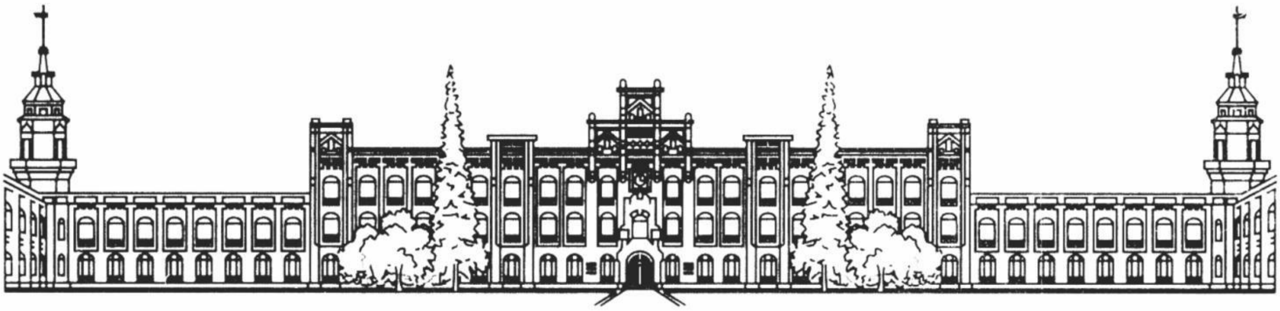


Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського» Інститут прикладного системного аналізу

Лабораторна робота №1

курсу «Чисельні методи 1»

з теми «Методи розв’язання нелінійних алгебраїчних рівнянь»

Варіант №9

Виконав студент 2 курсу групи КА-91

Косицький Вадим Вікторович

перевірила старший викладач

Хоменко Ольга Володимирівна

Київ-2021

Завдання:

1) Відокремити корені заданого рівняння, тобто для кожного з коренів визначити інтервал, до якого відповідний корінь належить та є єдиним.

2) запрограмувати методи половинного ділення, хорд та дотичних.

3) Критерієм закінчення мають бути нерівності:

А) для методу половинного ділення: $|b-a| < \varepsilon$ та $|f(x_k)| < \varepsilon$

Б) для методів хорд та дотичних $|x_k - x_{k-1}| < \varepsilon$ та $|f(x_k)| < \varepsilon$,

де $\varepsilon = 0.00001$

4) Рівняння: $-66x^7 + 73x^6 + 763x^5 + 179x^4 - 737x^3 - 406x^2 - 12x + 15 = 0$

Теоретичні відомості:

Теорема 1: Поліном n-того степеня має рівно n коренів, дійсних чи комплексних, за умови, що кожен корінь рахується стільки разів, яка його кратність.

Теорема 3: Нехай A = max $\{|a_6|, \dots, |a_0|\}$, B = max $\{|a_7|, \dots, |a_1|\}$, де

$a_k, k = 0,1, \dots, 7$ –коефіцієнти рівняння

$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$

Тоді модулі усіх коренів рівняння $x_{*i}, i = 0,1, \dots .7$ задовольняють нерівність

$$\frac{1}{1 + \frac{B}{|a_0|}} \leq |x_{*i}| \leq 1 + \frac{A}{|a_7|}$$

Теорема 7 (Больцано_Коші): Якщо функція $f(x)$, що визначає рівняння $f(x) = 0$, на кінцях відрізка $[a_i; b_i]$ приймає значення різних знаків, тобто $f(a_i) \cdot f(b_i) < 0$, то на цьому відрізку міститься принаймні один корінь рівняння.

Спосіб перебору*:* полягає в тому, що частину області визначення функції f(x) розбивають на відрізки точками xi , розташованими на умовно невеликій відстані h одна від одної. Обчисливши значення f(x) у всіх цих точках (або лише визначивши знаки f(xi), порівнюють їх в сусідніх точках, тобто, перевіряють виконання умови f(xi−1)f(xi) ≤ 0. Якщо відома кількість коренів в досліджуваній області, то, зменшуючи крок пошуку h таким чином можна їх локалізувати, або ж довести процес до стану, який дозволяє стверджувати про наявність пар коренів, які відрізняються один від одного на величину h = ε.

Опис процесу відокремлення коренів

Відокремлення інтервалів, де лежить рівно по 1 кореню:

1) Перевіряю знак коефіцієнту a_7. Оскільки він від'ємний, то дане рівняння f(x) = 0 я замінюю на еквівалентне -f(x) = 0. Цю дію я виконав для спрощення подальших розрахунків, та можливість використовувати деякі теореми.
2) Використовуючи теорему 3, знаходжу інтервал, де розміщені усі модулі коренів заданого рівняння.
3) Використавши спосіб перебору на інтервалі, знайденому в пункті 2, я виділив інтервали, де існує принаймні один корінь. Крок я зробив розміру 0.001
4) Оскільки таких інтервалів вийшло 7, а за теоремою 1 наше рівняння може мати 7 коренів 1 кратності, то відповідно я знайшов інтервали в яких розміщено рівно по 1 дійсному кореню даного рівняння.
5) Програмна частина даного процесу:

На рисунку 1 показано частину коду, де я в список **a** вписую коефіцієнти

```python
# input od data
a = [float(i) for i in input("Enter the list of koef : ").split()]
n = len(a) -1
if (a[0] < 0):
    for i in a:
        i = -i
        pass
    pass
a.reverse()
```

Рис.1

На рисунку 2 показано частину коду, де я застосовую Теорему 3

```python
# Theorem 3
inf_interval_pos = 1/(1+float(max(a[1:n+1]))/abs(a[0]))
sup_interval_pos = 1 + float(max(a[:n]))/abs(a[n])
inf_interval_neg = -sup_interval_pos
sup_interval_neg = -inf_interval_pos
```

Рис.2

На рисунку 3 показано частину коду, де я знаходжу конкретні інтервали, де лежить рівно по 1 кореню. Усі інтервали я записав в список intervals

```python
# Find intervals for each solve
intervals = []
i = inf_interval_neg
f_last = 0

while i < sup_interval_neg:
    f_cur = f(i, a)
    if (f_last * f_cur < 0):
        intervals.append([i - eps0, i])
        pass
    f_last = f_cur
    i = i+eps0
    pass

i = inf_interval_pos
f_last = 0
while i < sup_interval_pos:
    f_cur = f(i, a)
    if (f_last * f_cur < 0):
        intervals.append([i- eps0,i])
        pass
    f_last = f_cur
    i = i+eps0
    pass
```

Рис.3

В результаті роботи даної програми, я отримав такі інтервали:

[[-2.5136060606074158, -2.512606060607416], [-0.7896060606075822, -0.7886060606075822], [-0.47760606060758193, -0.4766060606075819], [-0.3476060606075818, -0.3466060606075818], [0.1592802056555271, 0.1602802056555271], [1.0922802056555176, 1.0932802056555175], [3.9792802056551997, 3.9802802056551996]]

6) Після відокремлення інтервалів існування одного кореня даного рівняння я реалізував за допомогою окремих функцій методи половинного ділення, метод хорд та дотичних.

Реалізація методу половинного ділення:

1) Задаю змінні a, b, де a – ліва границя певного інтервалу, b – права.
2) Обчислюю c = (a+b)/2.
3) Обчислюю $f(c)$
4) Якщо або b-a < ε або $|f(c)| < \varepsilon$, то до списку розв'язків додаю c, та переходжу до пункту 1, для наступного інтервалу розв'язків.
5) Якщо $f(a)f(c) < 0$, то значення b змінюю на значення c та повертаюсь до пункту 2, інакше значення a змінюю на значення c та повертаюсь до пункту 2

Реалізація методу хорд:

1) Задаю змінні a, b, де a – ліва границя певного інтервалу, b – права.
2) Якщо друга похідна в точці x, що належить даному інтервалу менша за 0 то замінюю нашу функцію f на $-f$
3) Якщо $f(a) > 0$, то в змінну b = b – $\frac{f(b)}{f(b)-f(a)} * (b-a)$ і якщо |b-b_{old}| < ε або $|f(b)| < \varepsilon$, то до списку розв'язків додаю b, та переходжу до пункту 1, для наступного інтервалу розв'язків. А якщо $f(b) > 0$, то в змінну a = a –

$\frac{f(a)}{f(b)-f(a)} * (b-a)$ і якщо |a-a_{old}| $< \varepsilon$ або $|f(a)| < \varepsilon$, то до списку розв'язків додаю a, та переходжу до пункту 1, для наступного інтервалу розв'язків.

Реалізація методу Ньютона:

1) Задаю змінні a, b, c, де a – ліва границя певного інтервалу, b – права, а c = a
2) Поки $f(c) * f''(c) < 0$, c = c + 0.01, інакше переходимо до наступного пункту.
3) Поки |c-c_{old}| $> \varepsilon$ і $|f(c)| > \varepsilon$ то в змінну c = c - $\frac{f(c)}{f'(c)}$, інакше до списку розв'язків додаю c, та переходжу до пункту 1, для наступного інтервалу розв'язків.

Лістинг програми

```
import math
import matplotlib.pyplot as plt
import numpy as np
eps0 = 0.001        # step when finding intervals
eps = 0.00001

def f(x, list):
    n = len(list)
    s = 0
    for i in range(n):
        s = s + list[i] * x**(i)
        pass
    return s

def division_in_half(intervals, eps, delta, koef):
    solution = []
    indicator = 0
    for interval in intervals:
        a = interval[0]
```

```
        b = interval[1]
        while 1:
            c = (a+b)/2.0
            if b-a < 2*eps :
                solution.append(c)
                indicator +=1
                break
            f_c = f(c, koef)
            if (abs(f_c) < delta):
                solution.append(c)
                indicator += 1
                break
            if(indicator == 0 ):
                print(f"interval [{a, b}] , f(left) = {f(a, koef)}, f(right) = {f(b, koef)} ")
            if(f(a, koef)*f_c < 0):
                b = c
            else:
                a = c
    return solution

def derivative_of_polinom(x, list, k):
    lis = list.copy()

    for el in range(k):
        new_koef = []
        n = len(lis)
        for i in range(n-1):
            new_koef.append(lis[i+1]*(i+1))
            pass
        lis.clear()
        lis.extend(new_koef)
    s = 0
    for i in range(len(lis)):
        s = s + lis[i]*x**(i)
    return s
```

```
def hord_method (intervals, eps, delta, koef_old):
    koef = koef_old.copy()
    solution = []
    indicator = 0
    for interval in intervals:
        a = interval[0]
        b = interval[1]
        if(derivative_of_polinom((b+a)/2, koef, 2) < 0 ):
            for i in koef:
                i = -i

        if (f(a, koef) > 0):
            while 1:
                b_new = b - f(b, koef)*(b-a)/(f(b, koef)-f(a, koef))
                if(abs(b_new - b)<eps or abs(f(b_new, koef))<delta):
                    solution.append(b_new)
                    indicator +=1
                    break
                b = b_new
                if (indicator == 0):
                    print(f"interval [{a, b}] , f(left) = {f(a, koef)}, f(right) = {f(b, koef)} ")
        else:
            while 1:
                a_new = a - f(a, koef) * (b - a) / (f(b, koef) - f(a, koef))
                if (abs(a_new - a) < eps or abs(f(a_new, koef)) < delta):
                    solution.append(a_new)
                    indicator += 1
                    break
                a = a_new
                if (indicator == 0):
                    print(f"interval [{a, b}] , f(left) = {f(a, koef)}, f(right) = {f(b, koef)} ")

    return solution
```

```
def newton_method(intervals, eps, delta, koef):
    solution = []
    indicator = 0
    for interval in intervals:
        a = interval[0]
        b = interval[1]
        c = a
        while 1:
            if (f(c, koef) * derivative_of_polinom(c, koef, 2) > 0):
                break
            c = c + eps0
            pass
        while 1:
            c_new = c - f(c, koef)/derivative_of_polinom(c, koef, 1)
            if (abs(c_new - c) < eps or abs(f(c_new, koef)) < delta):
                solution.append(c_new)
                indicator +=1
                break
            c = c_new
            if (indicator == 0):
                print(f"interval [{c, b}] , f(left) = {f(c, koef)}, f(right) = {f(b, koef)} ")
    return solution
# input od data
a = [float(i) for i in input("Enter the list of koef : ").split()]
n = len(a) -1
if (a[0] < 0):
    for i in a:
        i = -i
        pass
    pass
a.reverse()
# Theorem 3
inf_interval_pos = 1/(1+float(max(a[1:n+1]))/abs(a[0]))
sup_interval_pos = 1 + float(max(a[:n]))/abs(a[n])
inf_interval_neg = -sup_interval_pos
```

```
sup_interval_neg = -inf_interval_pos

# Find intervals for each solve
intervals = []
i = inf_interval_neg
f_last = 0
while i < sup_interval_neg:
    f_cur = f(i, a)
    if (f_last * f_cur < 0):
        intervals.append([i - eps0, i])
        pass
    f_last = f_cur
    i = i+eps0
    pass
i = inf_interval_pos
f_last = 0
while i < sup_interval_pos:
    f_cur = f(i, a)
    if (f_last * f_cur < 0):
        intervals.append([i- eps0,i])
        pass
    f_last = f_cur
    i = i+eps0
    pass

# print solution
print(intervals)
print("solution 1:", division_in_half(intervals, eps, eps, a))
print("solution 2:", hord_method(intervals, eps, eps, a))
print("solution 3:", newton_method(intervals, eps, eps, a))
# Draw plot
x = np.linspace(-3, 5, 10000)
y = f(x,a)
fig, ax = plt.subplots()
ax.plot(x, y, color="blue", label="f(x)")
```

```
plt.plot([-3,5],[0,0], color='g')
ax.legend()
plt.axis([-3, 5, -10, 30])
plt.show()
```

Приклад роботи програми:

1)

Enter the list of koef : -66 73 763 179 -737 -406 -12 15

[[-2.5136060606074158, -2.512606060607416], [-0.7896060606075822, -0.7886060606075822], [-0.47760606060758193, -0.4766060606075819], [-0.3476060606075818, -0.3466060606075818], [0.1592802056555271, 0.1602802056555271], [1.0922802056555176, 1.0932802056555175], [3.9792802056551997, 3.9802802056551996]]

У цій частині було виведено усі знайдені інтервали, в кожному з яких лежить рівно по одному кореню рівняння.

2)

interval [(-2.5136060606074158, -2.512606060607416)] , f(left) = 24.133981369457615, f(right) = -7.286792795988731

interval [(-2.5131060606074156, -2.512606060607416)] , f(left) = 8.405233227385907, f(right) = -7.286792795988731

interval [(-2.5128560606074157, -2.512606060607416)] , f(left) = 0.5546332158701262, f(right) = -7.286792795988731

interval [(-2.5128560606074157, -2.5127310606074156)] , f(left) = 0.5546332158701262, f(right) = -3.3672261320753023

interval [(-2.5128560606074157, -2.512793560607416)] , f(left) = 0.5546332158701262, f(right) = -1.406583094583766

interval [(-2.5128560606074157, -2.512824810607416)] , f(left) = 0.5546332158701262, f(right) = -0.42604660484357737

solution 1: [-2.512832623107416, -0.7887076231075822, -0.47737949810758196, -0.3468169981075818, 0.1593036431555271, 1.0923817681555177, 3.9801161431551995]

У цій частині показано результат роботи методу половинного ділення для першого інтервалу. Бачимо, що було виконано 6 ітерацій.

3)

interval [(-2.5136060606074158, -2.5128379706411286)] , f(left) = 24.133981369457615, f(right) = -0.013079131997074 solution 2: [-2.512838386673136, -0.7887068472996598, -0.47738295434329814, -0.3468170745694241, 0.1593041839785655, 1.0923886823730309, 3.9801130020819695]

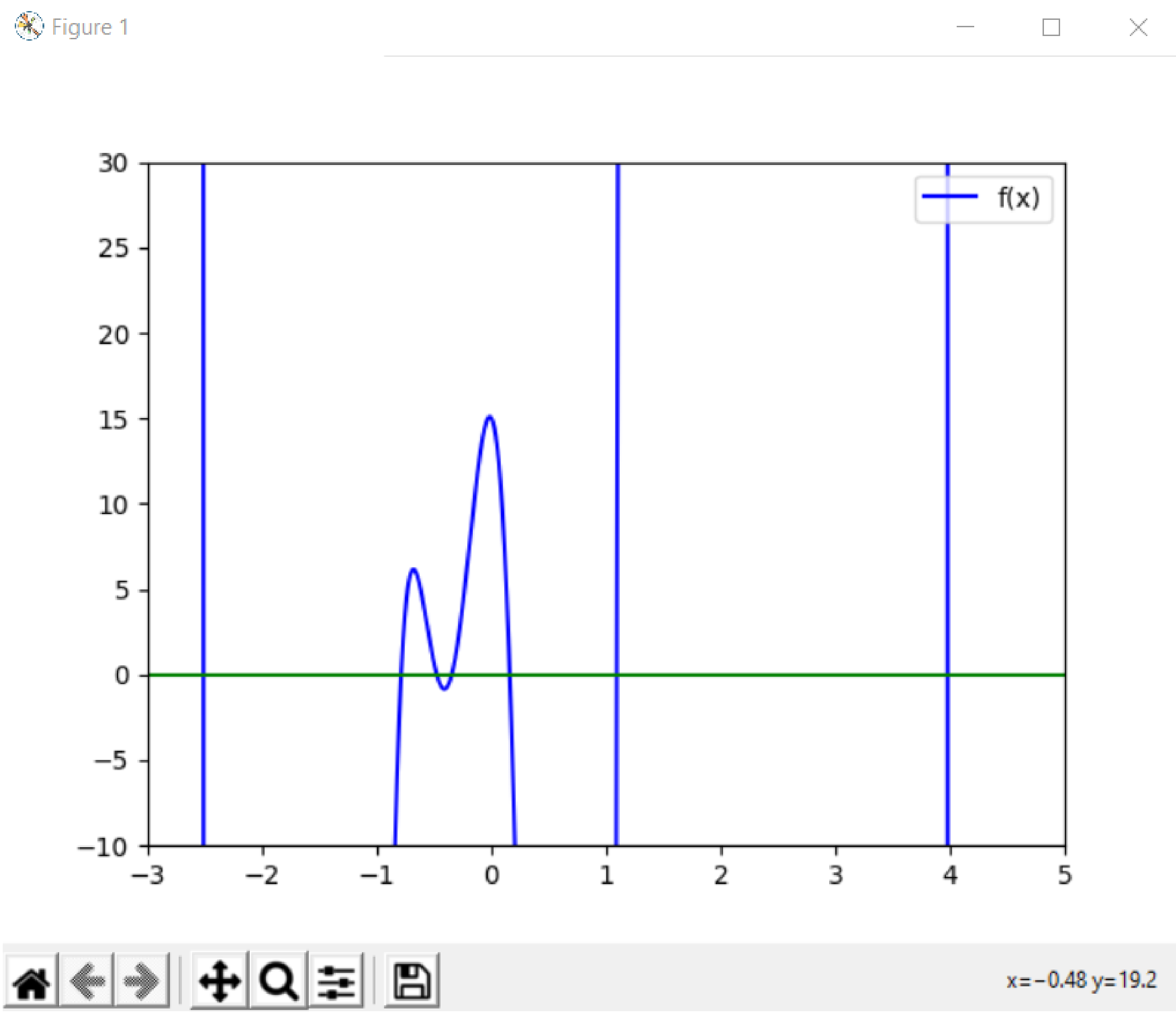
У цій частині показано результат роботи методу хорд для першого інтервалу. Бачимо, що було виконано 1 ітерацію.

4)

interval [(-2.5128397626678, -2.512606060607416)] , f(left) = 0.04315739593585022, f(right) = -7.286792795988731

solution 3: [-2.5128383874249938, -0.7887068470846572, -0.4773832726225109, -0.3468167406223343, 0.159304184014468, 1.092388683093923, 3.980113001845474]

У цій частині показано результат роботи методу Ньютона для першого інтервалу. Бачимо, що було виконано 1 ітерацію.



Висновок:

Виконуючи цю лабораторну роботу я навчився застосовувати чисельні методи розв'язання нелінійних алгебраїчних рівнянь, відокремлювати та уточнювати корені нелінійних рівнянь.

Результатами виконання даних методів:

1) Метод половинного ділення:

[-2.512832623107416, -0.7887076231075822, -0.47737949810758196, -0.3468169981075818, 0.1593036431555271, 1.0923817681555177, 3.9801161431551995]

2) Метод хорд:

[-2.512838386673136, -0.7887068472996598, -0.47738295434329814, -0.3468170745694241, 0.1593041839785655, 1.0923886823730309, 3.9801130020819695]

3) Метод Ньютона:

[-2.5128383874249938, -0.7887068470846572, -0.4773832726225109, -0.3468167406223343, 0.159304184014468, 1.092388683093923, 3.980113001845474]

Тепер якщо дати відповідь з точністю 0.00001 маємо:

[-2.51283, -0.78870, -0.47738, -0.34681, 0.15930, 1.09238, 3.98011]

Видно, що найшвидшим за кількістю ітерацій виявилися методи хорд та Ньютона (вони збігаються швидше). Проте, вони потребують додаткових перевірок, що збільшує тривалість обчислень та обсяг коду. При цьому, метод половинного ділення є більш універсальним та може підійти під усі задач.

Отже, кожен із методів є і справді дієвим, але жоден з них не є і універсальним і швидкодієвим. Кожен з них підходить під свою задачу, та вибір: який із способів застосувати, залежить і від відведеного часу для виконання програмою розрахунків, і від складності функції.