

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего
профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЁТ

по лабораторной работе №1

" РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ С ОДНОЙ НЕИЗВЕСТНОЙ"

по дисциплине

Вычислительная математика

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

(подпись)

Панкратова А.З.
(фамилия, и.,о.)

СТУДЕНТ:

(подпись)

Халеев А.А.
(фамилия, и.,о.)

21-ВМз-4
(шифр группы)

Работа защищена «__» _____

С оценкой _____

Нижний Новгород

2023

Тема работы:

Решение нелинейных уравнений с одной неизвестной.

Цель работы:

Изучить численные методы и алгоритмы решения нелинейных уравнений.

Постановка задачи:

Решить нелинейное уравнение с одним неизвестным с использованием трех методов (метод половинного деления, метод Ньютона, метод простой итерации). Задание по вариантам. Точность $\varepsilon=0.001$

Вариант №7:

$$x^3 + 0.2x^2 + 0.5x - 1.2 = 0$$

Шаговый метод

Дано уравнение $f(x) = 0$. Задан интервал поиска $[x_0, x_1]$. Требуется найти интервал $[a, b]$ длиной h , содержащий первый корень уравнения, начиная с левой границы интервала поиска.

Алгоритм метода:

1. Установить интервал $[a, b]$ на начало интервала поиска ($a=x_0$).
2. Определить координату точки b ($b=a+h$), а также значения функции в точках a и b : $F(a)$ и $F(b)$.
3. Проверить условие $F(a)*F(b)<0$. Если условие не выполнено – передвинуть интервал $[a, b]$ на один шаг ($a=b$) и перейти к пункту 2. Если условие выполнено - закончить алгоритм.

Решение в Excel:

Шаговый метод	
Начальное значение	0,8
Шаг табуляции	0,01
x	f(x)
0,8	-0,16000
0,81	-0,13234
0,82	-0,10415
0,83	-0,07543
0,84	-0,04618
0,85	-0,01637
0,86	0,01398
0,87	0,04488
0,88	0,07635
0,89	0,10839
0,9	0,14100

Программа на Python:

```
def f(x: float) -> float:
    """
    Вычисляет значение функции  $f(x) = x^3 + 0.2 * x^2 + 0.5 * x - 1.2$ 

    Параметры:
        x: Значение аргумента функции.

    Возвращаемое значение:
        Значение функции в точке x.
    """
    return x ** 3 + 0.2 * x ** 2 + 0.5 * x - 1.2

def step_method(func: callable, start: float, step: float) -> tuple[int, int]:
    """
    Находит отрезок, содержащий корень функции.

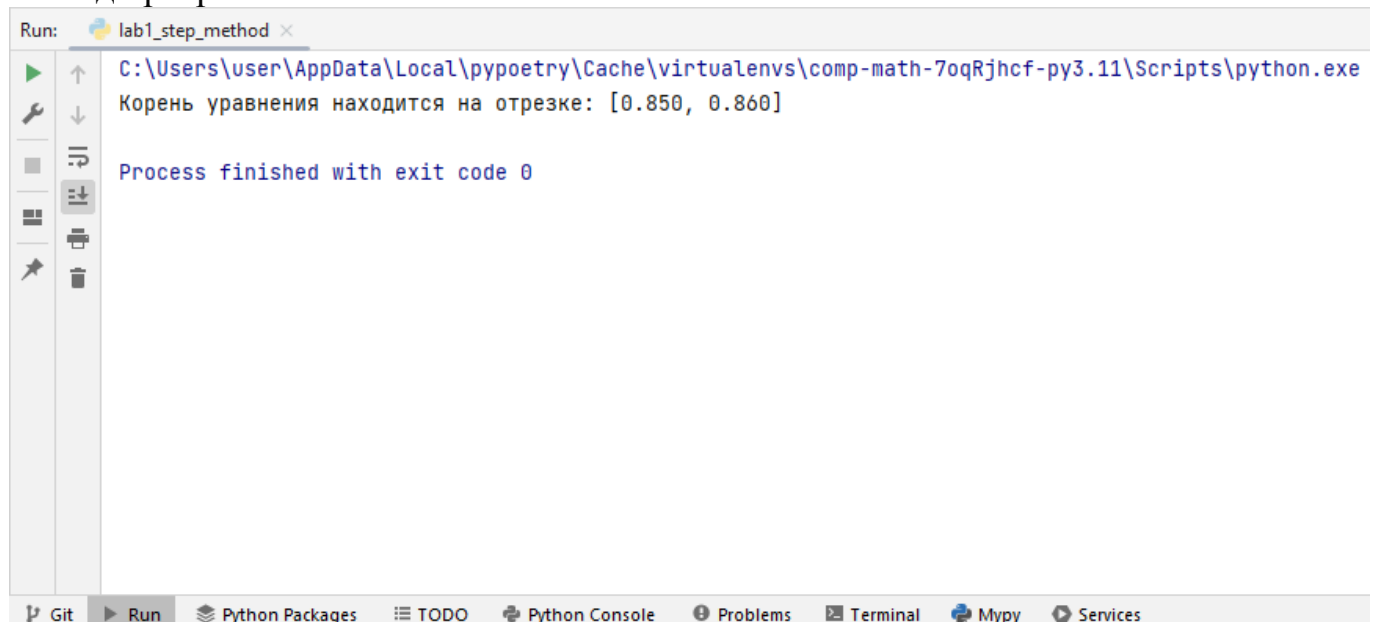
    Параметры:
        func: Функция  $f(x)$ , корень которой необходимо найти.
        start: Начало отрезка поиска.
        step: Шаг при переборе точек на отрезке.

    Возвращаемое значение:
        Кортеж с началом и концом сегмента, содержащего корень функции.
    """
    x0, x1 = start, start + step
    while func(x0) * func(x1) >= 0:
        x0, x1 = x1, x1 + step
    return round(x0, 2), round(x1, 2)

def main() -> None:
    # Использование методов
    a, b = step_method(func=f, start=0.8, step=0.01)
    print(f"Корень уравнения находится на отрезке: [{a:.3f}, {b:.3f}]")

if __name__ == "__main__":
    main()
```

Вывод программы:



Таким образом, на отрезке $[0.85; 0.86]$ существует единственный корень уравнения $x^3 + 0.2x^2 + 0.5x - 1.2 = 0$ рассмотренного на интервале $[0.80; 0.90]$.

После того как найден интервал, содержащий корень, применяют итерационные методы уточнения корня с заданной точностью.

Мы разберем следующие методы:

1. Метод половинного деления
2. Метод Ньютона (метод касательных)
3. Метод простой итерации (Якоби)

Метод половинного деления

Метод основан на последовательном сужении интервала, содержащего единственный корень уравнения $f(x) = 0$ до тех пор, пока не будет достигнута заданная точность ε . Пусть задан отрезок $[a, b]$, содержащий один корень уравнения. Этот отрезок может быть предварительно найден с помощью шагового метода.

Алгоритм

метода:

1. Определить новое приближение корня x в середине отрезка $[a, b]$: $x = (a + b) / 2$.
2. Найти значения функции в точках a и x : $f(a)$ и $f(x)$.
3. Проверить условие $f(a) \cdot f(x) < 0$. Если условие выполнено, то корень расположен на отрезке $[a, x]$. В этом случае необходимо точку b переместить в точку x ($b = x$). Если условие не выполнено, то корень расположен на отрезке $[x, b]$. В этом случае необходимо точку a переместить в точку x ($a = x$).
4. Перейти к пункту 1 и вновь поделить отрезок пополам. Алгоритм продолжить до тех пор, пока не будет выполнено условие $f(x) < \varepsilon$.

Решение в Excel:

Метод половинного деления					
Начальное значение	0,85				
Шаг табуляции	нет				
Точность	0,001				
a	x	b	f(a)	f(x)	f(a)*f(x)<0
0,8500	0,8550	0,8600	-0,0164	-0,0013	НЕТ
0,8550	0,8575	0,8600	-0,0013	0,0063	ДА
0,8550	0,8563	0,8575	-0,0013	0,0025	ДА
0,8550	0,8556	0,8563	-0,0013	0,0006	СТОП

Программа на Python:

```
def f(x: float) -> float:
    """
    Вычисляет значение функции  $f(x) = x^3 + 0.2 * x^2 + 0.5 * x - 1.2$ 

    Параметры:
        x: Значение аргумента функции.

    Возвращаемое значение:
        Значение функции в точке x.
    """
    return x ** 3 + 0.2 * x ** 2 + 0.5 * x - 1.2

def step_method(func: callable, start: float, step: float) -> tuple[int, int]:
    """
    Находит отрезок, содержащий корень функции.

    Параметры:
        func: Функция  $f(x)$ , корень которой необходимо найти.
        start: Начало отрезка поиска.
        step: Шаг при переборе точек на отрезке.

    Возвращаемое значение:
        Кортеж с началом и концом сегмента, содержащего корень функции.
    """
    x0, x1 = start, start + step
    while func(x0) * func(x1) >= 0:
        x0, x1 = x1, x1 + step
    return round(x0, 2), round(x1, 2)

def bisection_method(func: callable, a: float, b: float, epsilon: float) -> float:
    """
    Реализует метод половинного деления для численного решения уравнения
     $f(x) = 0$  на заданном отрезке  $[a, b]$ .

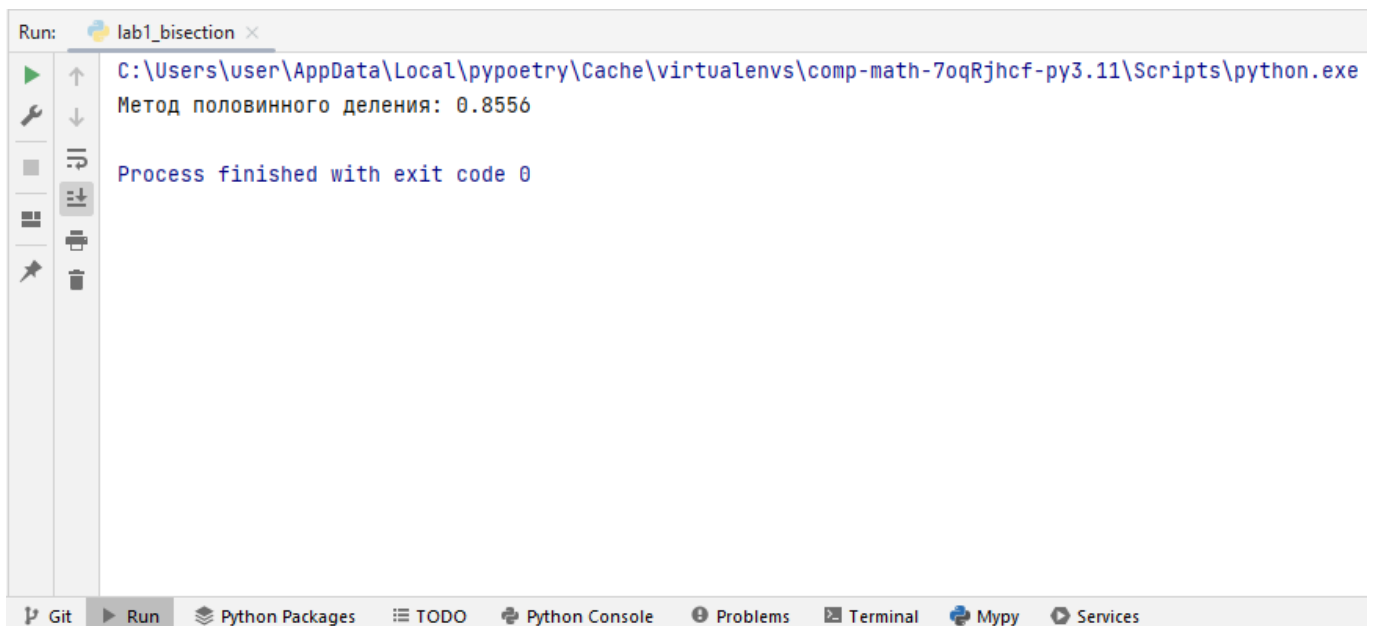
    Параметры:
        a: Начало отрезка.
        b: Конец отрезка.
        epsilon: Точность решения.

    Возвращаемое значение:
        Приближенное значение корня уравнения  $f(x) = 0$ .
    """
    while abs(func(x := ((a + b) / 2))) >= epsilon:
        if func(x) * func(a) < 0:
            b = x
        else:
            a = x
    return x

def main():
    # Использование методов
    a, b = step_method(func=f, start=0.8, step=0.01)
    root_bisection = bisection_method(f, a, b, epsilon=0.001)
    print(f"Метод половинного деления: {root_bisection:.4f}")

if __name__ == "__main__":
    main()
```

Вывод программы:



The screenshot shows the 'Run' console of a code editor. The title bar indicates the file 'lab1_bisection'. The console output is as follows:

```
C:\Users\user\AppData\Local\pypoetry\Cache\virtualenvs\comp-math-7oqRjhcf-py3.11\Scripts\python.exe
Метод половинного деления: 0.8556

Process finished with exit code 0
```

The bottom of the image shows the editor's sidebar with tabs for 'Git', 'Run', 'Python Packages', 'TODO', 'Python Console', 'Problems', 'Terminal', 'Mypy', and 'Services'.

Достоинство метода: более быстрая сходимость к заданной точности, чем у шагового.

Недостаток: если на отрезке $[a, b]$ содержится более одного корня, то метод не работает.

Метод Ньютона (метод касательных)

Задан отрезок $[a, b]$, содержащий корень $f(x) = 0$. Уточнение значения корня производится путем использования уравнения касательной. В качестве начального приближения задается тот из концов отрезка $[a, b]$, где значение функции и ее второй производной имеют одинаковые знаки (т.е. выполняется условие $f(x_0) * f''(x_0) > 0$). В точке $f(x_0)$ строится касательная к кривой $y = F(x)$ и ищется ее пересечение с осью x . Точка пересечения принимается за новую итерацию. Итерационная формула имеет вид:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Итерационный процесс продолжается до тех пор, пока не будет выполнено условие:

$$|f(x)| < \varepsilon, \text{ где } \varepsilon - \text{ заданная точность.}$$

Решение в Excel:

X	0,8	0,9
$f(x) = x^3 + 0.2x^2 + 0.5x - 1.2$	-0,16	0,141
$f'(x) = 3x^2 + 0.4x + 0.5$	2,74	3,29
$f''(x) = 6x + 0.4$	5,2	5,8
Так как в правой точке отрезка вторая производная и значение функции имеют одинаковые знаки, то начальной точкой выбрана правая граница отрезка, содержащего корень: $x_0 = 0.9$		

Метод Ньютона	
Начальное значение	0,9
Шаг табуляции	нет
Точность	0,001
$x_{i+1} = x_i - \frac{x_i^3 + 0.2x_i^2 + 0.5x_i - 1.2}{3x_i^2 + 0.4x_i + 0.5}$	
x_{i+1}	$f(x_{i+1})$
0,9000	0,1410
0,8571	0,0052478
0,8554	0,0000082

Программа на Python:

```
from scipy.misc import derivative
import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

def f(x: float) -> float:
    """
    Вычисляет значение функции  $f(x) = x^3 + 0.2 * x^2 + 0.5 * x - 1.2$ 

    Параметры:
        x: Значение аргумента функции.

    Возвращаемое значение:
        Значение функции в точке x.
    """
    return x ** 3 + 0.2 * x ** 2 + 0.5 * x - 1.2

def step_method(func: callable, start: float, step: float) -> tuple[int, int]:
    """
    Находит отрезок, содержащий корень функции.

    Параметры:
        func: Функция  $f(x)$ , корень которой необходимо найти.
        start: Начало отрезка поиска.
        step: Шаг при переборе точек на отрезке.

    Возвращаемое значение:
        Кортеж с началом и концом отрезка, содержащего корень функции.
    """
    x0, x1 = start, start + step
    while func(x0) * func(x1) >= 0:
        x0, x1 = x1, x1 + step
    return x0, x1

def newton_method(func: callable, a: float, b: float, epsilon: float) -> float:
    """
    Реализует метод Ньютона (метод касательных)
    для численного решения уравнения  $f(x) = 0$  на заданном отрезке  $[a, b]$ .

    Параметры:
        func: Функция  $f(x)$ , корень которой необходимо найти.
        a: Начало отрезка.
        b: Конец отрезка.
        epsilon: Точность решения.

    Возвращаемое значение:
        Приближенное значение корня уравнения  $f(x) = 0$ .
    """
    x = a if func(a) * derivative(func, x0=a, n=2) > 0 else b # начальное приближение
    next_value = lambda x: x - func(x) / derivative(func, x0=x) # итерационная формула

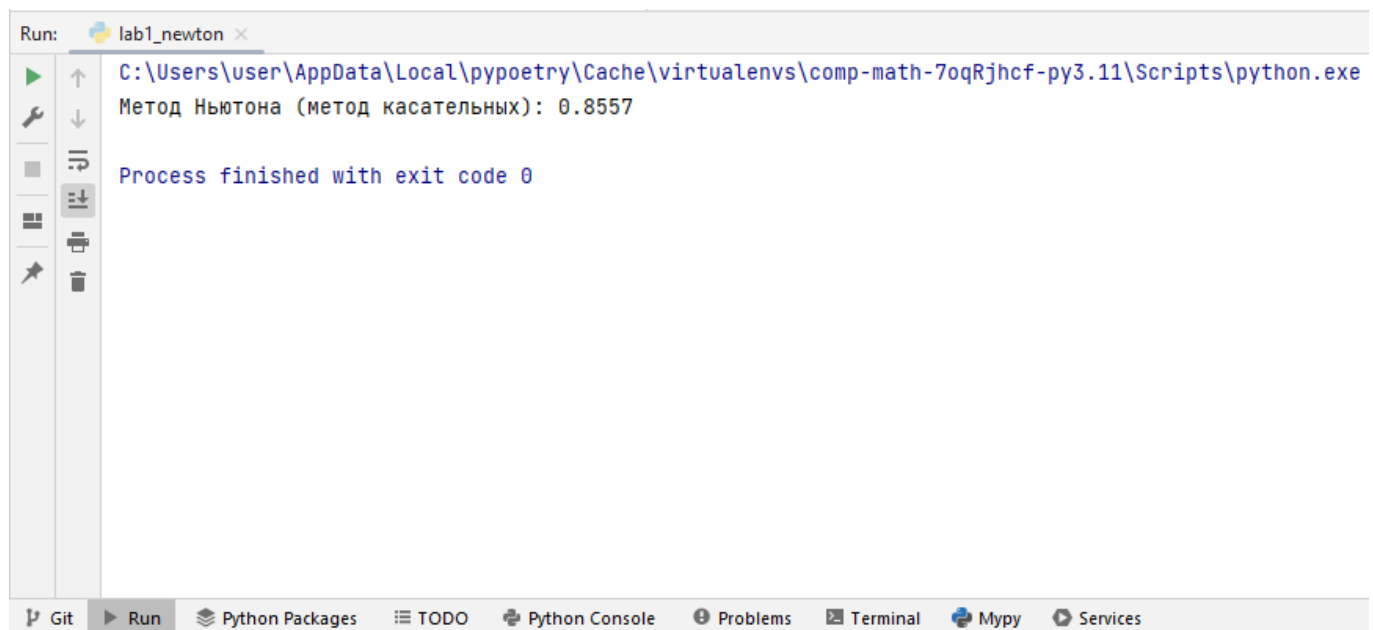
    while func(x) >= epsilon:
        x = next_value(x)

    return x

def main():
    # Использование методов
    a, b = step_method(func=f, start=0.8, step=0.01)
    root_newton = newton_method(f, a, b, epsilon=0.001)
    print(f"Метод Ньютона (метод касательных): {root_newton:.4f}")

if __name__ == "__main__":
    main()
```

Вывод программы:



The screenshot shows the 'Run' console in Visual Studio Code. The title bar indicates the file 'lab1_newton'. The console output shows the execution of a Python script using the interpreter 'C:\Users\user\AppData\Local\pypoetry\Cache\virtualenvs\comp-math-7oqRjhcf-py3.11\Scripts\python.exe'. The output text is 'Метод Ньютона (метод касательных): 0.8557', followed by 'Process finished with exit code 0'. The left sidebar contains icons for Run and Debug. The bottom status bar shows tabs for Git, Run, Python Packages, TODO, Python Console, Problems, Terminal, Mypy, and Services.

```
Run: lab1_newton x
C:\Users\user\AppData\Local\pypoetry\Cache\virtualenvs\comp-math-7oqRjhcf-py3.11\Scripts\python.exe
Метод Ньютона (метод касательных): 0.8557
Process finished with exit code 0
```

Достоинство метода: очень быстрая сходимость к заданной точности.

Недостаток: громоздкий алгоритм, на каждой итерации необходимо вычислять значение функции и ее первой производной.

Метод простой итерации (Якоби)

Метод основан на замене исходного уравнения $f(x) = 0$ на эквивалентное $x = \phi(x)$.

Функция $\phi(x)$ выбирается таким образом, чтобы на обоих концах отрезка $[a, b]$ выполнялось условие сходимости $|\phi'(x)| < 1$. В этом случае в качестве начального приближения можно выбрать любой из концов отрезка.

Итерационная формула имеет вид:

$$x_{i+1} = \phi(x_i)$$

Итерационный процесс продолжается до тех пор, пока не будет выполнено условие:

$$|f(x)| < \varepsilon, \text{ где } \varepsilon - \text{ заданная точность.}$$

На первом этапе нам необходимо выбрать функцию $\phi(x)$, удовлетворяющую условию сходимости.

Исходное уравнение:

$$x^3 + 0.2x^2 + 0.5x - 1.2 = 0$$

Запишем исходное уравнение в виде:

$$x = \sqrt[3]{1.2 - 0.2x^2 - 0.5x}$$

Тогда:

$$\phi(x) = \sqrt[3]{1.2 - 0.2x^2 - 0.5x}$$

$$\phi'(x) = \frac{-0.4x - 0.5}{3(1.2 - 0.2x^2 - 0.5x)^{\frac{2}{3}}}$$

$$\phi'(0.8) \approx 0.0782; \quad \phi'(0.9) \approx 0.0665$$

Условие сходимости выполнено, поскольку $|0.0782| < 1$ и $|0.0665| < 1$

Следовательно, итерационная формула имеет вид:

$$x_{i+1} = \sqrt[3]{1.2 - 0.2x_i^2 - 0.5x_i}$$

В качестве начального приближения можно выбрать любой из концов отрезка,
например: $x_0 = a = 0.8$

Решение в Excel:

Метод простой итерации (Якоби)	
Начальное значение	0,8
Шаг табуляции	нет
Точность	0,001
Эквивалентная формула $x_{i+1} = \sqrt[3]{1.2 - 0.2x_i^2 - 0.5x_i}$	
$\phi(x_i)$	$f(x_{i+1})$
0,8759	0,0634
0,8474	-0,0240
0,8585	0,0093
0,8542	-0,0035
0,8559	0,0014
0,8552	-0,0005

Программа на Python:

```
from scipy.misc import derivative
import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

def f(x: float) -> float:
    """
    Вычисляет значение функции  $f(x) = x^3 + 0.2 * x^2 + 0.5 * x - 1.2$ 

    Параметры:
        x: Значение аргумента функции.

    Возвращаемое значение:
        Значение функции в точке x.
    """
    return x ** 3 + 0.2 * x ** 2 + 0.5 * x - 1.2

def step_method(func: callable, start: float, step: float) -> tuple[int, int]:
    """
    Находит отрезок, содержащий корень функции.

    Параметры:
        func: Функция  $f(x)$ , корень которой необходимо найти.
        start: Начало отрезка поиска.
        step: Шаг при переборе точек на отрезке.

    Возвращаемое значение:
        Кортеж с началом и концом сегмента, содержащего корень функции.
    """
    x0, x1 = start, start + step
    while func(x0) * func(x1) >= 0:
        x0, x1 = x1, x1 + step
    return round(x0, 2), round(x1, 2)

def simple_iteration_method(func: callable, a: float, b: float, epsilon: float) -> float:
    """
    Реализует метод простых итераций для численного решения уравнения
     $f(x) = 0$  на заданном отрезке  $[a, b]$ .
    Реализовано для функции  $f(x) = x^3 + 0.2 * x^2 + 0.5 * x - 1.2 = 0$ 
    Тогда  $g(x) = (1.2 - 0.2 * x^2 - 0.5 * x) ** (1 / 3)$  является эквивалентной,
    так как  $g'(x)$  меньше 1 для обоих концов отрезка

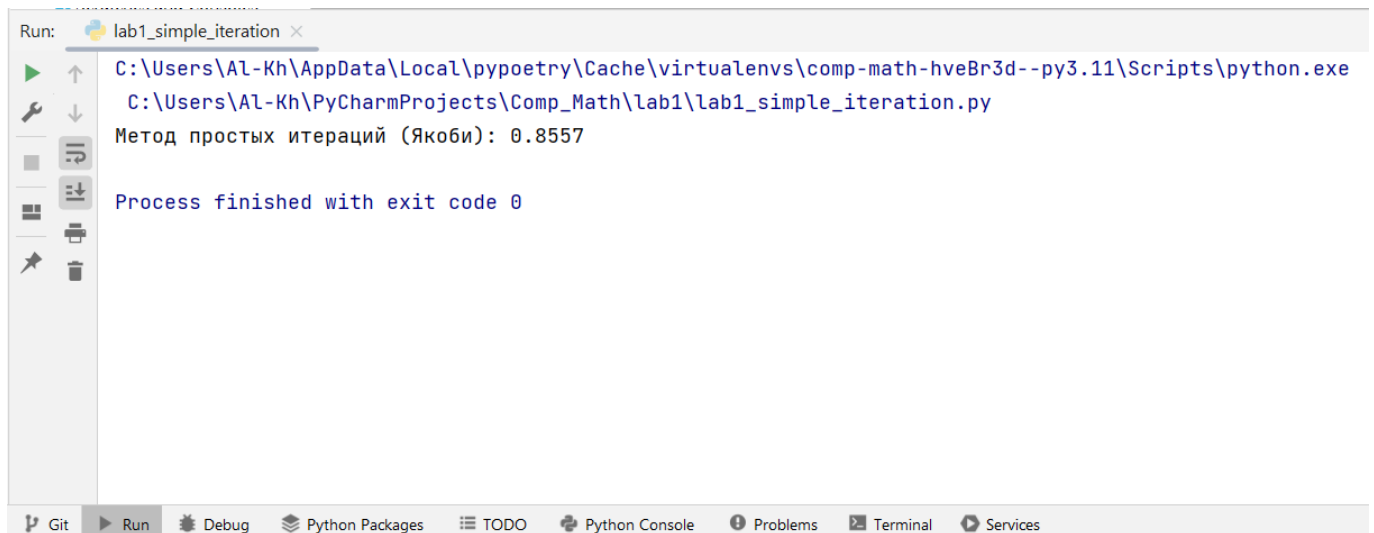
    Параметры:
        a: Начало отрезка.
        epsilon: Точность решения.

    Возвращаемое значение:
        Приближенное значение корня уравнения  $f(x) = 0$ .
    """
    g = lambda x: (1.2 - 0.2 * x ** 2 - 0.5 * x) ** (1 / 3)
    if any(derivative(g, x0=point) >= 1 for point in [a, b]):
        raise ValueError("Условие сходимости не выполнено на заданном отрезке")
    xi = g(a)
    while abs(func(xi)) >= epsilon:
        xi = g(xi)
    return xi

def main():
    # Использование методов
    a, b = step_method(func=f, start=0.8, step=0.01)
    root_jacobi = simple_iteration_method(f, a, b, epsilon=0.001)
    print(f"Метод простых итераций (Якоби): {root_jacobi:.4f}")

if __name__ == "__main__":
    main()
```

Вывод программы:



The screenshot shows the Run console in PyCharm. The title bar indicates the session is for 'lab1_simple_iteration'. The console output shows the execution of a Python script using the virtual environment's interpreter. The script's path is 'C:\Users\Al-Kh\PyCharmProjects\Comp_Math\lab1\lab1_simple_iteration.py'. The output of the script is 'Метод простых итераций (Якоби): 0.8557', followed by 'Process finished with exit code 0'. The bottom toolbar shows various IDE tools like Git, Run, Debug, Python Packages, TODO, Python Console, Problems, Terminal, and Services.

```
Run: lab1_simple_iteration ×  
C:\Users\Al-Kh\AppData\Local\pypoetry\Cache\virtualenvs\comp-math-hveBr3d--py3.11\Scripts\python.exe  
C:\Users\Al-Kh\PyCharmProjects\Comp_Math\lab1\lab1_simple_iteration.py  
Метод простых итераций (Якоби): 0.8557  
Process finished with exit code 0
```

Достоинство метода: простота алгоритма.

Недостатки: возможные сложности с выбором функции $\phi(x)$; более медленное достижение заданной точности, чем у других методов уточнения.

Вывод:

В данной лабораторной работе были изучены следующие методы численного решения нелинейных уравнений:

- шаговый метод
- метод половинного деления
- метод Ньютона
- метод простых итераций (Якоби)

Также были проведены программные вычисления с помощью MS Excel и языка программирования Python для уравнения, полученного согласно варианту.

Все решения сходятся на заданной точности. В процессе решения были проанализированы особенности, плюсы и минусы каждого из использованных методов.