МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по курсу: Численные методы

Выполнил: Группа: БПМ-18-2

Студент: Соседка Артём Валерьевич

Проверил: преподаватель: Рубчинский Александр Анатольевич

Москва, 2020 г.

Содержание

[Постановка задачи 3](#_Toc52323439)

[Метод решения 3](#_Toc52323440)

[Исследование функции 3](#_Toc52323441)

[Выполнение работы 4](#_Toc52323442)

[1. Бисекция 4](#_Toc52323443)

[Описание метода 4](#_Toc52323444)

[Алгоритм 4](#_Toc52323445)

[Результат работы алгоритма 4](#_Toc52323446)

[Вывод 5](#_Toc52323447)

[2. Метод ложной позиции 5](#_Toc52323448)

[Описание метода 5](#_Toc52323449)

[Алгоритм 5](#_Toc52323450)

[Результат работы алгоритма 5](#_Toc52323451)

[Вывод 5](#_Toc52323452)

[3. Метод секущих 5](#_Toc52323453)

[Описание метода 5](#_Toc52323454)

[Алгоритм 6](#_Toc52323455)

[Результат работы алгоритма 6](#_Toc52323456)

[Вывод 6](#_Toc52323457)

[4. Метод Ньютона 6](#_Toc52323458)

[Описание метода 6](#_Toc52323459)

[Алгоритм 6](#_Toc52323460)

[Результат работы алгоритма 6](#_Toc52323461)

[Вывод 6](#_Toc52323462)

[Вывод 7](#_Toc52323463)

[Приложения 7](#_Toc52323464)

[Приложение 1 7](#_Toc52323465)

[Приложение 2 7](#_Toc52323466)

[Приложение 2.1 7](#_Toc52323467)

[Приложение 2.2 8](#_Toc52323468)

[Приложение 2.3 8](#_Toc52323469)

[Приложение 2.4 9](#_Toc52323470)

# Постановка задачи

Вариант 20

A10 = B1 =

Необходимо рассмотреть 4 метода решения уравнений:

1. Бисекция
2. Метод секущих
3. Метод ложной позиции
4. Метод Ньютона

С помощью данных методов решения нелинейных уравнений требуется найти решение уравнения

,

при этом рассмотрев каждый метод с тремя разными степенями точности (0.01, 0.001, 0.0001).

## Метод решения

Мною был выбран язык Python для выполнения поставленной задачи, так как он позволяет просто и эффективно работать с математическими функциями и массивами данных, а также позволяет легко визуализировать данные (строить графики функций и выводить таблицы промежуточных значений). Написание и отладка кода производилась в интерактивной среде Jupyter Notebook, которая позволяет структурировать код в «ячейки», имеет поддержку Markdown для написания текста и обладает удобным выводом результатов работы каждой отдельной ячейки. При работе в Jupyter Notebook использовался бесплатный облачный сервис Google Colaboratory.

## Исследование функции

Рассмотрим график функции:

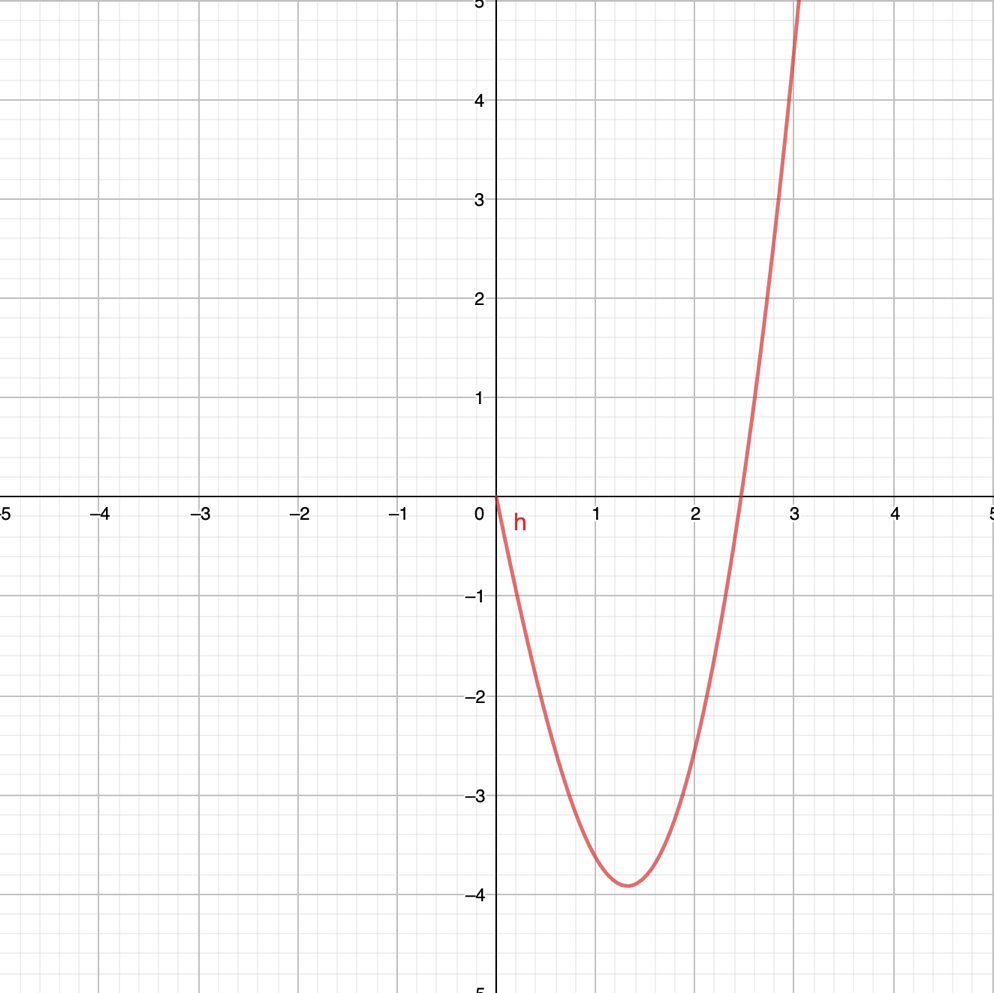


Рис. 1 - график функции f(x)

Поиск корней будет производиться на отрезке

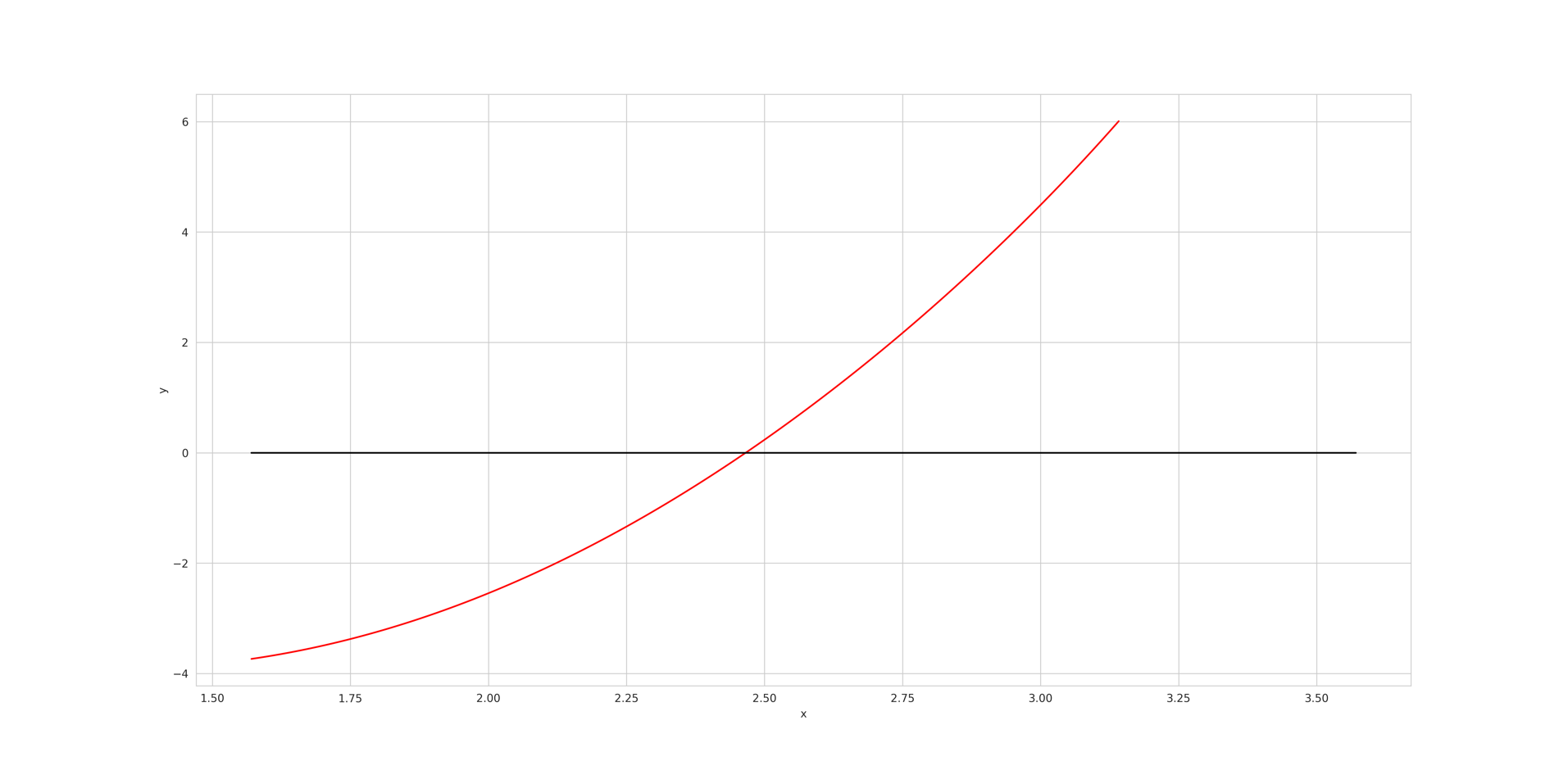


Рис. 2 - график функции f(x) на отрезке

# Выполнение работы

## 1. Бисекция

### Описание метода

Для решения уравнения методом бисекции мы выбираем некий отрезок от , который точно содержит корень нашего уравнения. Важно, что и были разных знаков. Далее постепенно уменьшая длину отрезка, во время каждой итерации алгоритма, мы считаем дельту между и и тогда, когда она будет меньше заданной точности, алгоритм останавливает свою работу. Таким образом, корень находится прямо между получившимися ().

### Алгоритм

Я выбрал отрезок , где . Алгоритм реализован через класс, который принимает функцию , отрезок и требуемые точности измерения в виде массива*.* В результате, алгоритм выводит таблицу, в которой отражены все промежуточные результаты, выдаёт полученный ответ и количество итераций.

### Результат работы алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **1** | 1,5708 | -3,73687 | 3,14159 | 6,01005 | 1,5708 |
| **2** | 2,35619 | -0,71461 | 3,14159 | 6,01005 | 0,7854 |
| **3** | 2,35619 | -0,71461 | 2,74889 | 2,16123 | 0,3927 |
| **4** | 2,35619 | -0,71461 | 2,55254 | 0,61182 | 0,19635 |
| **5** | 2,45437 | -0,07913 | 2,55254 | 0,61182 | 0,09817 |
| **6** | 2,45437 | -0,07913 | 2,50346 | 0,25942 | 0,04909 |
| **7** | 2,45437 | -0,07913 | 2,47891 | 0,08841 | 0,02454 |
| **8** | 2,45437 | -0,07913 | 2,46664 | 0,00421 | 0,01227 |
| **9** | 2,46051 | -0,03757 | 2,46664 | 0,00421 | 0,00614 |
| **10** | 2,46357 | -0,01671 | 2,46664 | 0,00421 | 0,00307 |
| **11** | 2,46511 | -0,00626 | 2,46664 | 0,00421 | 0,00153 |
| **12** | 2,46587 | -0,00103 | 2,46664 | 0,00421 | 0,00077 |
| **13** | 2,46587 | -0,00103 | 2,46626 | 0,00159 | 0,00038 |
| **14** | 2,46587 | -0,00103 | 2,46607 | 0,00028 | 0,00019 |
| **15** | 2,46597 | -0,00037 | 2,46607 | 0,00028 | 0,0001 |

Табл. 1 - таблица результатов работы алгоритма бисекции с точностью до 0.0001

### Вывод

Алгоритм дал ответ с точностью до 0.01 за 9 итераций, ответ с точностью до 0.001 за 12 итераций и ответ с точностью до 0.0001 за 15 итераций.

Ответ: 2.46597

## 2. Метод ложной позиции

### Описание метода

Для решения уравнения мы аналогично прошлому методу выбираем некий отрезок, где вероятнее всего находится корень нелинейной функции , между точками . При этом должно быть выполнено условие, что и имеют разные знаки. Между точками и строится линейная функция , которая будет иметь корень (по существу, это ). Следующая же линейная функция будет построена между точкой и точкой . Стоит упомянуть, что приближенная формула для вычисления точки

Таким образом нам следует проводить этот алгоритм до того момент пока не будет меньше заранее заданной точности.

### Алгоритм

Отрезок выбран такой же, как и при методе бисекции. Алгоритм работы метода идентичен методу бисекции за исключением непосредственного поиска корня.

### Результат работы алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **1** | 1,5708 | -3,73687 | 3,14159 | 6,01005 | 1,5708 |
| **2** | 2,17302 | -1,75048 | 3,14159 | 6,01005 | 0,60223 |
| **3** | 2,3915 | -0,49251 | 3,14159 | 6,01005 | 0,21847 |
| **4** | 2,44831 | -0,11997 | 3,14159 | 6,01005 | 0,05681 |
| **5** | 2,46188 | -0,02825 | 3,14159 | 6,01005 | 0,01357 |
| **6** | 2,46506 | -0,0066 | 3,14159 | 6,01005 | 0,00318 |
| **7** | 2,4658 | -0,00154 | 3,14159 | 6,01005 | 0,00074 |
| **8** | 2,46597 | -0,00036 | 3,14159 | 6,01005 | 0,00017 |
| **9** | 2,46601 | -0,00008 | 3,14159 | 6,01005 | 0,00004 |

Табл. 2 - таблица результатов работы алгоритма ложной позиции с точностью до 0.0001

### Вывод

Алгоритм дал ответ с точностью до 0.01 за 6 итераций, ответ с точностью до 0.001 за 7 итераций и ответ с точностью до 0.0001 за 9 итераций.

Ответ: 2.46601

## 3. Метод секущих

### Описание метода

Суть метода заключается в том же принципе, что в методе ложных позиций, но при этом единственное отличие состоит в том, что если метод ложных позиций действует путем сохранения двух последних если разных знаков, то в случае с методом секущих алгоритм не учитывает знак, тем самым всегда сохраняя два последних значения.

### Алгоритм

Отрезок выбран такой же, как и при методе бисекции. Алгоритм работы метода идентичен методу бисекции и методу ложной позиции за исключением непосредственного поиска корня.

### Результат работы алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **1** | 1,5708 | -3,73687 | 3,14159 | 6,01005 | 1,5708 |
| **2** | 3,14159 | 6,01005 | 2,17302 | -1,75048 | 0,96857 |
| **3** | 2,17302 | -1,75048 | 2,3915 | -0,49251 | 0,21847 |
| **4** | 2,3915 | -0,49251 | 2,47703 | 0,07544 | 0,08553 |
| **5** | 2,47703 | 0,07544 | 2,46567 | -0,00242 | 0,01136 |
| **6** | 2,46567 | -0,00242 | 2,46602 | -0,00001 | 0,00035 |
| **7** | 2,46602 | -0,00001 | 2,46602 | 0 | 0 |

Табл. 3 - таблица результатов работы алгоритма секущих с точностью до 0.0001

### Вывод

Алгоритм дал ответ с точностью до 0.01 за 6 итераций, ответ с точностью до 0.001 за 6 итераций и ответ с точностью до 0.0001 за 7 итераций. Таким образом, метод секущих быстрее, чем метод ложной позиции.

Ответ: 2.46602

## 4. Метод Ньютона

### Описание метода

Суть метода заключается в итерационном приближении первоначального значения к корню по следующей формуле: . Таким образом нам следует проводить этот алгоритм до того момент пока не будет меньше заранее заданной точности.

### Алгоритм

В качестве начальной точки возьмём середину выбранного отрезка – точку 2.35619. Для метода Ньютона нам также потребуется производная нашей функции . Решение методом Ньютона реализовано в виде класса, который принимает функцию, её производную, точку (в нашем случае, мы передаём отрезок и класс сам находит середину данного отрезка) и требуемые точности измерения в виде массива. На выходе мы получим таблицу с промежуточными вычислениями, найденный корень и количество итераций.

### Результат работы алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **0** | 2,35619 | -0,71461 | 6,18944 | 0 |
| **1** | 2,47165 | -0,71461 | 6,18944 | 0,11546 |
| **2** | 2,46604 | 0,03848 | 6,85512 | 0,00561 |
| **3** | 2,46602 | 0,00009 | 6,82285 | 0,00001 |

Табл. 4 - таблица результатов работы алгоритма Ньютона с точностью до 0.0001

### Вывод

Метод Ньютона является самым быстрым методом из рассмотренных, так, он дал ответ с точностью до 0.01 за 2 итерации, ответ с точностью до 0.001 за 3 итерации и ответ с точностью до 0.0001 снова за 3 итерации.

Ответ: 2.46602

# Вывод

Составим таблицу результатов различных методов:

|  |  |  |
| --- | --- | --- |
| Метод | *Количество итераций* | *Корень уравнения* |
| **Метод бисекции** | 15 | 2.46597 |
| **Метод ложной позиции** | 9 | 2.46601 |
| **Метод секущих** | 7 | 2.46602 |
| **Метод Ньютона** | 3 | 2.46602 |

Наглядно видно, что самым сложным для расчётов для нашей функции оказался метод бисекции, а самым быстрым – метод Ньютона. Стоит также отметить, что хоть метод бисекции и занял наибольшее количество итераций, он всё равно не смог продемонстрировать высокую точность. В свою очередь, метод секущих и метод Ньютона показали высокую скорость нахождения корня, а также высокую точность, из чего можно вывести, что для поиска корня уравнения с одним неизвестным лучше всего использовать методы секущих и Ньютона.

# Приложения

Приложение 1 – ссылка на исходные коды

1. <https://github.com/sostema/study_misis/blob/master/semester_5/numerical_methods/laboratory_work_1/laboratory_work_1.ipynb>
2. <https://github.com/sostema/study_misis/tree/master/semester_5/numerical_methods/laboratory_work_1>
3. <https://colab.research.google.com/drive/15038nY2trjCjYiV7KWjIKFOUmZFzInxT?usp=sharing>

Приложение 2 – реализация рассматриваемых методов (в частности, алгоритм поиска решения)

Приложение 2.1 – реализация метода бисекции (в частности, алгоритм поиска решения)

class BisectionMethod(Method):  
  
 def solve(self, epsilon: float) -> typing.Tuple[float, pd.DataFrame]:  
 df = self.dataframe  
  
 (x1, x2) = (self.segment.left\_border, self.segment.right\_border)  
 f1 = self.function(x1)  
 f2 = self.function(x2)  
 del\_x = abs(x2 - x1)  
 round\_base = self.get\_round\_base(epsilon)  
  
 middle = 0  
 while del\_x > epsilon:  
 df.loc[len(df)] = (list(map(lambda x:  
 round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
 middle = (x2 + x1) / 2  
 f\_middle = self.function(middle)  
 if f\_middle \* f1 < 0:  
 x2 = middle  
 f2 = f\_middle  
 else:  
 x1 = middle  
 f1 = f\_middle  
 del\_x = abs(x2 - x1)  
  
 df.loc[len(df)] = (list(map(lambda x:  
 round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
 result = middle  
 return (result, df)

Приложение 2.2 – реализация метода ложной позиции (в частности, алгоритм поиска решения)

class FalsePositionMethod(Method):  
  
 def solve(self, epsilon: float) -> typing.Tuple[float, pd.DataFrame]:  
 df = self.dataframe  
  
 (x1, x2) = (self.segment.left\_border, self.segment.right\_border)  
 f1 = self.function(x1)  
 f2 = self.function(x2)  
 del\_x = abs(x2 - x1)  
 round\_base = self.get\_round\_base(epsilon)  
  
 middle = 0  
 while del\_x > epsilon:  
 df.loc[len(df)] = (list(map(lambda x: round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
 middle = x2 \* f1 / (f1 - f2) + x1 \* f2 / (f2 - f1)  
 f\_middle = self.function(middle)  
 if f\_middle \* f1 > 0:  
 del\_x = middle - x1  
 x1 = middle  
 f1 = f\_middle  
 else:  
 del\_x = x2 - middle  
 x2 = middle  
 f2 = f\_middle  
  
 df.loc[len(df)] = (list(map(lambda x:  
 round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
  
 result = middle  
 return (result, df)

Приложение 2.3 – реализация метода секущих (в частности, алгоритм поиска решения)

class SecantMethod(Method):  
  
 def solve(self, epsilon: float) -> typing.Tuple[float, pd.DataFrame]:  
 df = self.dataframe  
  
 (x1, x2) = (self.segment.left\_border, self.segment.right\_border)  
 f1 = self.function(x1)  
 f2 = self.function(x2)  
 del\_x = abs(x2 - x1)  
 round\_base = self.get\_round\_base(epsilon)  
  
 middle = 0  
 while del\_x > epsilon:  
 df.loc[len(df)] = (list(map(lambda x: round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
 middle = x2 \* f1 / (f1 - f2) + x1 \* f2 / (f2 - f1)  
 f\_middle = self.function(middle)  
 x1 = x2  
 f1 = f2  
 x2 = middle  
 f2 = f\_middle  
 del\_x = abs(x2 - x1)  
  
 df.loc[len(df)] = (list(map(lambda x:  
 round(x, round\_base + 1),  
 [x1, f1, x2, f2, del\_x])))  
  
 result = middle  
 return (result, df)

Приложение 2.4 – реализация метода Ньютона (в частности, алгоритм поиска решения)

class NewtonMethod(Method):  
  
 def solve(self, epsilon: float) -> typing.Tuple[float, pd.DataFrame]:  
 df = self.dataframe  
  
 x = self.segment.middle  
 del\_x = epsilon \* 2  
 f1 = self.function(x)  
 df1 = self.derivative\_function(x)  
 round\_base = self.get\_round\_base(epsilon)  
 df.loc[len(df)] = (list(map(lambda k: round(k, round\_base + 1),  
 [x, f1, df1, 0])))  
  
 while del\_x > epsilon:  
 f1 = self.function(x)  
 df1 = self.derivative\_function(x)  
 x\_ = x - f1 / df1  
 del\_x = abs(x\_ - x)  
 x = x\_  
 df.loc[len(df)] = (list(map(lambda k: round(k, round\_base + 1),  
 [x, f1, df1, del\_x])))  
  
 result = x  
 return (result, df)