counter++;
                    if (std::abs(x - prev_x) < eps) {
                        std::cout << "Корень уравнения: " << x << "\n";

```

```
                        std::cout << "Количество итераций: " << counter <<
std::endl;
                        return x;
                    }
                    prev_x = x;
                }
            default:
                throw std::invalid_argument("Невозможно посчитать начальное
приближение!");
        }
    }
};

#endif //SIMPLEITERATIONSMETHOD_H
```

Метод Ньютона для систем в header файле:

```
#ifndef MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H
#define MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H

#include <vector>
#include "Functions.h"

class SystemNewtonMethod {
public:
    static std::vector<double> SolveEquation(double x0, double y0, double eps) {
        double x = x0;
        double y = y0;
        double dx, dy;

        short iterations = 0;
        std::vector<double> errors;

        dx = (-F4(x, y) * DF5_DY() + F5(x, y) * DF4_DY(y)) / (DF4_DX(x) * DF5_DY()
- DF5_DX(x) * DF4_DY(y));
        dy = (-F5(x, y) * DF4_DX(x) + F4(x, y) * DF5_DX(x)) / (DF4_DX(x) * DF5_DY()
- DF5_DX(x) * DF4_DY(y));
        x += dx;
        y += dy;
        iterations++;

        while (abs(dx) > eps && abs(dy) > eps) {
            dx = (-F4(x, y) * DF5_DY() + F5(x, y) * DF4_DY(y)) / (DF4_DX(x) *
DF5_DY() - DF5_DX(x) * DF4_DY(y));
            dy = (-F5(x, y) * DF4_DX(x) + F4(x, y) * DF5_DX(x)) / (DF4_DX(x) *
DF5_DY() - DF5_DX(x) * DF4_DY(y));
            x += dx;
            y += dy;
            iterations++;
            errors.push_back(sqrt(dx * dx + dy * dy));
        }

        std::cout << "Количество итераций: " << iterations << std::endl;
        std::cout << "Вектор погрешностей: ";
        for (const auto& error : errors) {
            std::cout << error << " ";
        }
        std::cout << std::endl;

        std::vector<double> result = {x, y};
```

```
        return result;
    }
};

#endif //MATPLOTLIB_H_SYSTEMNEWTONMETHOD_H
```

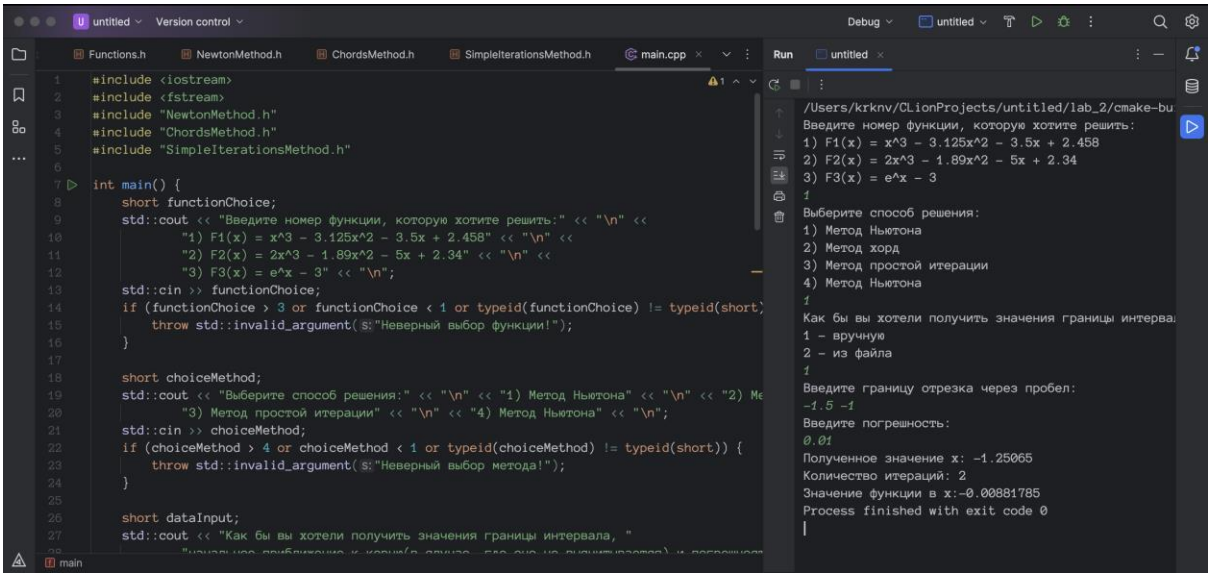
Тело программы в main.cpp:

```
#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, "
              "начальное приближение к корню(в случае, где оно не высчитывается) и
погрешность вычисления?" << "\n" <<
              "1 - вручную" << "\n" << "2 - из файла" << "\n";
    std::cin >> dataInput;
    if (dataInput > 2 or dataInput < 1 or typeid(dataInput) != typeid(short)) {
        throw std::invalid_argument("Неверный выбор способа получения данных!");
    }

    std::vector<double> segment(2);
    double EPS;
    switch (dataInput) {
        case 1:
            std::cout << "Введите границу отрезка через пробел:" << "\n";
            std::cin >> segment[0] >> segment[1];
            if (segment[0] >= segment[1]) {
                throw std::invalid_argument("Начальное значение интервала должно
быть меньше конечного значения!");
            }
            switch (functionChoice) {
                case 1:
                    if (F1(segment[0]) * F1(segment[1]) > 0) {
```

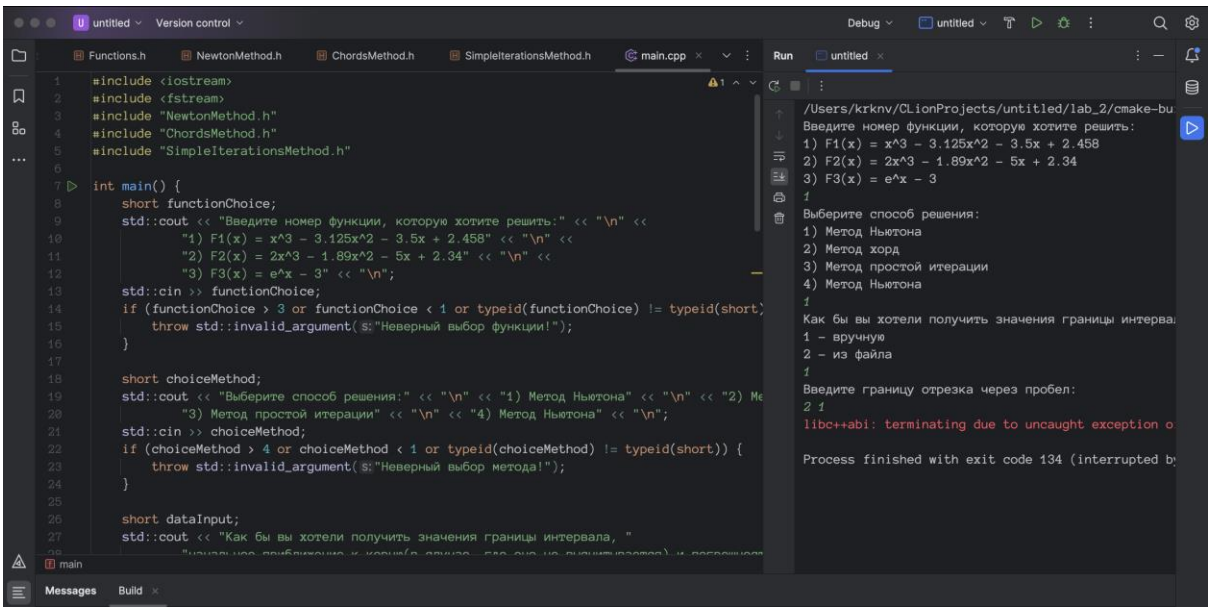
```
53                          throw std::invalid_argument("На заданном интервале нет
54 корня!");
55                      }
56                      break;
57                  case 2:
58                      if (F2(segment[0]) * F2(segment[1]) > 0) {
59                          throw std::invalid_argument("На 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              }
73              std::cout << "Введите погрешность:" << "\n";
74              std::cin >> EPS;
75              if (EPS < 0 or typeid(EPS) != typeid(double)) {
76                  throw std::invalid_argument("Некорректно введена погрешность!");
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;
            case 2:
                ChordsMethod::SolveEquation(segment, EPS, functionChoice);
            break;
            case 3:
                SimpleIterationsMetgod::SolveEquation(segment, EPS, functionChoice);
            break;
            default: ;
      }
      return
```

5. Примеры и результаты работы программы

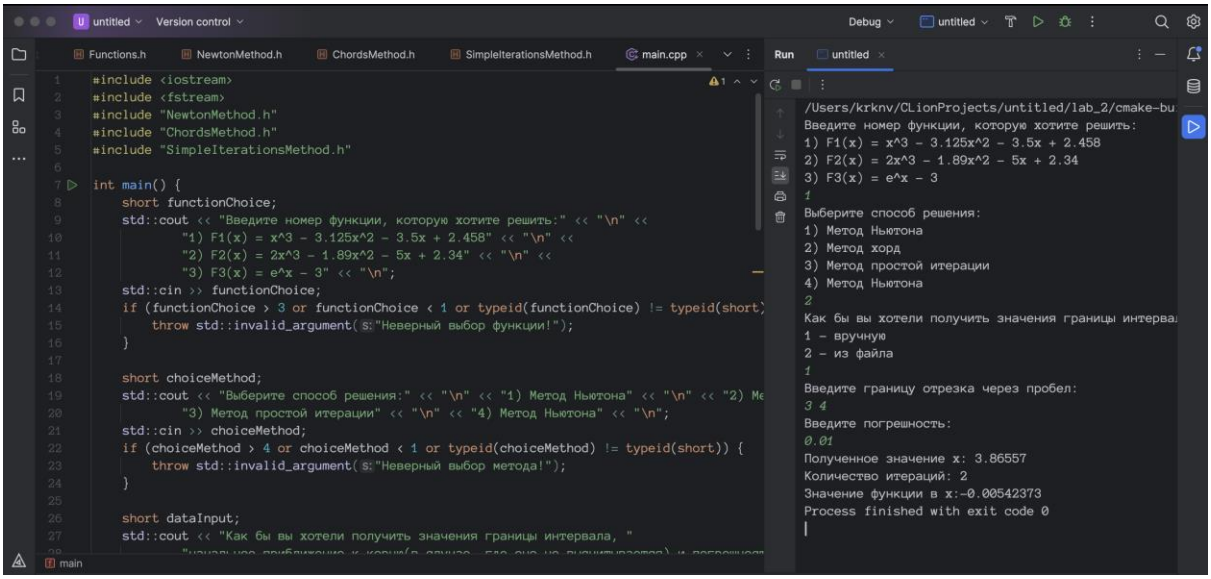
Пример удачного выполнения программы при решении методом Ньютона:



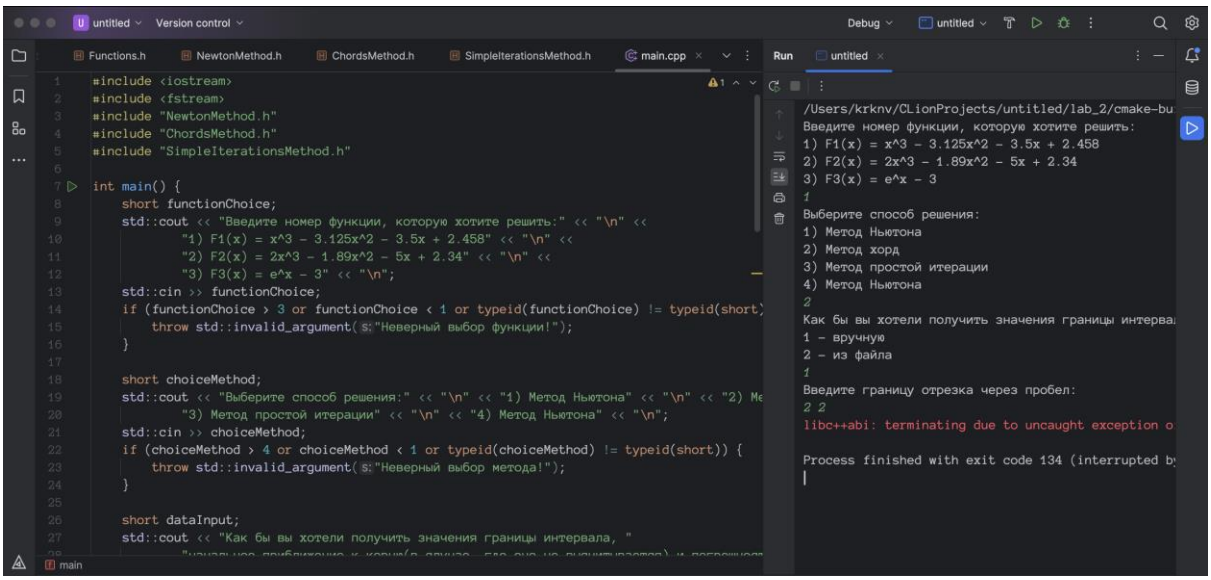
Пример неудачного выполнения программы при решении методом Ньютона (ввели неправильный отрезок):



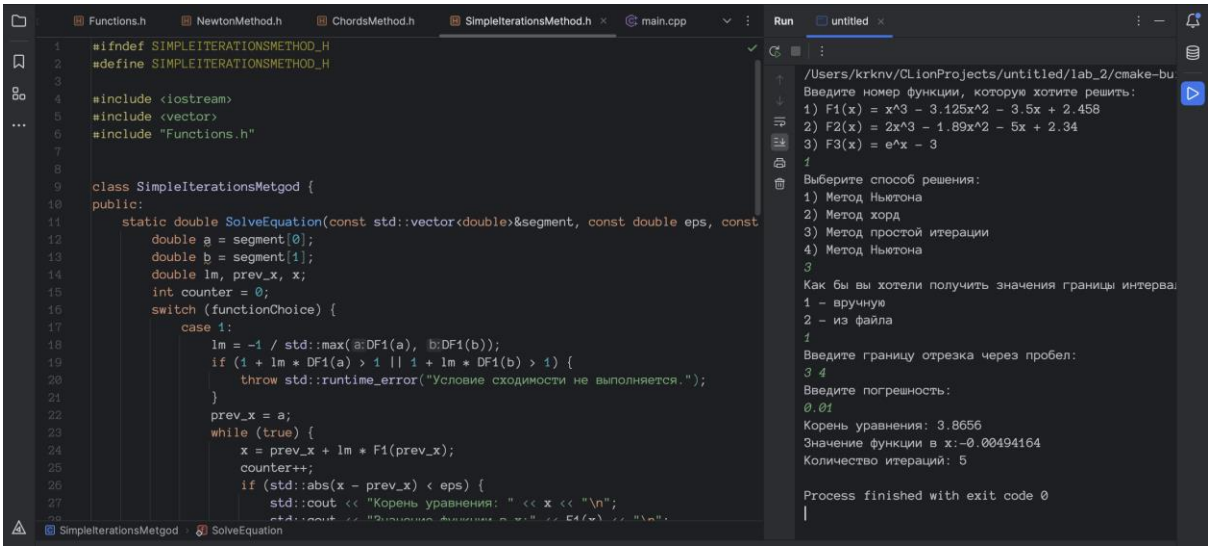
Пример удачного выполнения программы при решении методом хорд:



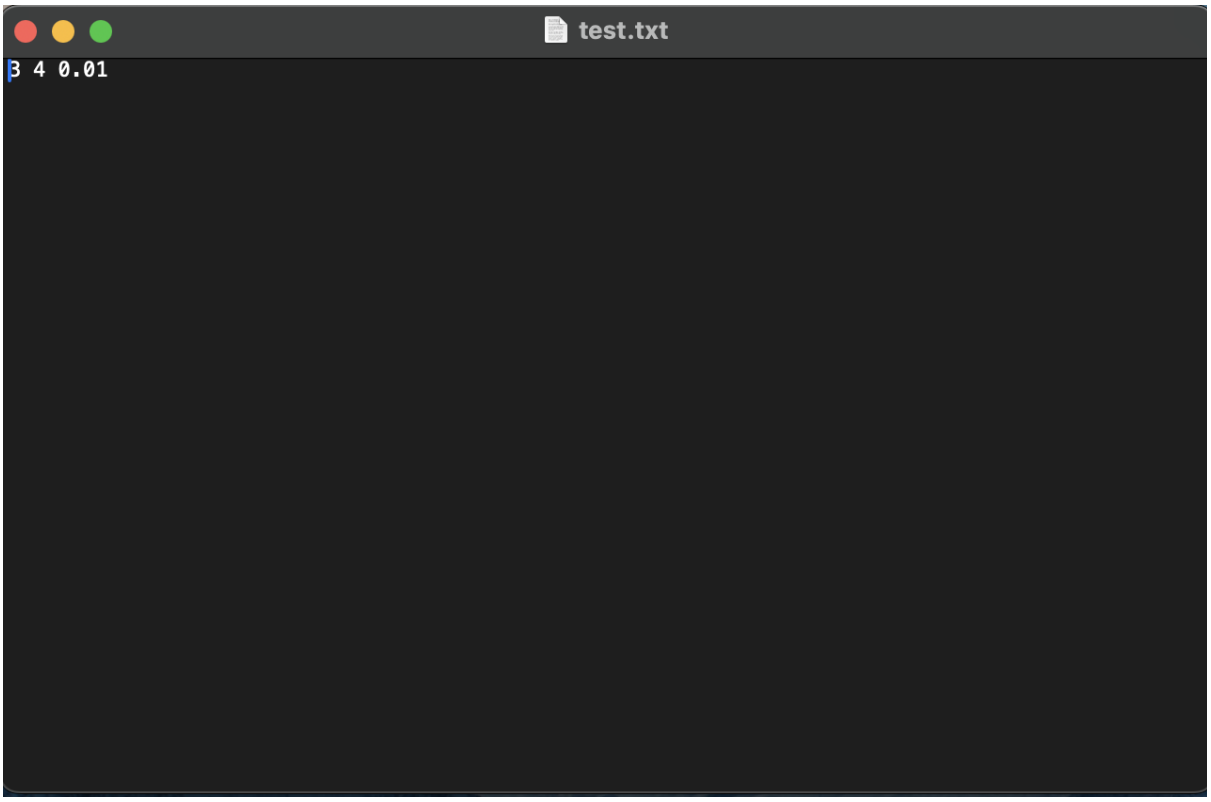
Пример неудачного выполнения программы методом хорд (отрезок – одна точка):

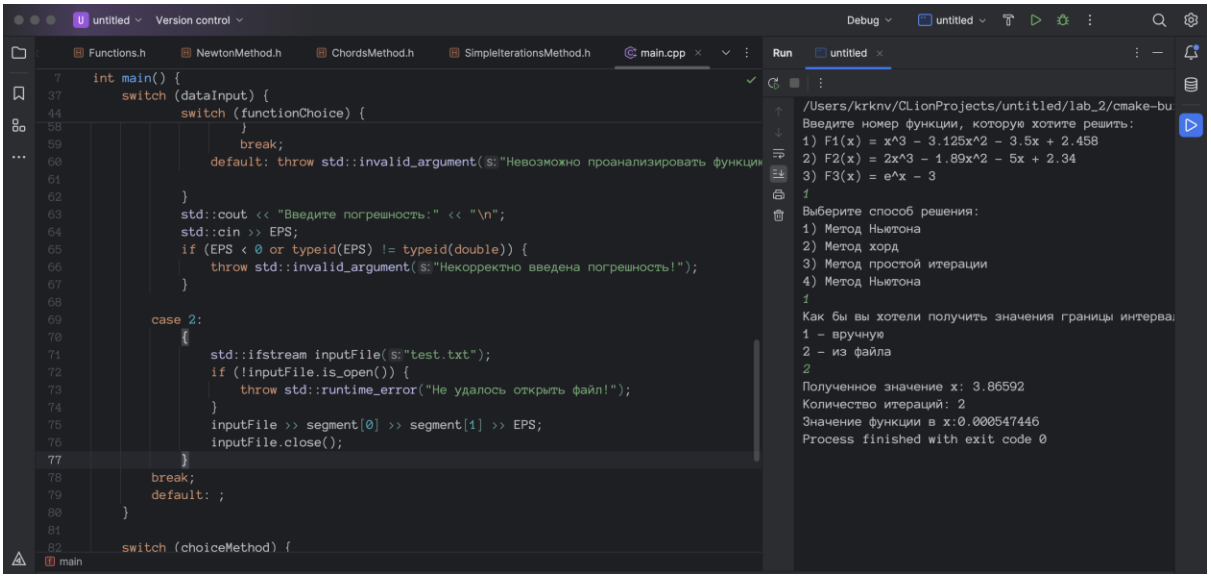


Пример удачного выполнения программы при решении методом простых итераций:

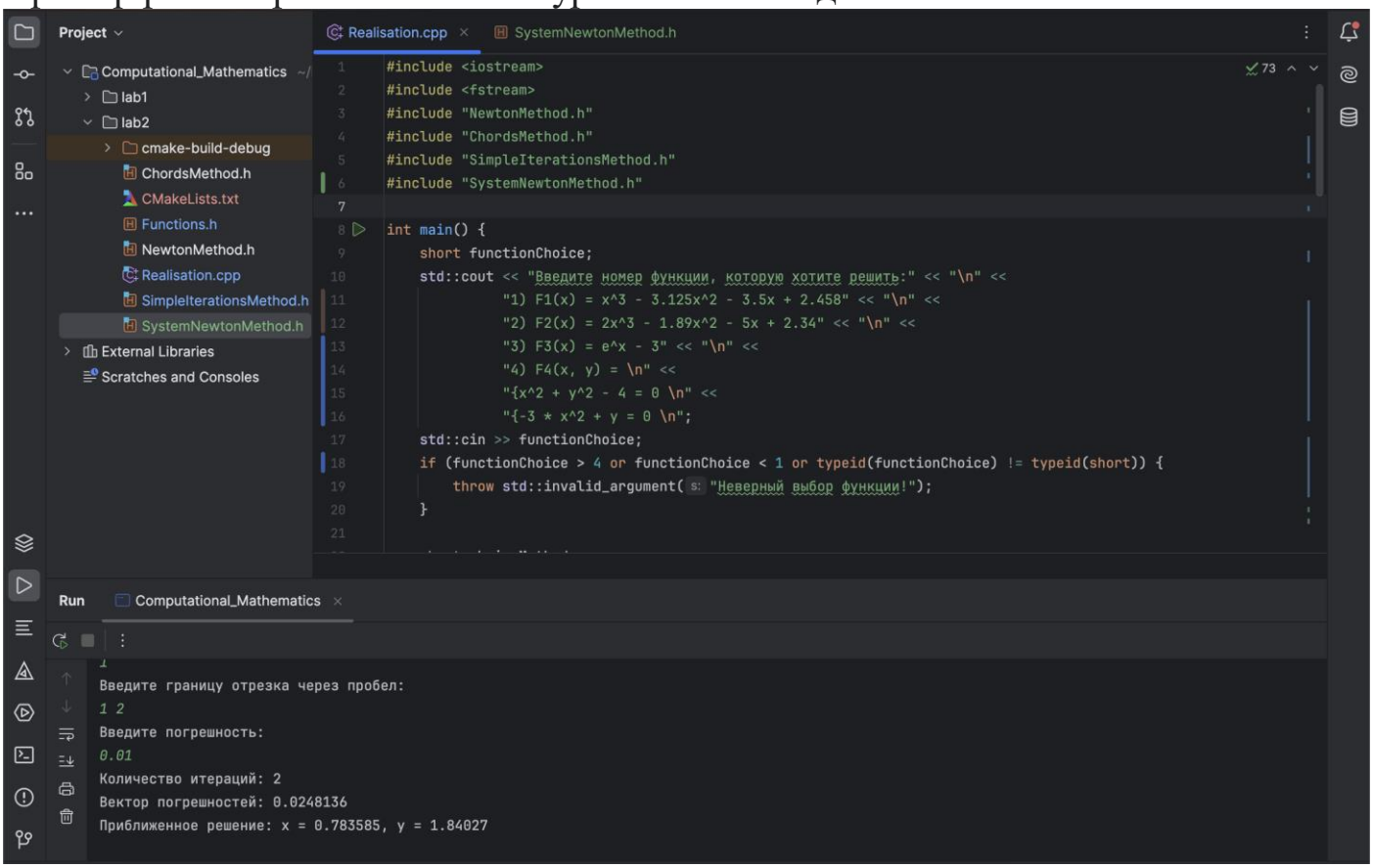


Пример работы, в котором значения берутся из txt файла:





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