Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа

Решение нелинейных уравнений

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Общие функции для ЛР на Python **1**

2 Метод бисекции (2.1.16) **2**
2.1 Формулировка задачи . . . 2
2.2 Аналитический расчет корня функции . . . 2
2.3 Код на Python . . . 3
2.4 Результат работы программы . . . 4
2.5 Графический метод, объяснение результата . . . 4

3 Метод Ньютона для кратного корня **4**
3.1 Формулировка задачи . . . 4
3.2 Код на Python . . . 5
3.3 Результат работы программы . . . 5

4 Неявно заданная функция **5**
4.1 Формулировка задачи . . . 5
4.2 Код на Python . . . 6
4.3 Результат работы программы . . . 6
4.4 График функции . . . 7

1 Общие функции для ЛР на Python

Здесь реализованы метод Ньютона, метод Ньютона для кратного корня, численная производная по одной переменной и метод бисекции для решения нелинейных уравнений, которые используются в программах ниже.

```
import typing as tp


def derivative(func: tp.Callable, point: float) -> float:
    return (func(point + 1e-10) - func(point - 1e-10)) / 2e-10


def multiplicity_newton(func: tp.Callable,
                        interval: tuple[float, float],
                        m: int = 1,
                        eps: float = 1e-5) -> tuple[float, int]:
    x, _ = interval
    cnt = 0
    while abs(func(x)) > eps:
        x -= m * func(x) / derivative(func, x)
        cnt += 1
    return x, cnt


def newton(func: tp.Callable,
           interval: tuple[float, float],
           eps: float = 1e-5) -> tuple[float, int]:
    x_min, x_max = interval
```

```
    x = x_max
    cnt = 0
    while abs(func(x)) > eps:
        x -= func(x) / derivative(func, x)
        cnt += 1
    return x, cnt


def bisection(f: tp.Callable,
              eps: float,
              interval: tuple[float, float]) -> float:
    a, b = interval
    while abs(a - b) > 2 * eps:
        x = (a + b) / 2
        a_val, x_val = f(a), f(x)
        if a_val * x_val <= 0:
            b = x
        else:
            a = x
    return (a + b) / 2
```

2 Метод бисекции (2.1.16)

2.1 Формулировка задачи

Даны две функции, надо найти корни с заданной точностью $\epsilon = 10^{-10}$ при помощи:

1. графического метода
2. готового численного метода
3. самостоятельно реализованного метода бисекции

Также необходимо объяснить полученные результаты для функции $g(x)$.

$$f(x) = sin^2(x) + \frac{5}{6}sin(x) + \frac{1}{6}$$

$$g(x) = sin^2(x) + \frac{2}{3}sin(x) + \frac{1}{9}$$

2.2 Аналитический расчет корня функции

Сначала нужно решить квадратное уравнение относительно синуса, потом получить серии корней изначального уравнения из корней квадратного. Для этого сделаем замену, а затем получим серии решений и выберем те, что попали в промежутки:

$$f(x) = sin^2(x) + \frac{5}{6}sin(x) + \frac{1}{6} = 0$$

$$sin(x) = t : t^2 + \frac{5}{6}t + \frac{1}{6} = 0$$

$$sin(x_1) = -1/2$$

$$sin(x_2) = -1/3$$

$$x_1 = Arcsin(-1/3)$$
$$x_2 = Arcsin(-1/2)$$

$$x_1 = -0.33984$$
$$x_2 = -0.52360$$

2.3 Код на Python

```
import math
import typing as tp

import scipy  # type: ignore
import numpy as np  # type: ignore
import matplotlib.pyplot as plt  # type: ignore

from utils import bisection


def root_finder(func: tp.Callable,
                name: str,
                interval: tuple[float, float]) -> None:
    print(f'{name}(x):')
    scipy_root = scipy.optimize.root(fun=func, x0=-1).x[0]
    bisect_root = bisection(func, 1e-10, interval)
    print(f'scipy root = {scipy_root:.5f}')
    print(f'bisection = {bisect_root:.5f}')

    x = np.linspace(-1, 0, 100)
    vals = np.vectorize(func)(x)
    plt.figure(figsize=(6, 4))
    plt.plot(x, vals)
    plt.axhline(y=0, color='gray', linestyle='--')
    plt.xticks(np.arange(-1, 0.1, 0.1))
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    plt.title(f'$y={name}(x)$ graph near root')
    plt.savefig(f'plots/bisec_{name}.png', dpi=300)


functions = {
    'f': lambda x: math.sin(x)**2 + 5/6 * math.sin(x) + 1/6,
    'g': lambda x: math.sin(x)**2 + 2/3 * math.sin(x) + 1/9,
}

for key, func in functions.items():
    root_finder(func, key, (-1, 0))
```

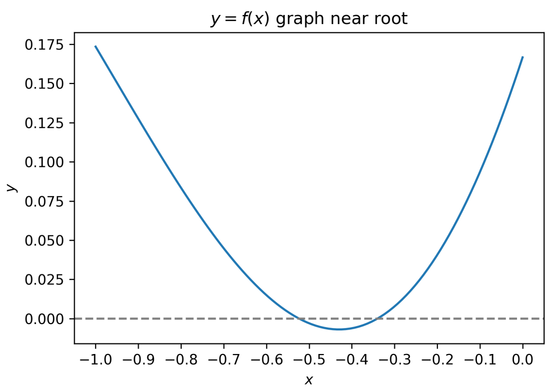
2.4 Результат работы программы

```
f(x):
scipy root = -0.52360
bisection = -0.52360
g(x):
scipy root = -0.33984
bisection = -0.00000
```

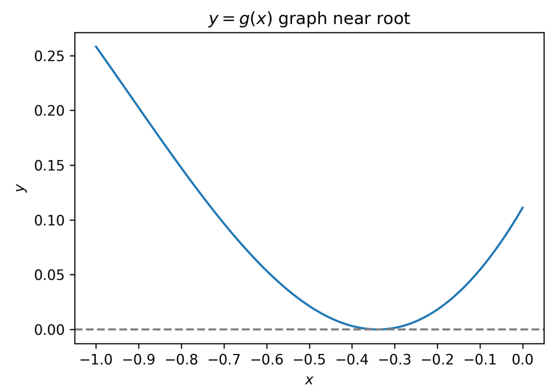
2.5 Графический метод, объяснение результата

Из графиков видно, что:

- у функции $f(x)$ два корня, один из которых действительно -0.5236, то есть расчеты обоими методами выполнены верно
- у функции $g(x)$ один корень кратности два, и метод бисекции не смог его обнаружить



(a) Функция $f(x)$



(b) Функция $g(x)$

Рис. 1: Корни функций

Так произошло из-за реализации метода, данной в задании - так как левая граница сдвигается, если центр и левая граница имеют одинаковые знаки, то она просто постоянно сдвигалась, ведь нет участка отрицательных значений, при попадании в который метод бы сдвинул правую границу. Поэтому метод просто сойдется к изначальной правой границе.

3 Метод Ньютона для кратного корня

3.1 Формулировка задачи

Дана функция, надо найти приближенно корень уравнения $f(x) = 0$ в отрезке $[a, b]$ с заданной точностью $\epsilon = 10^{-5}$ при помощи модификации метода Ньютона для кратного корня при значениях $m = (1, 2, 3, 4, 5)$, по числу итераций определить кратность корня. Описание метода Ньютона:

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$$

Функция:

$$f(x) = 32\sqrt{2}sin(x) + 8\pi + 16x^2 + \pi^2 - 32 - 8\pi x - 32x$$

$$[a, b] = [0.5, 1]$$

3.2 Код на Python

```
import math

from utils import multiplicity_newton


def ugly_func(x: float) -> float:
    return (32 * math.sqrt(2) * math.sin(x) + 8 * math.pi + 16 * x**2
            + math.pi**2 - 32 - 8 * math.pi * x - 32 * x)


print('m - root multiplicity')
print('x - root')
print('cnt - number of iterations')
for m in range(1, 6):
    x, cnt = multiplicity_newton(ugly_func, (0.5, 1), m, 1e-5)
    print(f'm = {m}: x = {x:.7f}, cnt = {cnt}')
```

3.3 Результат работы программы

```
m - root multiplicity
x - root
cnt - number of iterations
m = 1: x = 0.7739446, cnt = 8
m = 2: x = 0.7741703, cnt = 3
m = 3: x = 0.7799195, cnt = 1
m = 4: x = 0.7952828, cnt = 3
m = 5: x = 0.7739685, cnt = 8
```

По результату работы видно, что наименьшее количество итераций при значении кратности корня $m = 3$, значит корень кратности 3; значения корня отличаются из-за заданной точности для метода.

4 Неявно заданная функция

4.1 Формулировка задачи

Дана неявно заданная функция $F(x, y) = 0$, надо составить таблицу значений y по x, подставляя разные значения x с шагом $h = 0.5$ в неявно заданную функцию и решая её численно как уравнение одной переменной y с точностью $\epsilon = 10^{-7}$. Построить график функции на получившемся промежутке.

$$F(x, y) = sh(ye^y - \frac{x}{20}) + arctg(20ye^y - x) - 0.5$$
$$1 \leq x \leq 5$$

4.2 Код на Python

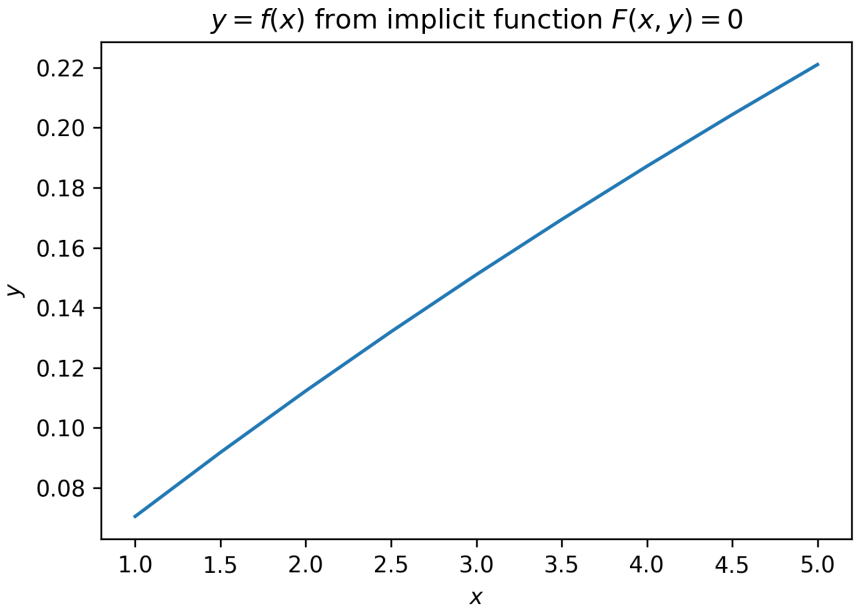
```
import math
import typing as tp

import matplotlib.pyplot as plt  # type: ignore
from scipy.optimize import root  # type: ignore


def implicit_to_normal(x_stable: float):
    def decorator(func: tp.Callable):
        def wrapper(y):
            return func(x_stable, y)
        return wrapper
    return decorator


x_interval = (1, 5)
y_interval = (0.1, 1.2)
x_vals = [x / 2 for x in range(2, 11)]
y_vals = []


def calc_point(x_val: float):

    @implicit_to_normal(x_stable=x_val)
    def implicit_function(x, y):
        return (math.sinh(y * math.exp(y) - x / 20)
                + math.atan(20 * y * math.exp(y) - x) - 0.5)

    return root(fun=implicit_function, x0=y_interval[1])


for x in x_vals:
    sol = calc_point(x)
    y_vals.append(sol.x[0])

for x, y in zip(x_vals, y_vals):
    print(f'x={x}: y={y:.7f}')

plt.figure(figsize=(6, 4))
plt.plot(x_vals, y_vals)
plt.xticks(x_vals)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title('$y=f(x)$ from implicit function $F(x, y) = 0$')
plt.savefig('plots/implicit.png', dpi=300)
```

4.3 Результат работы программы

Здесь представлена таблица значений неявно заданной функции.

```
x=1.0: y=0.0705186
x=1.5: y=0.0918375
x=2.0: y=0.1123196
x=2.5: y=0.1320346
x=3.0: y=0.1510437
x=3.5: y=0.1694006
x=4.0: y=0.1871528
x=4.5: y=0.2043427
x=5.0: y=0.2210082
```

4.4 График функции



(a) Функция $f(x)$