МИНИСТЕРСТВО НАУКИ и высшего образования РОССИЙСКОЙ ФЕДЕРАЦИИ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**

**ОТЧЁТ**

**по лабораторной работе №1**

**по дисциплине: «** *Вычислительная математика* **»**

**Вариант 2**

Выполнил(а):Проверил:

Студенты гр. *АП-227* *Ландовский В.В.*

*Бузмаков А.И.*

*Шестаков К.Д.*

*Федотов И.В.*

«\_\_\_» \_\_\_\_\_\_ 20\_\_г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск

2024

**Цель работы**

1. Изучение различных методов решения нелинейных уравнений, таких как метод хорд, метод Ньютона, метод секущих, метод простых итераций, метод половинного деления.
2. Разработка программного кода для реализации данных методов.
3. Проведение численных экспериментов с использованием разработанных методов для решения различных типов нелинейных уравнений.
4. Анализ и сравнение полученных результатов для определения эффективности и применимости каждого метода.

**Постановка задачи**

1. Для каждого корня заданного уравнения f (x) = 0 задать отрезок, содержащий корень, с помощью построения графика f (x) в любом доступном математическом программном пакете. Границы отрезка выбрать как ближайшие к корню целые числа. При необходимости скорректировать значения границ так, чтобы выполнялись условия сходимости метода Ньютона, поведение производных проанализировать по графикам. В том же или подобном программном средстве вычислить значения корней максимально точно.

Вариант:

2. Выполнить уточнение корней тремя методами: Ньютона, простых итераций и одного из методов заданного вариантом: половинного деления, хорд, секущих. Для этого реализовать итерационный процесс соответствующих методов с помощью доступных систем программирования, для работы с дробными числами использовать 32 битное представление. При разработке алгоритмов стараться по возможности минимизировать вычислительные затраты (не вычислять f (x) повторно, если можно использовать ранее вычисленное значение). Для метода Ньютона и его модификаций вычисления вести до выполнения условия (1.4), задать точность 10-3. В методах, использующих качестве начального приближения одно значение, попытаться (вопреки теоретическим представлениям) использовать как x0 различные концы отрезков. Вывести промежуточные результаты (приближенные значения на каждой итерации), объяснить полученные результаты.

3. Для каждого метода и каждого корня экспериментально подобрать минимальное значение заданной погрешности, при которой вычисления сходятся к конечному значению (не возникает переполнения) и завершаются корректно (программа не входит в бесконечный цикл). Сравнить значения, вычисленные с этой точностью, с полученными в пунктах 1 и 2. Проделать тоже самое для положительного корня уравнения f (x) = 100 x2-10000 x-v, где v – номер варианта. Объяснить полученные результаты.

**Графики функций**

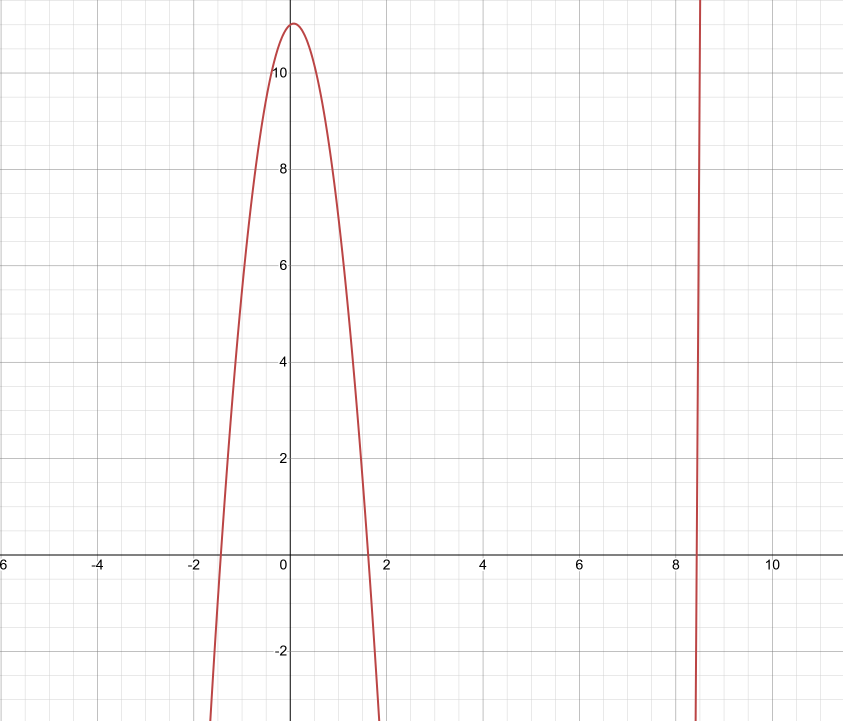


Рисунок 1 – График функции (крупный масштаб)

По графику функции можно увидеть, что искомое уравнение имеет три корня. Эти корни находятся на отрезках соответственно.

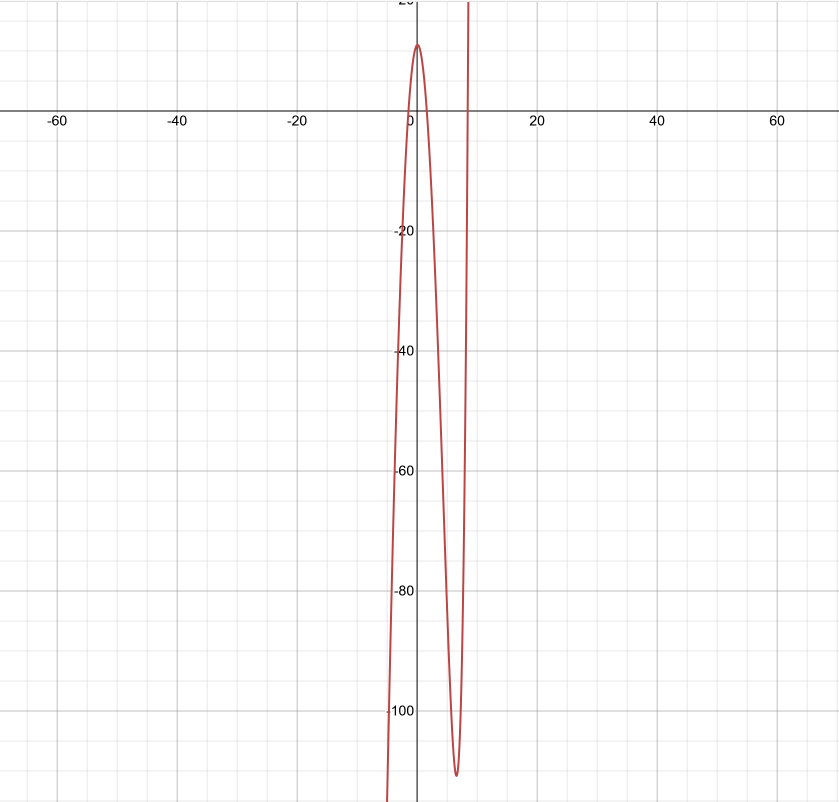
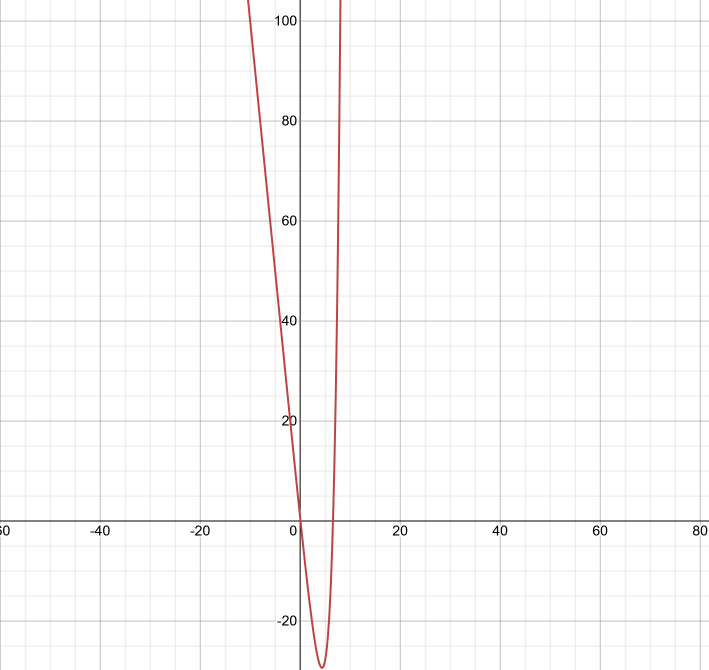


Рисунок 2 – График функции (малый масштаб)

  
Рисунок 3 – Первая производная

Анализируя первую производную, можно увидеть, что она меняет знак в двух точках. Эти точки находятся на отрезках соответственно.

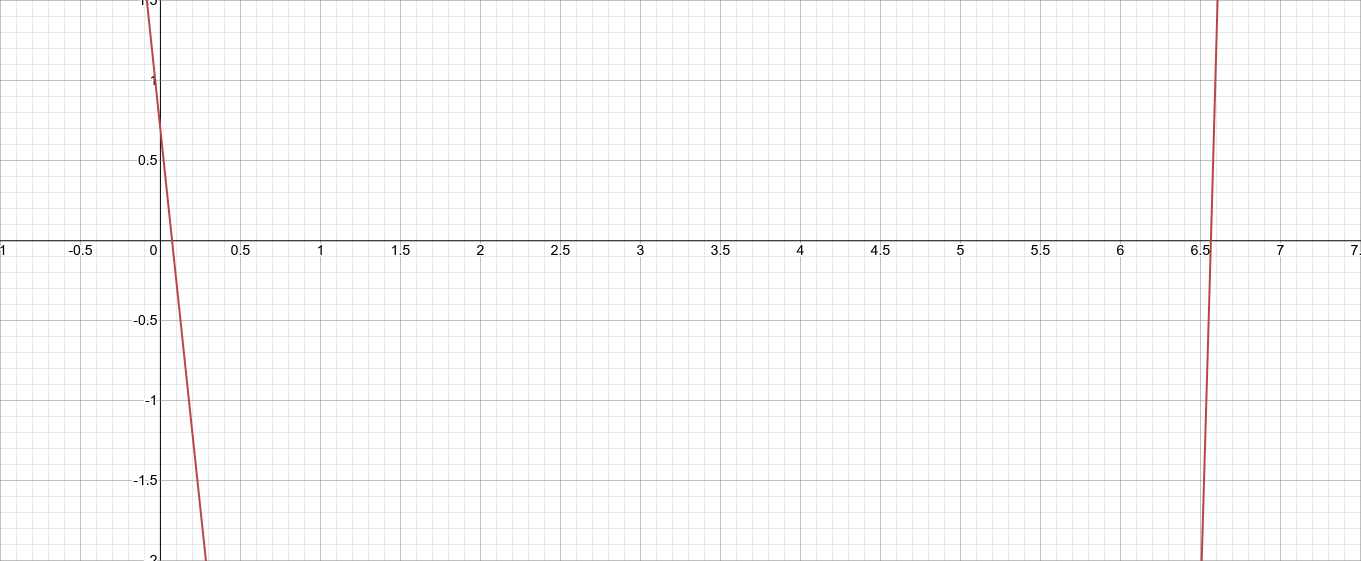


Рисунок 5 – Нули первой производной

Проанализировав соответствующим образом вторую производную, получаем отрезок, в одной из точек которого производная меняет знак:

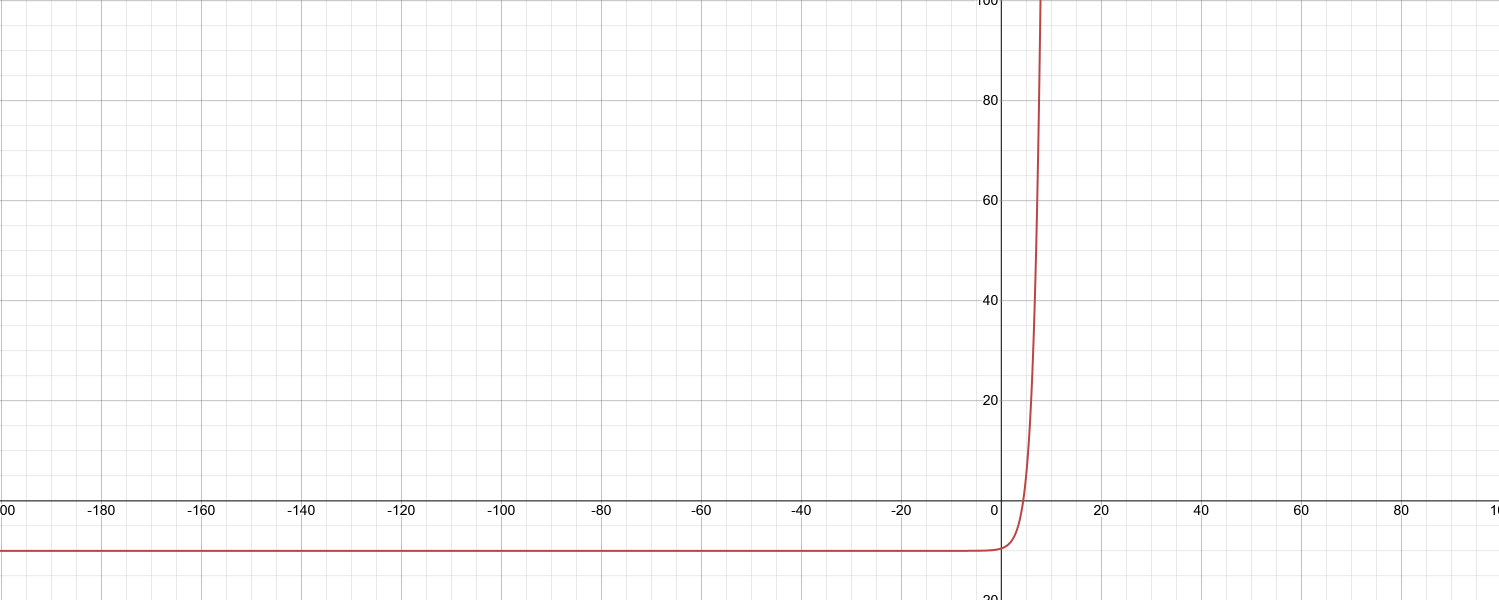


Рисунок 4 – Вторая производная

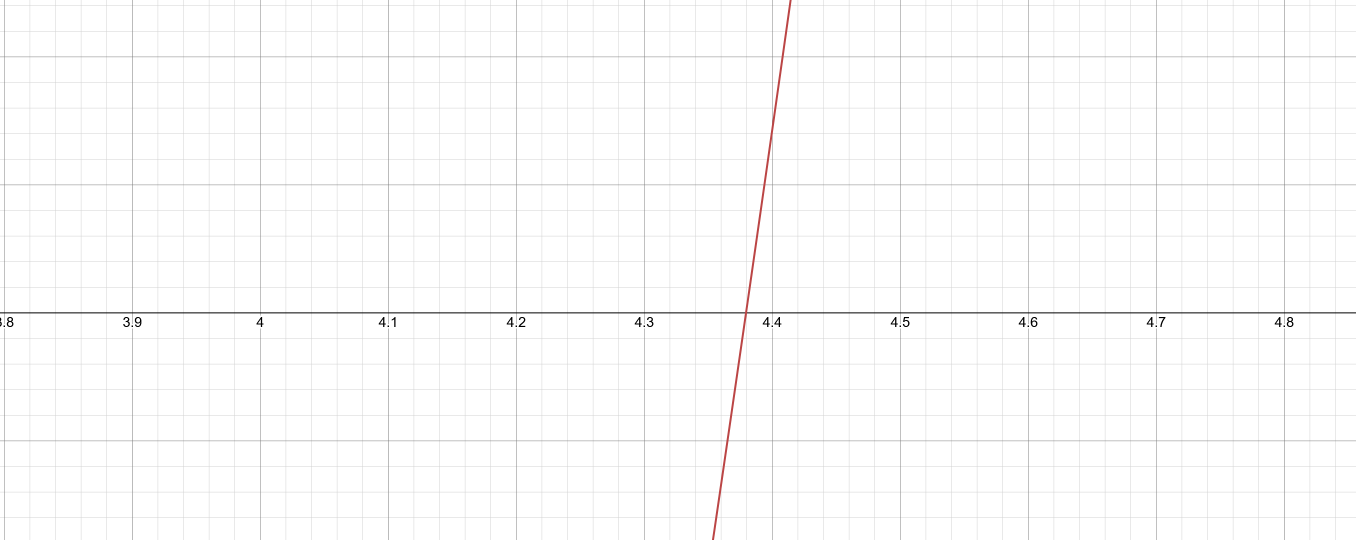


Рисунок 6 – Ноль второй производной

Рассчитаем корни уравнения с максимальной точностью. Для этого воспользуемся математическим пакетом Mathcad. Полученные в результате вычислений корни приведены ниже.

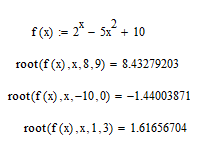


Рисунок 7 - Корни

Сравним рассчитанные корни с точками на графике.

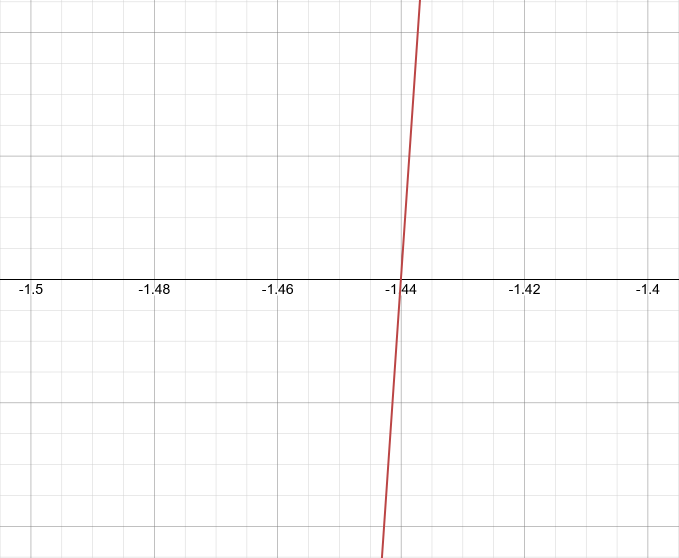


Рисунок 8 – Первый корень на графике



Рисунок 9 – Второй корень на графике

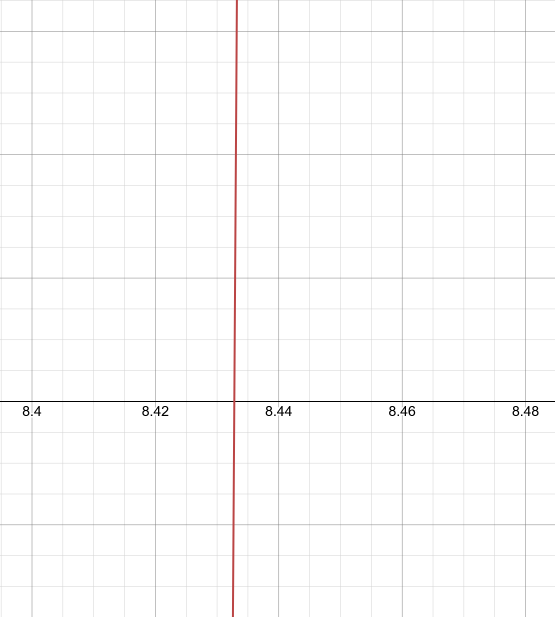


Рисунок 10 – Третий корень на графике

По графику видно, что вычисления выполнены максимально точно. В нашем случае точность вычислений составила

Для уравнения проделаем те же действия, что и при анализе первой функции.

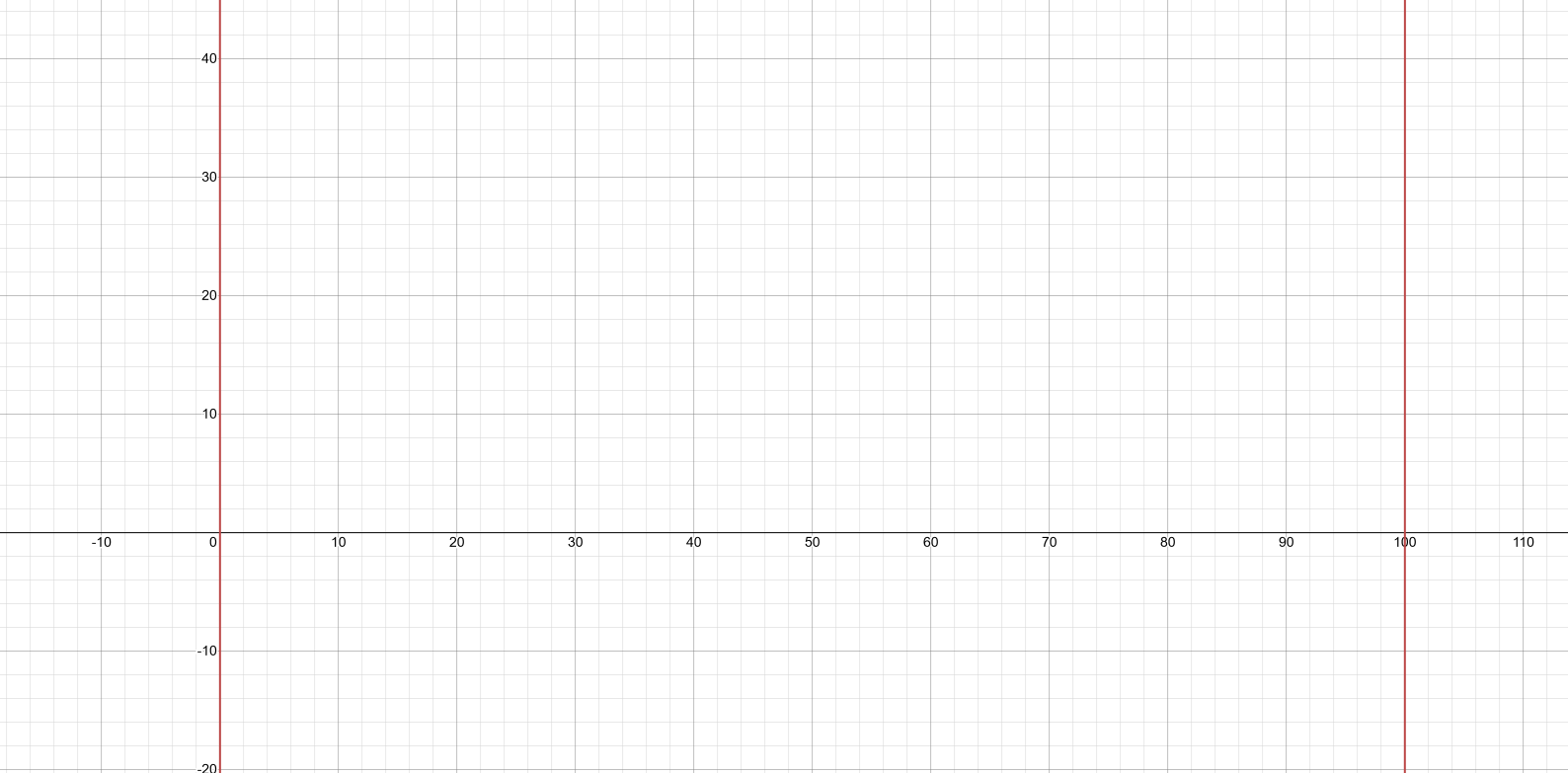


Рисунок 11 – График функции

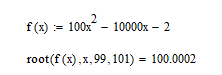
**

Рисунок 12 – Положительный корень уравнения

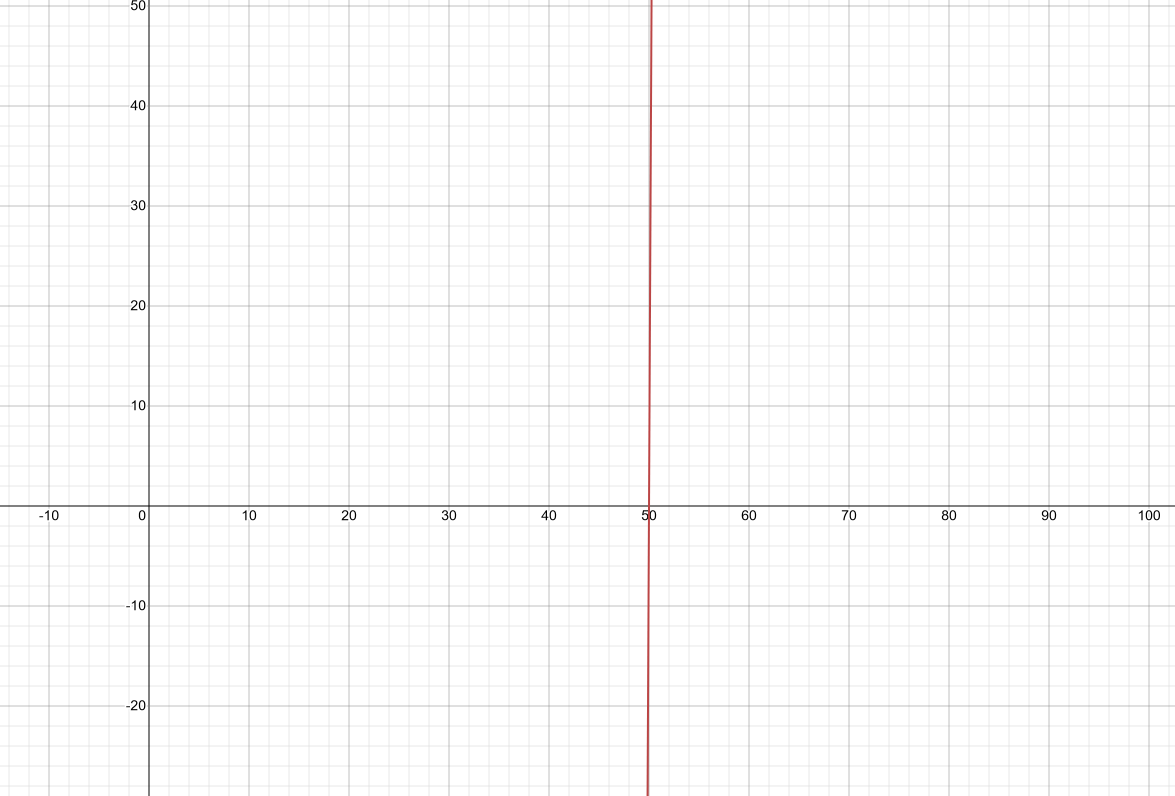


Рисунок 13 – График производной функции

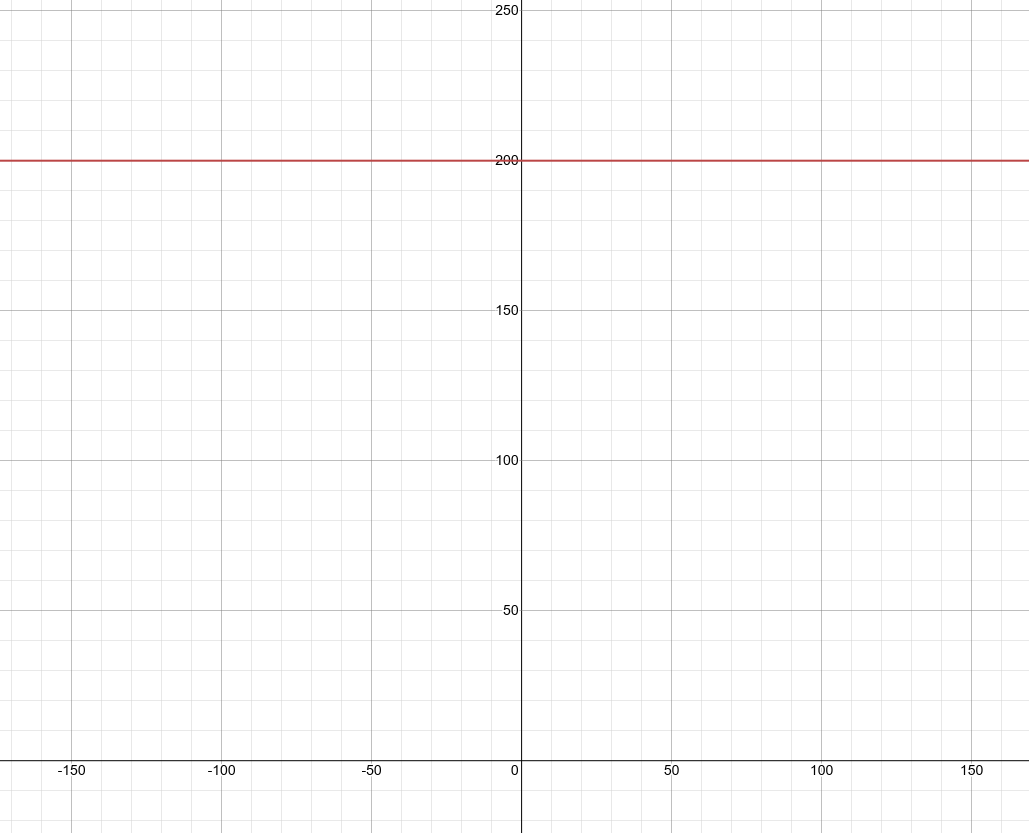


Рисунок 14 – График второй производной

После анализа получили: корни функции находятся на отрезках Первая производная меняет знак на отрезке Вторая производная знак не меняет.

**Результаты вычислений, полученные с помощью разработанных программ**

Таблица 1 – Результаты вычислений для первого уравнения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** |  | **Кол-во итераций** |  |  |  |
| **Метод Ньютона** | **1e-3** | **3** | **-1,44041** | **1,616567** | **8,432819** |
| **max = 1e-6** | **5** | **-1,440039** | **1,616567** | **8,432792** |
| **Метод простых итераций** | **1e-3** | **4** | **-1,440040** | **1, 616567** | **8,432817** |
| **max = 1e-45** | **5** | **-1,440039** | **1,616567** | **8,432792** |
| **Метод хорд** | **1e-3** | **2** | **-1,440037** | **1,616566** | **8,432765** |
| **max = любое** | **3** | **-1,440039** | **1,616567** | **8,432792** |

Таблица 2 – Результаты вычислений для второго уравнения

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** |  | **Кол-во итераций** |  |
| **Метод Ньютона** | **1e-3** | **2** | **100,0002** |
| **max = 1e-45** | **4** | **100,0002** |
| **Метод простых итераций** | **1e-3** | **3** | **100,0002** |
| **max = 1e-45** | **3** | **100,0002** |
| **Метод хорд** | **1e-3** | **1** | **100,0001** |
| **max = 1e-5** | **2** | **100,0002** |

**Объяснение полученных результатов**

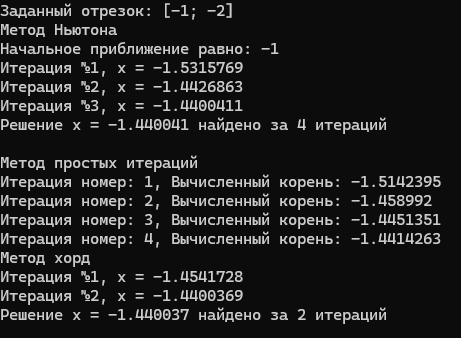
Проанализировав полученные результаты, можно сделать следующие выводы:

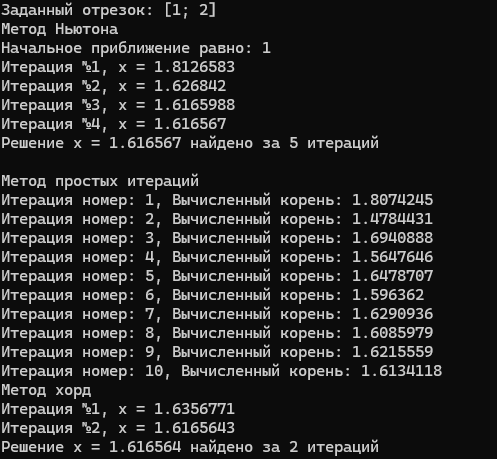
1) При увеличении точности вычислений увеличивается количество итераций, необходимых для получения ответа. При этом, если точность является меньшей, чем некоторое минимальное значение, программа уходит в бесконечный цикл.

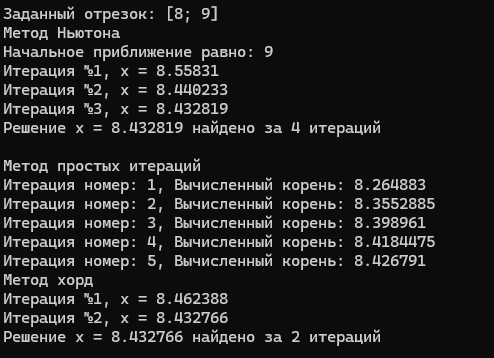
2) Полученные в процессе использования разработанной программы расчеты с точностью до шести цифр после запятой совпадают с расчетами, сделанными в специализированном математическом пакете, что говорит об эффективности использования разработанной программы для нахождения решений уравнения с заданной точностью.

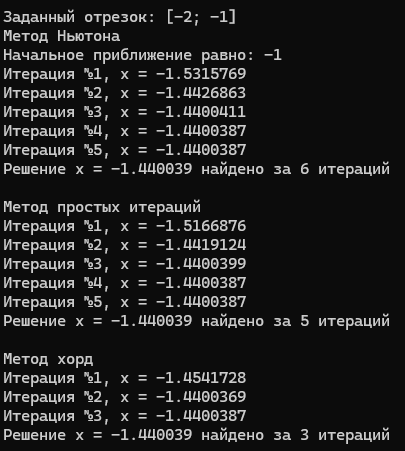
3) В зависимости от выбора приближенного значения (или отрезка) меняется время, необходимое для расчета корня(-ей) уравнения. Это связано с тем, что при достаточно близком нахождении приближенного значения и действительного корня программе требуется меньше итераций для вычисления результата. При этом, если отрезок достаточно велик, разные методы вычисления дают разные результаты. Это объясняется работой алгоритмов разных методов. Например, метод простых итераций на каждой итерации получает значение, меньшее предыдущего, соответственно если за начальное приближение принять величину, меньшую чем искомый корень, шанс найти этот корень будет меньше шанса нахождения корня, лежащего левее.

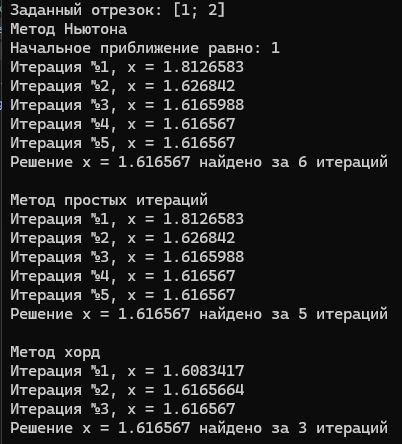
**Приложение**

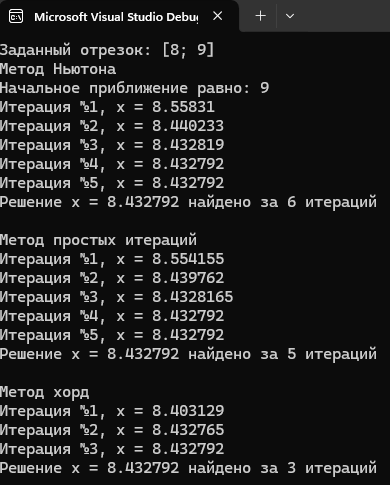
****

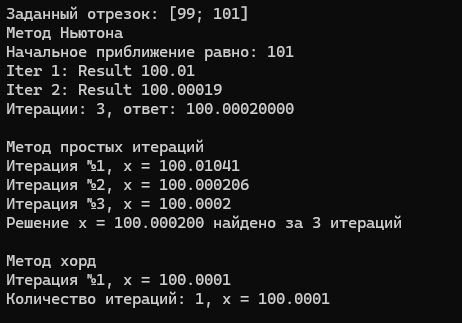
****

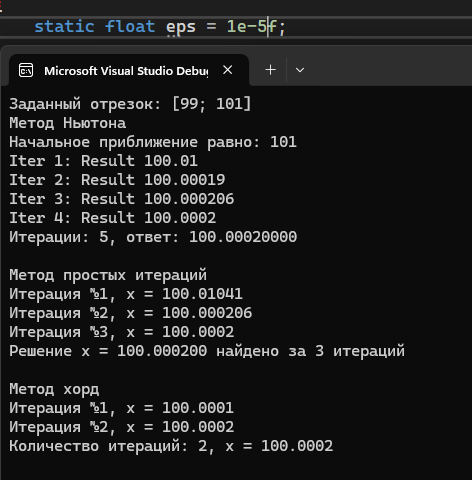
****

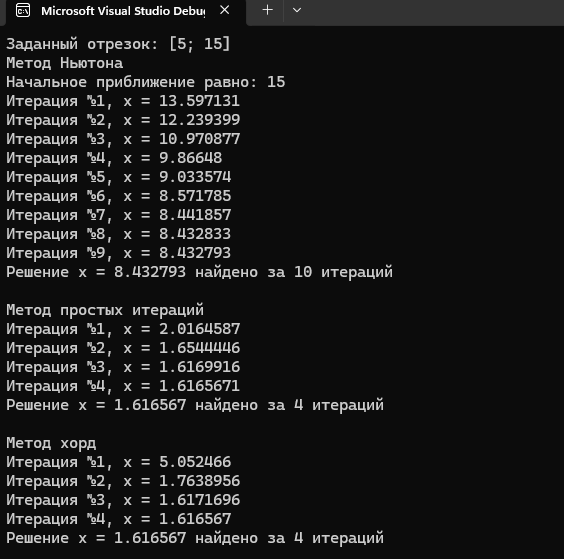
****

****

****

****

****

****

**Код разработнной программы**

|  |
| --- |
| Первое уравнение |
| class Program  {  static float eps = 1e-6f;  static float fx(float x)  {  return (float)Math.Pow(2, x) - 5 \* (float)Math.Pow(x, 2) + 10;  }  static float dfx(float x)  {  return (float)Math.Log(2) \* (float)Math.Pow(2, x) - 10 \* x;  }  static float ddfx(float x)  {  return (float)Math.Pow(Math.Log(2), 2) \* (float)Math.Pow(2, x) + 10;  }  static float gx(float x)  {  float result = (float)(Math.Pow(2, x) + 10) / 5;  return (float)Math.Sqrt(result);  }  static float dgx(float x)  {  float result = (float)(Math.Sqrt(5) \* Math.Log(2) \* Math.Pow(2, x - 1));  result = result / (float)(5 \* Math.Sqrt(Math.Pow(2, x) + 10));  return result;  }  static void NewtonMethod(float a, float b)  {  float x = 0;  // Проверяем знаки первой производной на интервале  if ((Math.Sign(dfx(a)) != Math.Sign(dfx(b))) && (Math.Sign(dfx(a)) != Math.Sign(dfx(b))))  {  return;  }  // Проверяем знаки второй производной на интервале  if ((Math.Sign(ddfx(a)) != Math.Sign(ddfx(b))) && (Math.Sign(ddfx(a)) != Math.Sign(ddfx(b))))  {  return;  }  if (fx(a) \* ddfx(a) > 0) // если знаки функций совпадают  {  x = a;  }  else if (fx(b) \* ddfx(b) > 0) // если знаки функций совпадают  {  x = b;  }  else  {  Console.WriteLine("Ошибка: не соблюдается условие совпадаемости знаков f(x) и f''(x)\n");  return;  }  Console.WriteLine($"Начальное приближение равно: {x}");  int iterations = 1;  while (iterations < 1000)  {  float y1 = fx(x); float y2 = dfx(x);  float dx = y1 / y2;  x -= dx;  Console.WriteLine($"Итерация №{iterations}, x = {x}");  iterations++;  if (Math.Abs(dx) < eps)  {  Console.WriteLine($"Решение x = {x:0.000000} найдено за {iterations} итераций\n");  break;  }  }  }  static void HordeMethod(float a, float b)  {  int iterations = 0;  while (Math.Abs(b - a) > eps)  {  iterations++;  a = b - (b - a) \* fx(b) / (fx(b) - fx(a));  b = a - (a - b) \* fx(a) / (fx(a) - fx(b));  Console.WriteLine($"Итерация №{iterations}, x = {b}");  }  Console.WriteLine($"Решение x = {b:0.000000} найдено за {iterations} итераций\n");  }  public static void SimpleIterMethod(float x0, float M)  {  int count = 1;  float x1 = x0;  while (true)  {  float l = -1 / M;  x1 = x0 + l \* fx(x0);  Console.WriteLine("Итерация номер: " + count++ + ", Вычисленный корень: " + x1);  if (Math.Abs(x1 - x0) < eps)  break;  x0 = x1;  }  }  static void Main(string[] args)  {  float a = 1;  float b = 2;  Console.WriteLine($"Заданный отрезок: [{a}; {b}]");  Console.WriteLine("Метод Ньютона");  NewtonMethod(a, b);  Console.WriteLine("Метод простых итераций");  float M = Math.Max(dfx(a), dfx(b));  SimpleIterMethod((float)1.1, M);  Console.WriteLine("Метод хорд");  HordeMethod(a, b);  }  } |

|  |
| --- |
| Второе уравнение |
| class Program  {  static float eps = 1e-45f;    static float fx(float x)  {  return 100 \* (float)Math.Pow(x, 2) - 10000 \* x - 2;  }  static float dfx(float x)  {  return 200 \* x - 10000;  }  static float ddfx(float x)  {  return 200;  }  static float gx(float x)  {  float result = 100 \* x - (float)0.02;  return (float)(Math.Sqrt(result));  }  static float dgx(float x)  {  float result = 125 \* 2 \* (float)Math.Sqrt(2);  result = result / (float)Math.Sqrt(5000 \* x - 1);  return result;  }  static void NewtonMethod(float a, float b)  {  float x = 0;  // Проверяем знаки первой производной на интервале  if ( (Math.Sign(dfx(a)) != Math.Sign(dfx(b))) && (Math.Sign(dfx(a)) != Math.Sign(dfx(b))) )  {  return;  }  // Проверяем знаки второй производной на интервале  if ((Math.Sign(ddfx(a)) != Math.Sign(ddfx(b))) && (Math.Sign(ddfx(a)) != Math.Sign(ddfx(b))))  {  return;  }  if (fx(a) \* ddfx(a) > 0) // если знаки функций совпадают  {  x = a;  }  else if (fx(b) \* ddfx(b) > 0) // если знаки функций совпадают  {  x = b;  }  else  {  Console.WriteLine("Ошибка: не соблюдается условие совпадаемости знаков f(x) и f''(x)\n");  return;  }  Console.WriteLine($"Начальное приближение равно: {x}");  int iterations = 1;  while (true)  {  float y1 = fx(x); float y2 = dfx(x);  float dx = y1 / y2;  x -= dx;  Console.WriteLine($"Iter {iterations}: Result {x}");  iterations++;  if (Math.Abs(dx) < eps)  {  Console.WriteLine($"Итерации: {iterations}, ответ: {x:0.00000000}\n");  break;  }  }  }  public static void SimpleIterMethod(float x0, float M)  {  int count = 1;  float x1 = x0;  while (true)  {  float l = -1 / M;  x1 = x0 + l \* fx(x0);  Console.WriteLine("Итерация номер: " + count++ + ", Вычисленный корень: " + x1);  if (Math.Abs(x1 - x0) < eps)  break;  x0 = x1;  }  }  static void HordeMethod(float a, float b)  {  int iterations = 0;  while (Math.Abs(b - a) > eps)  {  iterations++;  a = b - (b - a) \* fx(b) / (fx(b) - fx(a));  b = a - (a - b) \* fx(a) / (fx(a) - fx(b));  Console.WriteLine($"Итерация №{iterations}, x = {b}");  }  Console.WriteLine($"Количество итераций: {iterations}, x = {b}\n");  }  static void Main(string[] args)  {  float a = 99;  float b = 101;  Console.WriteLine($"Заданный отрезок: [{a}; {b}]");  Console.WriteLine("Метод Ньютона");  NewtonMethod(a, b);  Console.WriteLine("Метод простых итераций");  float M = Math.Max(dfx(a), dfx(b));  SimpleIterMethod((float)100, M);  Console.WriteLine("Метод хорд");  HordeMethod(a, b);  }  } |