Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

ОТЧЕТ

по лабораторной работе № 6

по курсу «Численные методы»

на тему: «Решение нелинейных уравнений и систем нелинейных уравнений методом Ньютона»

Вариант № 4

Студент ИУ9-61Б (Группа) _____ (Подпись, дата) Афанасьев И. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Домрачева А. Б. (И. О. Фамилия)

2024 г.

1 Постановка задачи

1.1 Сравнение приближённых методов решения нелинейных уравнений

1. Нарисовать график функции $f(x)$ и найти отрезки, где функция имеет простые корни и отличные от нуля две первые производные.

2. Найти с точностью $0,001$ все корни уравнения $f(x) = 0$ методом деления отрезка пополам и методом Ньютона; определить число приближений в каждом случае.

3. Сравнить полученные результаты.

Индивидуальный вариант: $f(x) = 2x^3 + 9x^2 - 21$.

1.2 Решение систем линейных уравнений методом Ньютона

1. Решить систему нелинейных уравнений графически и принять полученное решение за начальное приближение.

2. Решить систему методом Ньютона с точностью $\varepsilon = 0.01$.

Индивидуальный вариант: $\begin{cases} \cos x + y = 1.5 \\ 2x - \sin(y - 0.5) = 1 \end{cases}$.

2 Основные теоретические сведения

2.1 Сравнение приближённых методов решения нелинейных уравнений

Пусть на отрезке $[a, b]$ нелинейное уравнение $f(x) = 0$ имеет единственный простой корень, т.е. $f(a)f(b) < 0$.

В *методе деления отрезка пополам* поиск корня функции $f(x)$ начинается с деления отрезка $[a, b]$ пополам точкой $x = \frac{a+b}{2}$. Из двух получившихся отрезков выбирается тот, где находится корень уравнения. Обозначим этот отрезок $[a_1, b_1]$. Если $f(a)f(x) < 0$, то это отрезок $[a, x]$, иначе — $[x, b]$. Отправляясь от отрезка $[a_1, b_1]$ вдвое меньшей длины, опять находим середину отрезка $x = \frac{a_1+b_1}{2}$ и определяем по описанному алгоритму отрезок $[a_2, b_2]$ и т.д. На k-м шаге длина получившегося отрезка $[a_k, b_k]$ будет равна $b_k - a_k = \frac{b-a}{2}$. Процесс продолжается, пока $b_k - a_k > 2\varepsilon$, где ε — требуемая точность нахождения корня. Тогда приближённое значение корня уравнения $x = \frac{a_k+b_k}{2}$

Метод Ньютона решения нелинейного уравнения является итерационным. Для получения $(k + 1)$-й итерации метода x_{k+1} из точки $(x_k, f(x_k))$, лежащей на графике функции, проводится касательная. Точка пересечения касательной с осью абсцисс и есть следующее, $(k + 1)$-е приближение к корню уравнения. Алгебраически метод Ньютона сводится к рекуррентной зависимости

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots.$$

Достаточным условием сходимости метода является отличие от нуля первых двух производных функции $f(x)$ на отрезке $[a, b]$. В качестве начального приближения x_0 выбирается тот конец отрезка, где знак функции совпадает со знаком второй производной. Заданная погрешность ε считается достигнутой, если

$$f(x_k)f(x_k + \operatorname{sgn}(x_k - x_{k-1})\varepsilon) < 0.$$

2.2 Решение систем нелинейных уравнений методом Ньютона

Пусть $\vec{f}(\vec{x}) = \vec{0}$, где $\vec{x} = (x_1, \ldots, x_n)$ — вектор неизвестных; $\vec{f} = (f_1, \ldots, f_n)$ — вектор-функция. Выбрав начальное приближение $\vec{x^0} =$

$(x_1^0, \ldots, x_n^0)$ к решению системы, следующие приближения в методе Ньютона строим по рекуррентной зависимости

$$\vec{x^{k+1}} = \vec{x^k} - [\vec{f'}(\vec{x^k})]^{-1}\vec{f}(\vec{x^k}), \quad k = 0, 1, \ldots.$$

Здесь $\vec{f'}(\vec{x^k})$ — матрица Якоби системы, являющаяся производной вектор-функции $\vec{f}(\vec{x})$ в точке $\vec{x^k}$. Мы предполагаем, что матрица Якоби обратима в достаточно большой окрестности точного решения системы.

Для решения системы нелинейных уравнений с заданной точностью ε необходимо сравнить ε с погрешностью k-го приближения

$$||\vec{x^k} - \vec{x^{k-1}}|| = \max_{i \le i \le n} |x_i^k - x_i^{k-1}|.$$

Метод Ньютона сходится, если все функции $f_i(x_1, \ldots, x_n)$ дважды непрерывно дифференцируемы по всем переменным, и начальное приближение $\vec{x^0}$ находится достаточно близко к точному решению системы. Рецепта выбора начального приближения при $n > 1$ нет, поэтому желательно оценить, хотя бы грубо, значение точного решения, например, решив систему графически.

3 Реализация

В листинге 3.1 представлен исходный код программы на языке C++.

Листинг 3.1 – Исходный код программы на языке C++

```
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
#include <memory>

namespace {

using Segment = std::pair<double, double>;
using Function = std::function<double(const double x)>;

int Sgn(const double x) { return x < 0 ? -1 : x > 0; }

struct Result final {
  std::size_t k;
  double x;
};

class Method {
 public:
  virtual ~Method() = default;

  virtual Result FindRoot(const Segment segment, const double
      epsilon) = 0;
  virtual std::string_view Name() const = 0;
};

class SplitMethod final : public Method {
 public:
  SplitMethod(Function f) : f_(std::move(f)) {}

  Result FindRoot(const Segment segment, const double epsilon)
      override {
    auto [a, b] = segment;
    auto k = std::size_t{};
    auto x = (a + b) / 2;

```

```
36      while (b - a > 2 * epsilon) {
37        if (f_(a) * f_(x) < 0) {
38          b = x;
39        } else {
40          a = x;
41        }
42
43        x = (a + b) / 2;
44        ++k;
45      }
46
47      return {k, x};
48    }
49
50    std::string_view Name() const override { return kName; }
51
52   private:
53    static constexpr std::string_view kName = "Dichotomy";
54
55    Function f_;
56  };
57
58  class NewtonMethod final : public Method {
59   public:
60    NewtonMethod(Function f, Function df_dx, Function d2f_dx2)
61        : f_(std::move(f)),
62          df_dx_(std::move(df_dx)),
63          d2f_dx2_(std::move(d2f_dx2)) {}
64
65    Result FindRoot(const Segment segment, const double epsilon)
       override {
66      auto [a, b] = segment;
67      auto k = std::size_t{};
68      auto x_curr = (f_(a) * d2f_dx2_(a) > 0) ? a : b;
69      auto x_prev = 0.0;
70
71      do {
72        x_prev = x_curr;
73        x_curr = x_prev - f_(x_prev) / df_dx_(x_prev);
74        ++k;
```

```
75      } while (f_(x_curr) * f_(x_curr + Sgn(x_curr - x_prev) *
            epsilon) >= 0);
76
77      return {k, x_curr};
78    }
79
80    std::string_view Name() const override { return kName; }
81
82   private:
83    static constexpr std::string_view kName = "Newton";
84
85    Function f_, df_dx_, d2f_dx2_;
86  };
87
88  }  // namespace
89
90  int main() {
91    constexpr std::size_t kColW = 15;
92
93    {
94      std::cout << "Dichotomy and Newton nonlinear equation
            methods comparison.\n"
95                << "* The columns store {iterations, root} on the
                  segment.\n";
96
97      constexpr auto kEpsilon = 1e-3;
98
99      constexpr auto kF = [](const double x) {
100       return 2 * std::pow(x, 3) + 9 * std::pow(x, 2) - 21;
101     };
102     constexpr auto kDfDx = [](const double x) {
103       return 6 * std::pow(x, 2) + 18 * x;
104     };
105     constexpr auto kD2fDx2 = [](const double x) { return 12 * x
            + 18; };
106
107     const auto segments =
108         std::vector<Segment>{{-4.0, -3.5}, {-2.5, -2.0}, {1.0,
              1.5}};
109
110     std::cout << std::setw(kColW) << "Method";
```

```
111     for (auto&& [a, b] : segments) {
112       std::ostringstream oss;
113       oss << "[" << a << ", " << b << "]";
114       std::cout << std::setw(kColW) << oss.str();
115     }
116     std::cout << "\n";
117 
118     auto methods = std::vector<std::unique_ptr<Method>>{};
119     methods.push_back(std::make_unique<SplitMethod>(kF));
120     methods.push_back(std::make_unique<NewtonMethod>(kF, kDfDx,
        kD2fDx2));
121 
122     for (auto&& method : methods) {
123       std::cout << std::setw(kColW) << method->Name();
124       for (auto&& segment : segments) {
125         const auto [k, x] = method->FindRoot(segment, kEpsilon);
126         std::ostringstream oss;
127         oss << "{" << k << ", " << x << "}";
128         std::cout << std::setw(kColW) << oss.str();
129       }
130       std::cout << "\n";
131     }
132     std::cout << "\n";
133   }
134 
135   {
136     std::cout << "Newton nonlinear equation system method.\n";
137     constexpr auto kF = [](const double x, const double y) {
138       return std::cos(x) + y - 1.5;
139     };
140     constexpr auto kG = [](const double x, const double y) {
141       return 2 * x - std::sin(y - 0.5) - 1;
142     };
143 
144     constexpr auto kDfDx = [](const double x, const double y) {
145       return -std::sin(x);
146     };
147     constexpr auto kDfDy = [](const double x, const double y) {
        return 1; };
148     constexpr auto kDgDx = [](const double x, const double y) {
        return 2; };
```

```
149     constexpr auto kDgDy = [](const double x, const double y) {
150       return -std::cos(y - 0.5);
151     };
152
153     constexpr auto JacDetRev = [=](const double x, const double
          y) {
154       return 1 / (kDfDx(x, y) * kDgDy(x, y) - kDfDy(x, y) *
            kDgDx(x, y));
155     };
156
157     constexpr auto kEpsilon = 1e-2;
158     constexpr auto kX0 = 0.582;
159     constexpr auto kY0 = 0.665;
160
161     auto x_c = 0.5, y_c = 0.6;
162     double x_p, y_p, jdr, f, g;
163     std::size_t k = 0;
164
165     do {
166       x_p = x_c;
167       y_p = y_c;
168
169       jdr = JacDetRev(x_p, y_p);
170       f = kF(x_p, y_p);
171       g = kG(x_p, y_p);
172
173       x_c = x_p - jdr * (kDgDy(x_p, y_p) * f - kDfDy(x_p, y_p) *
            g);
174       y_c = y_p - jdr * (-kDgDx(x_p, y_p) * f + kDfDx(x_p, y_p)
            * g);
175
176       ++k;
177     } while (std::max(std::abs(x_c - x_p), std::abs(y_c - y_p))
          >= kEpsilon);
178
179     std::cout << "Analytical solution: (" << kX0 << ", " << kY0
          << ").\n"
180               << "Newton method's solution: (" << x_c << ", " <<
                    y_c << ").\n"
181               << "Iterations: " << k << ".\n"
182               << "Error: (" << std::abs(kX0 - x_c) << ", "
```

```
183                 << std::abs(kY0 - y_c) << ").\n";
184     }
185 }
```

4 Результаты

В листинге 4.1 представлены результаты работы программы.

Листинг 4.1 – Результаты работы программы

```
Dichotomy and Newton nonlinear equation methods comparison.
* The columns store {iterations, root} on the segment.
         Method     [-4, -3.5]     [-2.5, -2]       [1, 1.5]
      Dichotomy  {8, -3.75488}  {8, -2.08496}   {8, 1.34082}
         Newton  {2, -3.75652}  {2, -2.08525}   {2, 1.34084}

Newton nonlinear equation system method.
Analytical solution: (0.582, 0.665).
Newton method's solution: (0.581926, 0.664593).
Iterations: 2.
Error: (7.39242e-05, 0.000406812).
```