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 44

A4 = B7 =

(изменил коэффициент в части B с 2 на 1, так как при 2 не было решений)

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:

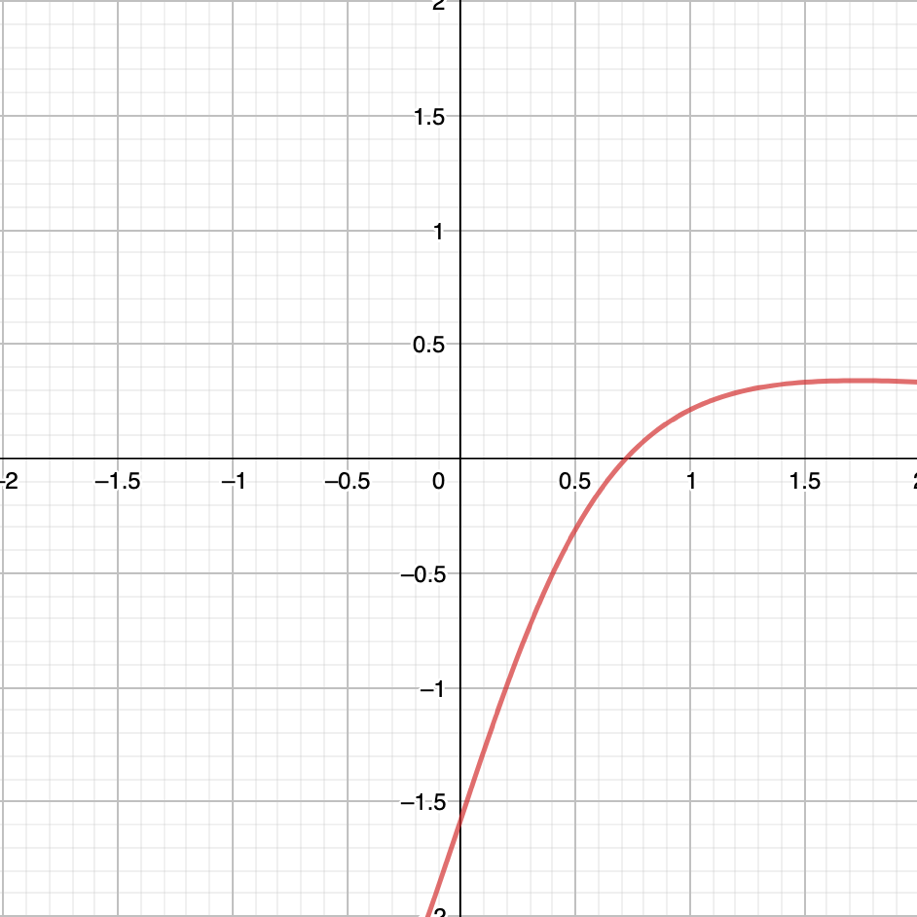


Рис. 1 - график функции f(x)

Поиск корней будет производиться на отрезке

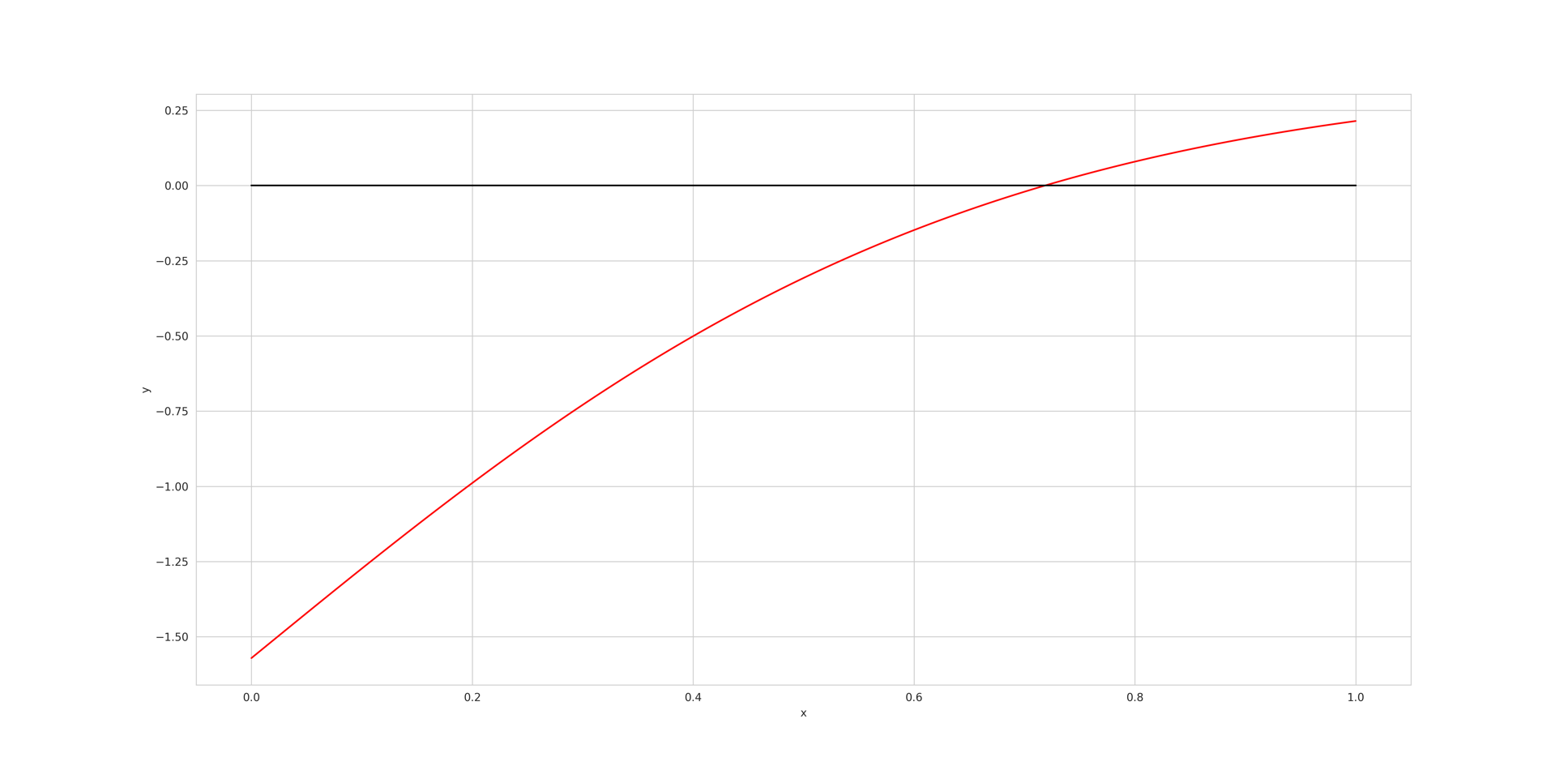


Рис. 2 - график функции f(x) на отрезке

# 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. На каждой итерации, считаем значение функции в середине отрезка и заменяем одну из границ на это значение в зависимости от знака. 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. Алгоритм займёт итераций, где длина изначального отрезка, а – желаемая точность. 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** | 0 | -1,5708 | 1 | 0,2146 | 1 |
| **2** | 0,5 | -0,30715 | 1 | 0,2146 | 0,5 |
| **3** | 0,5 | -0,30715 | 0,75 | 0,0327 | 0,25 |
| **4** | 0,625 | -0,11332 | 0,75 | 0,0327 | 0,125 |
| **5** | 0,6875 | -0,03482 | 0,75 | 0,0327 | 0,0625 |
| **6** | 0,6875 | -0,03482 | 0,71875 | 0,00025 | 0,03125 |
| **7** | 0,70312 | -0,01695 | 0,71875 | 0,00025 | 0,01562 |
| **8** | 0,71094 | -0,00827 | 0,71875 | 0,00025 | 0,00781 |
| **9** | 0,71484 | -0,00399 | 0,71875 | 0,00025 | 0,00391 |
| **10** | 0,7168 | -0,00187 | 0,71875 | 0,00025 | 0,00195 |
| **11** | 0,71777 | -0,00081 | 0,71875 | 0,00025 | 0,00098 |
| **12** | 0,71826 | -0,00028 | 0,71875 | 0,00025 | 0,00049 |
| **13** | 0,71851 | -0,00002 | 0,71875 | 0,00025 | 0,00024 |
| **14** | 0,71851 | -0,00002 | 0,71863 | 0,00011 | 0,00012 |
| **15** | 0,71851 | -0,00002 | 0,71857 | 0,00005 | 0,00006 |

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

### Вывод

Алгоритм дал ответ с точностью до 0.01 за 8 итераций, ответ с точностью до 0.001 за 11 итераций и ответ с точностью до 0.0001 за 15 итераций.

Ответ: 0.71851

## 2. Метод ложной позиции

### Описание метода

В методе ложной позиции, нелинейная функция заменяется на линейную функцию на интервале , где корень линейной функции берётся как следующее приближение корня нашей нелинейной функции . Таким образом, корень не является корнем данной нам нелинейной функции , отчего этот метод и получил своё название (ложная позиция). В итоге, у нас получается два интервала – Следующий интервал мы выбираем в зависимости от знака корня . Нахождение следующей точки (приближённое значение ):

В отличие от метода бисекции, интервал не уменьшается до одной точки, поэтому критерий остановы определяется близостью двух подряд идущих приближений корня. 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. На итерации мы высчитываем , и выбираем новый интервал, если , то и , иначе и .

### Результат работы алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **1** | 0 | -1,5708 | 1 | 0,2146 | 1 |
| **2** | 0 | -1,5708 | 0,8798 | 0,1426 | 0,1202 |
| **3** | 0 | -1,5708 | 0,80658 | 0,08528 | 0,07322 |
| **4** | 0 | -1,5708 | 0,76504 | 0,04744 | 0,04153 |
| **5** | 0 | -1,5708 | 0,74262 | 0,02526 | 0,02243 |
| **6** | 0 | -1,5708 | 0,73086 | 0,01313 | 0,01175 |
| **7** | 0 | -1,5708 | 0,7248 | 0,00673 | 0,00606 |
| **8** | 0 | -1,5708 | 0,72171 | 0,00343 | 0,00309 |
| **9** | 0 | -1,5708 | 0,72014 | 0,00174 | 0,00157 |
| **10** | 0 | -1,5708 | 0,71934 | 0,00088 | 0,0008 |
| **11** | 0 | -1,5708 | 0,71894 | 0,00045 | 0,0004 |
| **12** | 0 | -1,5708 | 0,71873 | 0,00023 | 0,0002 |
| **13** | 0 | -1,5708 | 0,71863 | 0,00011 | 0,0001 |
| **14** | 0 | -1,5708 | 0,71858 | 0,00006 | 0,00005 |

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

### Вывод

Алгоритм дал ответ с точностью до 0.01 за 7 итераций, ответ с точностью до 0.001 за 10 итераций и ответ с точностью до 0.0001 за 14 итераций.

Ответ: 0.71858

## 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 метод секущих между интервалами всегда выбирает который сформирован последними двумя точками, вне зависимости от знаков функции на концах.

### Алгоритм

Отрезок выбран такой же, как и при предыдущих методах.

### Результат работы алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **1** | 0 | -1,5708 | 1 | 0,2146 | 1 |
| **2** | 1 | 0,2146 | 0,8798 | 0,1426 | 0,1202 |
| **3** | 0,8798 | 0,1426 | 0,64173 | -0,09117 | 0,23807 |
| **4** | 0,64173 | -0,09117 | 0,73458 | 0,017 | 0,09284 |
| **5** | 0,73458 | 0,017 | 0,71998 | 0,00158 | 0,01459 |
| **6** | 0,71998 | 0,00158 | 0,71849 | -0,00003 | 0,00149 |
| **7** | 0,71849 | -0,00003 | 0,71852 | 0 | 0,00003 |

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 7 итераций и ответ с точностью до 0.0001 за 7 итераций. Таким образом, метод секущих оказался быстрее, чем метод ложной позиции.

Ответ: 0.71858

## 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 0.5. 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** | 0,5 | -0,30715 | 1,76 | 0 |
| **1** | 0,67452 | -0,30715 | 1,76 | 0,17452 |
| **2** | 0,71627 | -0,05019 | 1,20222 | 0,04175 |
| **3** | 0,71852 | -0,00245 | 1,08635 | 0,00225 |
| **4** | 0,71852 | -0,00001 | 1,08033 | 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 3 итерации, ответ с точностью до 0.001 за 4 итерации и ответ с точностью до 0.0001 снова за 4 итерации.

Ответ: 0.71852

# Вывод

Составим таблицу результатов различных методов:

|  |  |  |
| --- | --- | --- |
| Метод | *Количество итераций* | *Корень уравнения* |
| **Метод бисекции** | 15 | 0.71851 |
| **Метод ложной позиции** | 14 | 0.71858 |
| **Метод секущих** | 7 | 0.71858 |
| **Метод Ньютона** | 4 | 0.71852 |

Наглядно видно, что самым сложным для расчётов для нашей функции оказался метод бисекции, а самым быстрым – метод Ньютона. Стоит отметить, что метод секущих показал высокую скорость нахождения корня, но сравнительно не 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)