**Лабораторная работа №3**

**«Численное решение нелинейных уравнений»**

Чеботаревский Никита, 2 курс, 7группа

**Постановка задачи**

Написать программу, которая находит решение уравнения *f* (*x*) = 0 c точностью ε = методами, указанными в варианте задания. Корень отделяем сначала графически, затем с помощью метода половинного деления с точностью ε = 0.1. Провести сравнительный анализ полученных результатов.

**В содержание отчета должна быть включена следующая информация***:*

Графики, которые использовались для отделения корня. Отрезок отделенного

корня.

Алгоритм метода половинного деления. Сводные данные по результатам работы

метода половинного деления, оформленные в виде таблицы 1 (см. ниже).

Алгоритмы методов, применяемые для нахождения корня уравнения с заданной

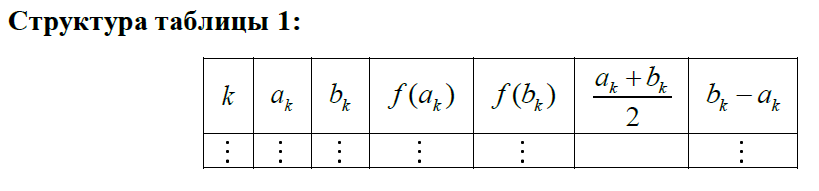
точностью ε. Использовать в качестве отрезка отделенного корня суженный отрезок,

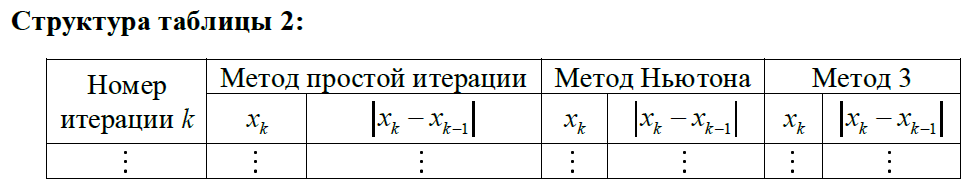
полученный с помощью метода половинного деления.

Проверка условий теоремы о сходимости метода простой итерации. Проверка

условий теоремы о сходимости метода Ньютона.

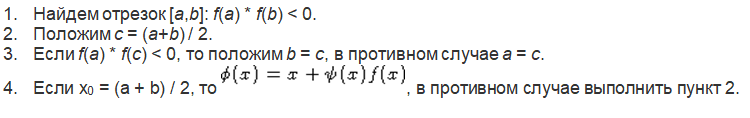
Сводные данные по результатам работы методов, оформленные в виде таблицы 2





**Теоретические сведения**

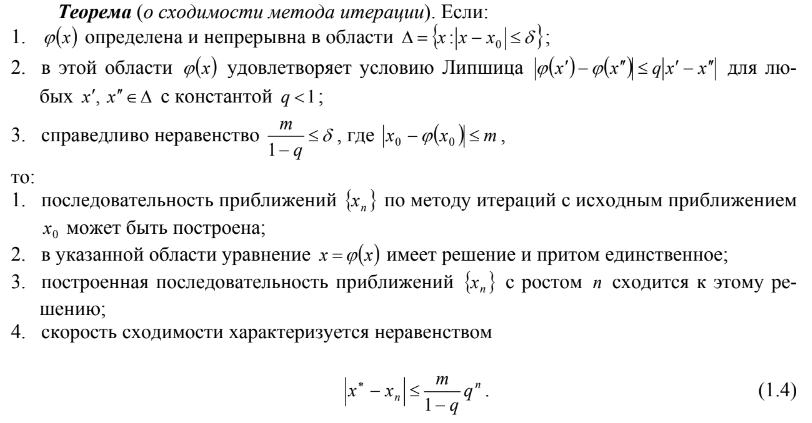
1. **Метод половинного деления (дихотомии)**:



1. **Метод простой итерации**:

Пусть - корень и он отделён, и, таким образом, указано некоторое начальное приближение (произвольное значение из отрезка отделённости). Тогда уточнение этого значения производят по правилу:



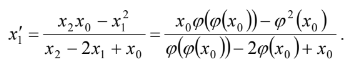


1. **Метод Стеффенсена**:

По приближению построим следующие приближения:

** и 

Применив к трём числам преобразование Эйткена, получим:

**

Заменив соответствующим образом индексы, получим:



1. **Метод Ньютона с постоянной производной**:

Для наилучшей сходимости в точке очередного приближения должно выполняться условие  Решение данного уравнения ищут в виде:



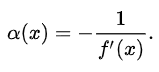
Тогда:



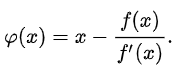
В предположении, что точка приближения достаточно близка к корню x,

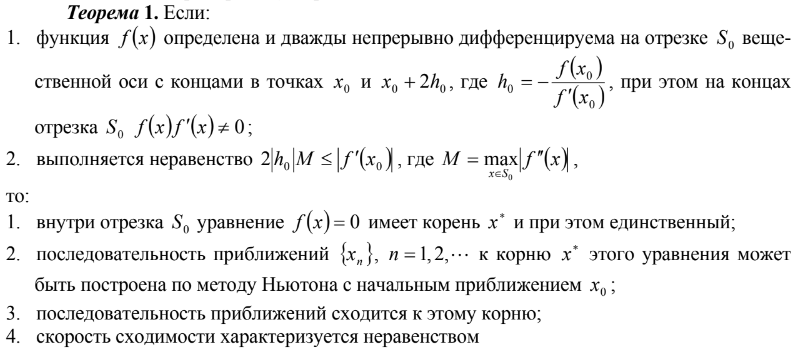
И что заданная функция непрерывна, то окончательная формула для

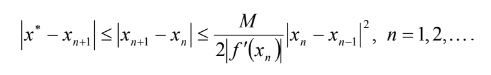
примет вид:

**

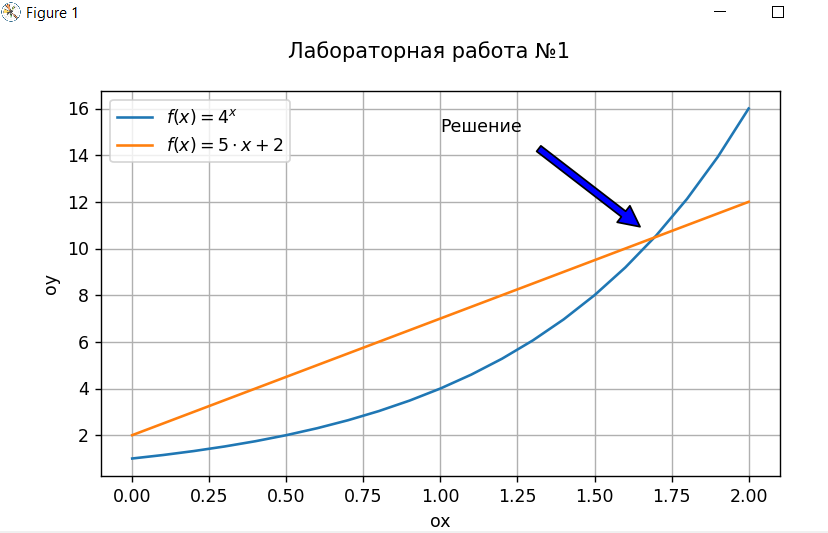
С учётом этого формула для примет вид:

**



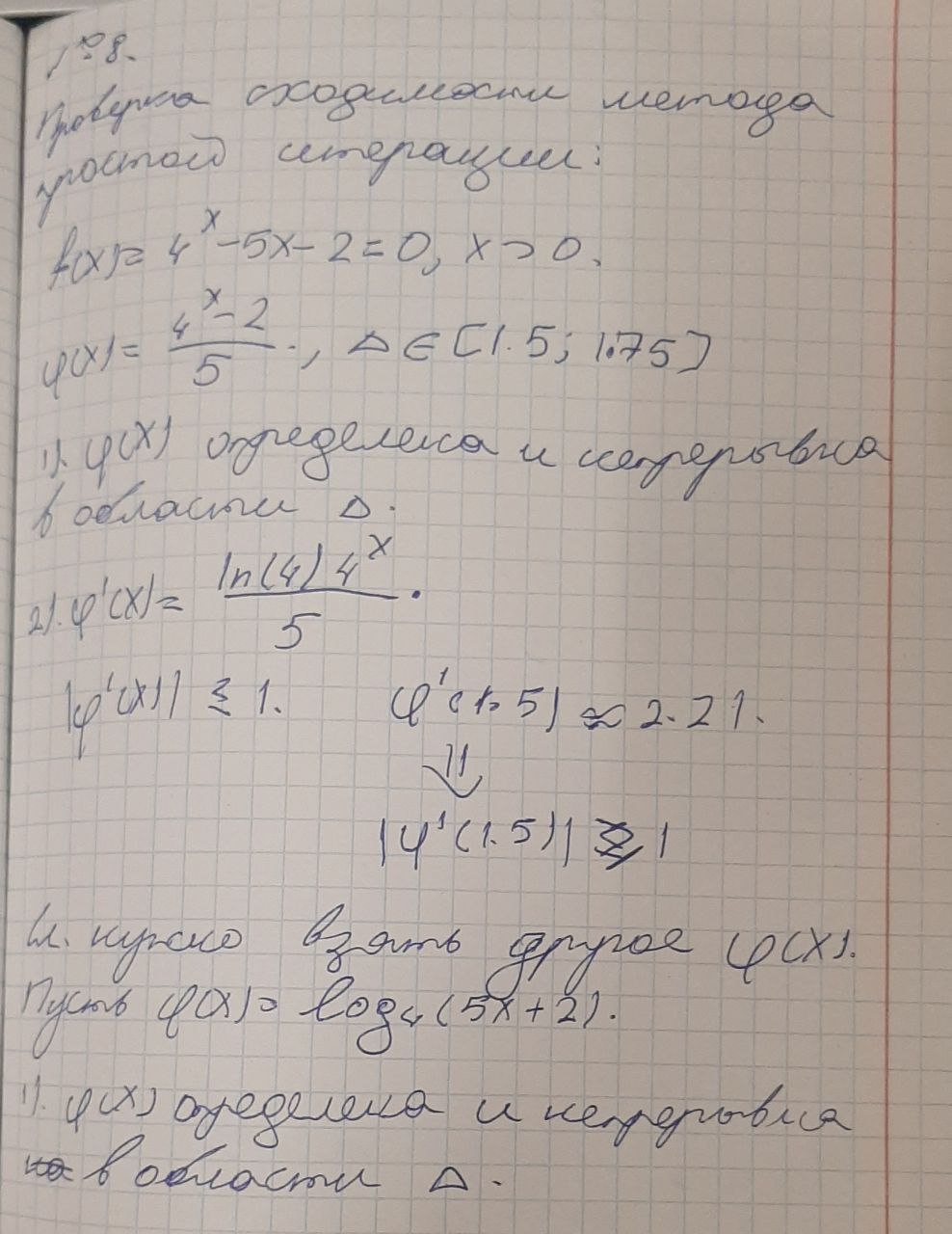


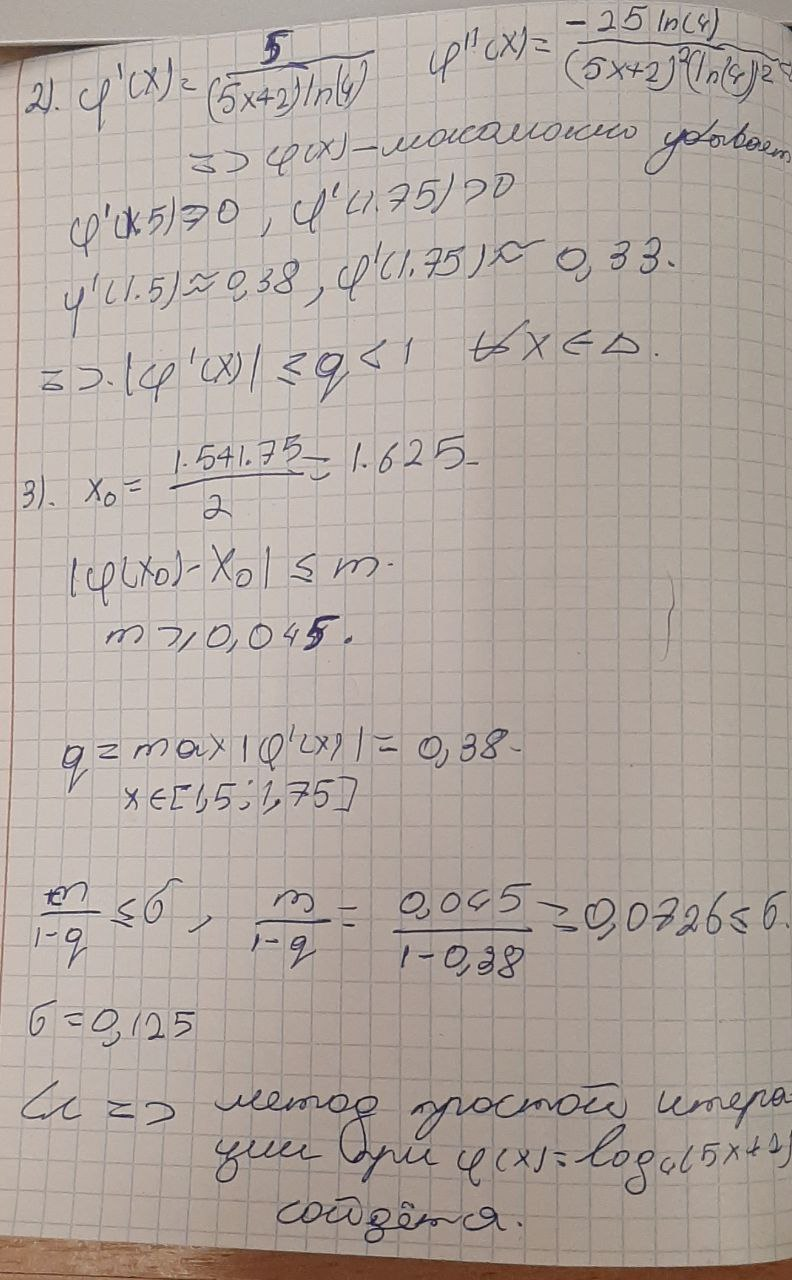
**График функции**

****

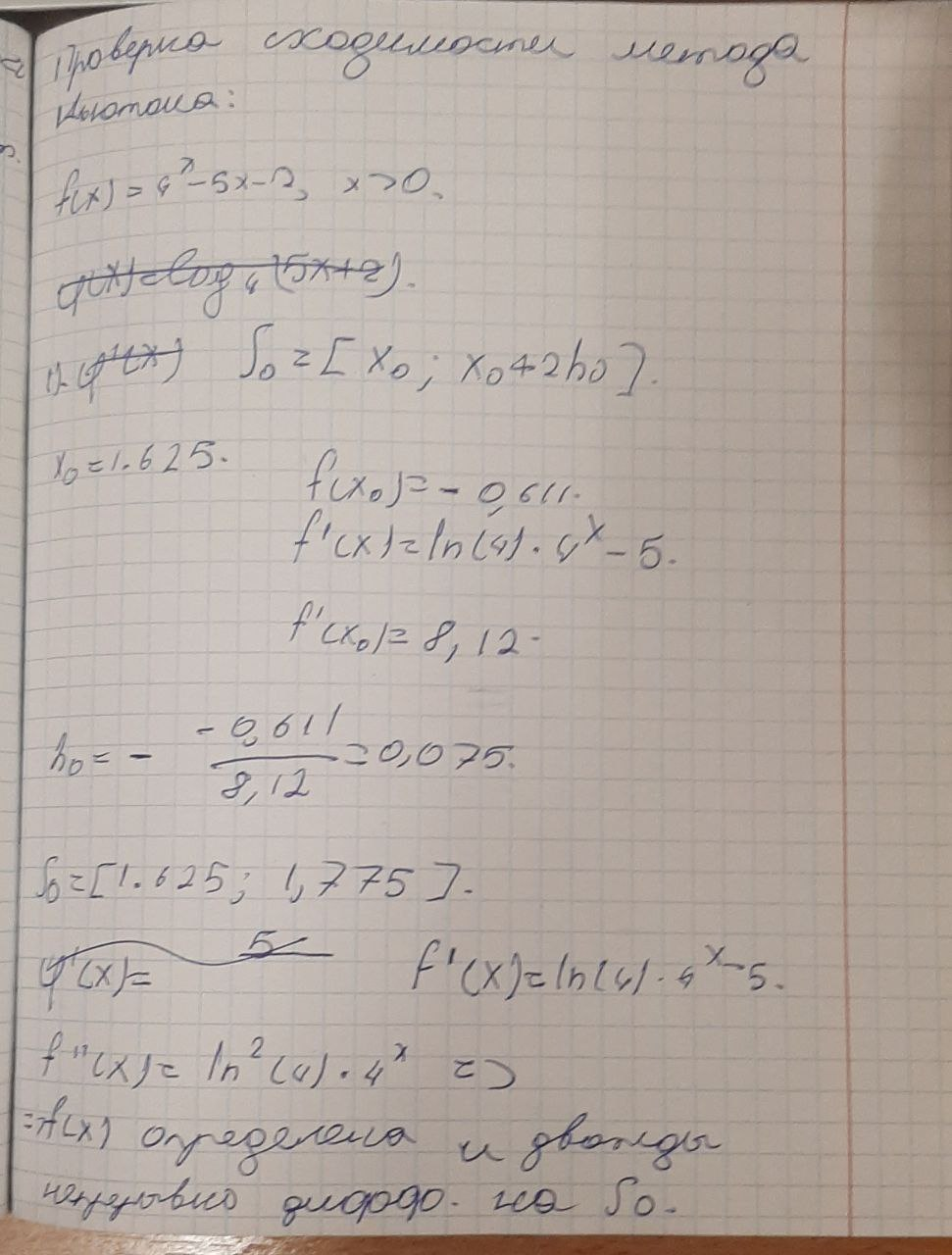
**Проверка условий теорем**

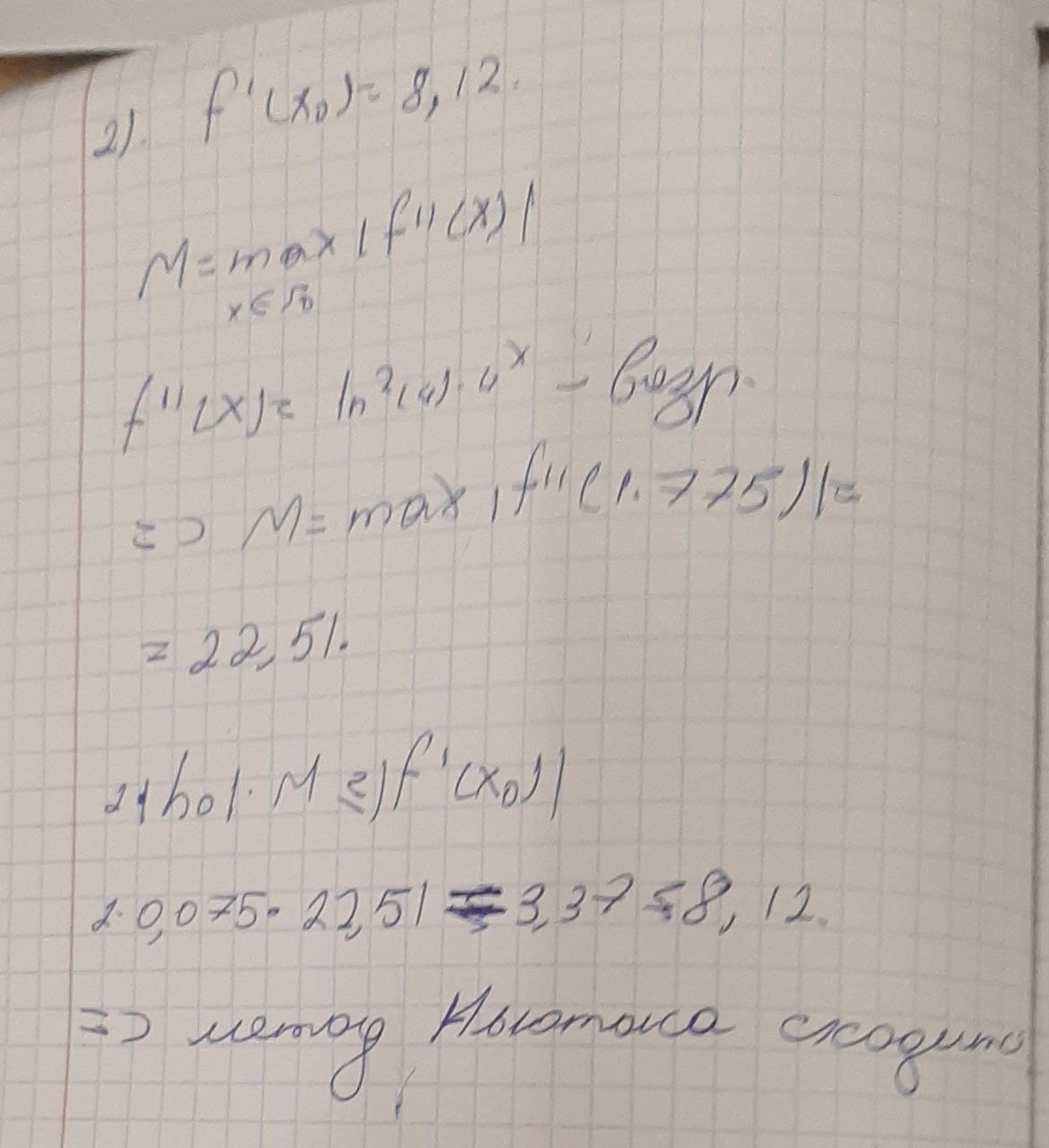
**Для метода простой итерации**





**Для метода Ньютона**





**Листинг программы**

*from* math *import* \*  
*import* matplotlib.pyplot *as* plt  
*import* pandas *as* pd  
*import* numpy *as* np  
  
  
*def* my\_function(number):  
 *return* 4 \*\* number - 5 \* number - 2  
  
  
*def* count\_phi\_for\_simple\_iteration(number):  
 *return* log(5 \* number + 2, 4)  
  
  
*def* derivative\_of\_the\_function(number):  
 *return* (4 \*\* number) \* log(4) - 5  
  
  
*def* method\_of\_half\_division(number0, number1, eps):  
 print("Method of half division: ")  
 data = []  
 list\_of\_pairs = []  
 number\_of\_iteration = 0  
 *while* number1 - number0 > 2 \* eps:  
 number\_of\_iteration += 1  
 number2 = (number0 + number1) / 2  
 data.append([number0, number1, my\_function(number0), my\_function(number1), number2, number1 - number0])  
  
 list\_of\_pairs.append((number0, number1))  
 *if* my\_function(number2) \* my\_function(number0) < 0:  
 number1 = number2  
 *else*:  
 number0 = number2  
 *return* list\_of\_pairs[-1], data, np.arange(1, number\_of\_iteration + 1)  
  
  
*def* method\_of\_simple\_iteration(x, eps):  
 print("Method of simple iteration: ")  
 x\_res = count\_phi\_for\_simple\_iteration(x)  
 data = []  
 number\_of\_iteration = 1  
 data.append([x\_res, abs(x\_res - x)])  
 *while* abs(x\_res - x) > eps:  
 x = x\_res  
 x\_res = count\_phi\_for\_simple\_iteration(x)  
 number\_of\_iteration += 1  
 data.append([x\_res, abs(x\_res - x)])  
  
 *return* data, np.arange(1, number\_of\_iteration + 1)  
  
  
*def* newton\_method(x, eps):  
 print("The method of Newton: ")  
 data = []  
 x\_res = x - my\_function(x) / derivative\_of\_the\_function(x)  
 number\_of\_iteration = 1  
 data.append([x\_res, abs(x\_res - x)])  
 *while* abs(x\_res - x) > eps:  
 x = x\_res  
 x\_res = x - my\_function(x) / derivative\_of\_the\_function(x)  
 number\_of\_iteration += 1  
 data.append([x\_res, abs(x\_res - x)])  
  
 *return* data, np.arange(1, number\_of\_iteration + 1)  
  
  
*def* steffensen\_method(x, eps):  
 print("Method of Steffensen: ")  
 numerator = x \* count\_phi\_for\_simple\_iteration(count\_phi\_for\_simple\_iteration(x)) - \  
 count\_phi\_for\_simple\_iteration(x) \*\* 2  
 denominator = count\_phi\_for\_simple\_iteration(count\_phi\_for\_simple\_iteration(x)) - \  
 2 \* count\_phi\_for\_simple\_iteration(x) + x  
 x\_res = numerator / denominator  
  
 data = []  
 number\_of\_iteration = 1  
 data.append([x\_res, abs(x\_res - x)])  
  
 *while* abs(x\_res - x) > eps:  
 x = x\_res  
 numerator = x \* count\_phi\_for\_simple\_iteration(count\_phi\_for\_simple\_iteration(x)) - \  
 count\_phi\_for\_simple\_iteration(x) \*\* 2  
 denominator = count\_phi\_for\_simple\_iteration(count\_phi\_for\_simple\_iteration(x)) - \  
 2 \* count\_phi\_for\_simple\_iteration(x) + x  
 x\_res = numerator / denominator  
  
 number\_of\_iteration += 1  
 data.append([x\_res, abs(x\_res - x)])  
  
 *return* data, np.arange(1, number\_of\_iteration + 1)  
  
  
*def* print\_plot():  
 fig = plt.figure(figsize=(7, 4))  
 ax = fig.add\_subplot()  
 fig.suptitle("Лабораторная работа №1")  
 line1, = ax.plot(np.arange(0, 2.1, 0.1), [4 \*\* x *for* x *in* np.arange(0, 2.1, 0.1)])  
 line2, = ax.plot(np.arange(0, 2.1, 0.1), [5 \* x + 2 *for* x *in* np.arange(0, 2.1, 0.1)])  
  
 ax.set\_xlabel("ox")  
 ax.set\_ylabel("oy")  
 ax.legend((line1, line2), [r'$f(x) = 4^x$', r'$f(x) = 5 \cdot x + 2$'])  
 ax.annotate("Решение", xy=(1.69, 10.5), xytext=(1, 15), arrowprops={'facecolor': 'blue', 'shrink': 0.1})  
  
 plt.grid()  
 plt.show()  
  
  
*def* print\_data\_frame(data, index, name):  
 print(pd.DataFrame(data, index=index, columns=name))  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 print\_plot()  
  
 names\_simple\_iteration = ['ak', 'bk', 'f(ak)', 'f(bk)', '(ak + bk) / 2', 'bk - ak']  
 names = ['xk', '|xk - x(k - 1)|']  
  
 e = 10 \*\* (-1)  
 e\_for\_method = 10 \*\* (-6)  
 x0, x1 = 1, 2  
 pair\_x01, information, order = method\_of\_half\_division(x0, x1, e)  
 print\_data\_frame(information, order, names\_simple\_iteration)  
 information, order = method\_of\_simple\_iteration((pair\_x01[0] + pair\_x01[1]) / 2, e\_for\_method)  
 print\_data\_frame(information, order, names)  
 information, order = newton\_method((pair\_x01[0] + pair\_x01[1]) / 2, e\_for\_method)  
 print\_data\_frame(information, order, names)  
 information, order = steffensen\_method((pair\_x01[0] + pair\_x01[1]) / 2, e\_for\_method)  
 print\_data\_frame(information, order, names)

**Результаты программы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **k** | **ak** | **bk** | **f(ak)** | **f(bk)** | **(ak + bk)**  **/ 2** | **bk - ak** |
| 1 | 1 | 2 | -3 | 4 | 1.5 | 1 |
| 2 | 1.5 | 2 | -1.5 | 4 | 1.75 | 0.5 |
| 3 | 1.5 | 1.75 | -1.5 | 0.5637084989847612 | 1.625 | 0.25 |

Method of simple iteration:

xk |xk - x(k - 1)|

1 1.669925 4.492500e-02

2 1.685753 1.582829e-02

3 1.691248 5.495014e-03

4 1.693146 1.897925e-03

5 1.693801 6.543663e-04

6 1.694026 2.254748e-04

7 1.694104 7.767546e-05

8 1.694131 2.675705e-05

9 1.694140 9.216835e-06

10 1.694143 3.174839e-06

11 1.694144 1.093604e-06

12 1.694144 3.767024e-07

The method of Newton:

xk |xk - x(k - 1)|

1 1.699657 7.465665e-02

2 1.694176 5.480166e-03

3 1.694145 3.191563e-05

4 1.694145 1.077074e-09

Method of Steffensen:

xk |xk - x(k - 1)|

1 1.694364 6.936372e-02

2 1.694145 2.191462e-04

3 1.694144 2.565286e-07

**Вывод**

С помощью всех методов, рассмотренных выше, можно решить уравнение

f(x) = 0 с заданной точностью. Однако они все сходятся с разной скоростью.

Метод Ньютона сходится быстрее всего, а простой итерации – медленнее.

Также для методов Ньютона и простой итерации необходимо проверить

условия сходимости итерационных процессов.