Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра обчислювальної техніки

Лабораторна робота № 6

з дисципліни «Чисельні методи»

на тему

“Розв’язання нелінійних рівнянь”

Виконав:
студент гр. ІП-93
Завальнюк Максим

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

Виконати допрограмовий етап: визначити кількість дійсних коренів рівняння, відокремити корені рівняння (письмово). Встановити проміжки.

Виконати програмний етап: уточнити корені рівняння: методом бісекції, методом хорд, методом Ньютона (дотичних).

Порівняти отримані результати, зробити висновки, який метод приводить до меншої кількості ітерацій і чим це зумовлено.

2 Розв'язок

Допрограмовий етап

Рівняння: 10 * x^5 -3 * x^4 + 7 * x^2 -27 = 0 (1)

Спираючись на наслідок з **теореми 1** (про число коренів алгебраїчного рівняння) та **теореми 2** (про властивість парної спряженості комплексних коренів рівняння), можемо зробити висновок, про існування хоча б одного дійсного кореня у нашого рівняння (1), оскільки воно має степінь 5.

За допомогою **теореми 3** (про оцінку модулів коренів рівняння) та наслідку з неї, визначемо верхні та нижні межі додатних та від'ємних коренів.

A = 27, B = 10, $|a_0|$ = 27, $|a_n|$ = 10

r = $\frac{1}{1+\frac{B}{|a_0|}} = \frac{1}{1+\frac{10}{27}}$ = 0.729

R = 1 + $\frac{A}{|a_{10}|}$ = 1 + $\frac{27}{|10|}$ = 3.7

0.729 < $|x_i^*|$ < 3.7, i = 1, 2, …, n

Розкриваємо модуль та отримуємо межі:

0.729 < x_i^{*+} < 3.7, i = 1, 2, …, n
-3.7 < x_i^{*-} < -0.729, i = 1, 2, …, n

Визначимо точніші межі дійсних коренів для рівняння, використовуючи **теорему 4** (теорема Лагранжа про верхню межу додатних коренів рівняння) та **теорему 5** (про нижню і верхню межі додатних та від'ємних коренів алгебраїчного рівняння).

i = 4, C = 27

Тоді верхня межа додатних коренів:

R = 1 + $\sqrt[n-i]{\frac{C}{a_n}}$ = 1 + $\sqrt[5-4]{\frac{27}{10}}$ = 3.7

Далі використаємо **теорему 5**:

Нехай R — верхня межа додатних коренів рівняння $P_n(x)=0$,

R_1 — верхня межа додатних коренів рівняння $P^1(x)=x^n P_n(\frac{1}{x})=0$,

R_2 — верхня межа додатних коренів рівняння $P^2(x)=P_n(-x)=0$,

R_3 — верхня межа додатних коренів рівняння $P^3(x)=x^n P_n(-\frac{1}{x})=0$.

Тоді додатні корені x_i^{*+} та від'ємні корені x_i^{*-} рівняння (1) задовольняють нерівності

$$\frac{1}{R_1} \le x_i^{*+} \le R; \qquad -R_2 \le x_i^{*-} \le -\frac{1}{R_3}. \tag{4}$$

Побудуємо рівняння $P^1(x) = x^n P_n(\frac{1}{x}) = 0$:

$x^5(10(\frac{1}{x})^5 - 3\left(\frac{1}{x}\right)^4 + 7\left(\frac{1}{x}\right)^2 - 27) = 0$, 10 -3x + $7x^3$ - $27x^5$ = 0 або $27x^5$ - $7x^3$ +3x -10 = 0

a_n = 27, i = 3, C = 10

R_1 = 1 + $\sqrt[n-i]{\frac{C}{a_n}}$ = 1 + $\sqrt[5-3]{\frac{10}{27}}$ = 1.60858

Побудуємо рівняння $P^2(x) = P_n$(-x) = 0:

$10(-x)^5$ – 3 $(-x)^4 + 7$ $(-x)^2$ – 27 = 0 або -10x^5 – 3 x^4 + 7 x^2 – 27 = 0 або 10x^5 + 3x^4 - 7x^2 + 27 = 0

a_n = 10, i = 2, C = 7

R_2 = 1 + $\sqrt[n-i]{\frac{C}{a_n}}$ = 1 + $\sqrt[5-2]{\frac{7}{10}}$ = 1.8879

Побудуємо рівняння $P^3(x) = x^n P_n(-\frac{1}{x}) = 0$:

$x^5(10(-\frac{1}{x})^5 - 3\left(-\frac{1}{x}\right)^4 + 7\left(-\frac{1}{x}\right)^2 - 27)$ = 0, -10 -3x + $7x^3$ - $27x^5$ = 0 або $27x^5$ - $7x^3$ +3x +10 = 0

a_n = 27, i = 3, C = 7

R_3 = 1 + $\sqrt[n-i]{\frac{C}{a_n}}$ = 1 + $\sqrt[5-3]{\frac{7}{27}}$ = 1.50918

Межі для додатних коренів:

$\frac{1}{R_1} \le x_i^{*+} \le$ R, $\frac{1}{1.60858} \le x_i^{*+} \le 3.7$, 0.621666 $\le x_i^{*+} \le 3.7$

Межі для від'ємних коренів:

-R_2 $\le x_i^{*-} \le$ -$\frac{1}{R_3}$, -1.8879 $\le x_i^{*-} \le$ -0.662611

За допомогою **теореми 6** (теорема Декарта про кількість дійсних коренів алгебраїчних рівнянь) визначаємо кількість дійсних додатних та від'ємних коренів.

Число знакозмін у рівнянні $P_n(x)$ = 0 рівне S_1 = 3, тобто кількість додатних коренів рівняння рівна 3-м або менше від нього на парне число, тобто рівна 1.

Число знакозмін у рівнянні $P_n(-x)$ = 0 рівне S_2 = 2, тобто кількість від'ємних коренів рівняння рівна 2-м або менше від нього на парне число, тобто рівна 0, що означає, що від'ємних дійсних коренів не має.

Використаємо **теорему 7** (теорема Гюа про необхідну умову дійсності всіх коренів алгебраїчного рівняння) Взявши за k = 1, отримаємо:

9 > 10 * 0. Але Теорема Гюа є лише необхідною умовою дійсності всіх коренів алгебраїчного рівняння.

Виконаємо відокремлення коренів для f(x) = 10 * x^5 -3 * x^4 + 7 * x^2 -27 = 0 методом Штурма

Побудуємо ряд Штурма
f_0(x) = f(x) = 10 * x^5 - 3 * x^4 + 7 * x^2 -27
f_1(x) = f'(x) = 50 * x^4 - 12 * x^3 + 14 * x
Виконаємо ділення f_0 на f_1та знайдемо таким чином f_2

$$
\begin{array}{llllll|l}
10x^5 & -3x^4 & +0x^3 & +7x^2 & +0x & -27 & 50x^4 \; -12x^3 \; +0x^2 \; +14x \\
\cline{7-7}
10x^5 & -2{,}4x^4 & +0x^3 & +2{,}8x^2 & & & 0{,}2x - \frac{3}{250} \\
\cline{1-6}
 & -0{,}6x^4 & +0x^3 & +4{,}2x^2 & +0x & -27 & \\
 & -0{,}6x^4 & +\frac{18}{125}x^3 & +0x^2 & -\frac{21}{125}x & & \\
\cline{2-6}
 & & -\frac{18}{125}x^3 & +4{,}2x^2 & +\frac{21}{125}x & -27 &
\end{array}
$$

Помножимо залишок на 125/3, та візьмемо з протилежним знаком отримаємо
f_2(x) = 6x^3 - 175x^2 – 7x + 1125
Виконаємо ділення f_1 на f_2 та знайдемо таким чином f_3

$$
\begin{array}{lllll|l}
50x^4 & -12x^3 & +0x^2 & +14x & +0 & 6x^3 \; -175x^2 \; -175x \; +112 \\
\cline{6-6}
50x^4 & -1458\frac{1}{3}x^3 & -1458\frac{1}{3}x^2 & +9375x & & 8\frac{1}{3}x \; +241\frac{1}{18} \\
\cline{1-5}
 & 1446\frac{1}{3}x^3 & +1458\frac{1}{3}x^2 & -9361x & +0 & \\
 & 1446\frac{1}{3}x^3 & -42184\frac{13}{18}x^2 & -42184\frac{13}{18}x & +271187{,}5 & \\
\cline{2-5}
 & & 43643\frac{1}{18}x^2 & +32823\frac{13}{18}x & -271187{,}5 &
\end{array}
$$

Помножимо залишок на 18/125 та візьмемо з протилежним знаком, отримаємо
f_3(x) = -6083x^2 + 1105x + 39051
Таким же чином f_4 = 1255x – 146478 та f_5 = 1

Ряд Штурма

f_0(x) = 10 * x^5 - 3 * x^4 + 7 * x^2 -27
f_1(x) = 50 * x^4 - 12 * x^3 + 14 * x
f_2(x) = 6x^3 - 175x^2 – 7x + 1125
f_3(x) = -6083x^2 + 1105x + 39051
f_4(x) = 1255x – 146478
f_5 = 1

Визначимо знаки цих многочленів при x = $-\infty$ та при x = $+\infty$

x	f_0(x)	f_1(x)	f_2(x)	f_3(x)	f_4(x)	f_5(x)	К-сть знакозмін
$+\infty$	+	+	+	-	+	+	2
$-\infty$	-	+	-	-	-	+	3

Висновок: многочлен має рівно 3 – 2 = 1 дійсний корінь.

Локалізуємо корені. Спираючись на попередні дані, можемо зробити висновок, що многочлен має один дійсний додатній корінь і належить проміжку
$0.621666 \leq x_i^{*+} \leq 3.7$
Продовжимо локалізувати корені

x	f_0(x)	f_1(x)	f_2(x)	f_3(x)	f_4(x)	f_5(x)	К-сть знакозмін
0.7	-	+	+	+	-	+	3
0.9	-	+	+	+	-	+	3
1.1	-	+	+	+	-	+	3
1.3	+	+	+	+	-	+	2

Поліном **10 * x^5 - 3 * x^4 + 7 * x^2 -27** має лише один дійсний корінь, котрий належить проміжку **[1.1, 1.3]**.

Програмовий етап

```
########## Start polynomial ##########
27.0 + 0.0 x**1 + 7.0 x**2 + 0.0 x**3 - 3.0 x**4 + 10.0 x**5
########## Bisection method ##########
Answers: [1.182041168212891], iterations: 18
########## Fault for bisection method ##########
0.0
########## Newton method ##########
Answers: [1.1820411802794375], iterations: 4
########## Fault for newton method ##########
0.0
########## Chords method ##########
Answers: [1.1820411704925782], iterations: 9
########## Fault for chords method ##########
0.0
```

З кількості ітерацій та застосування методів можемо зробити висновки:

- До переваг методу бісекцій можна віднести його високу надійність і простоту. Недоліком методу є той факт, що перед тим, як його використовувати, необхідно знайти дві точки, значення функції в яких мають різні знаки. Порядок збіжності методу лінійний, на кожному кроці точність зростає удвічі.
- Збіжність методу Ньютона квадратична. Таким чином, збіжність методу дотичних Ньютона дуже швидка. Недоліком методу є необхідність обчислення похідних на кожному кроці.
- Щоб уникнути обчислення похідної, метод Ньютона можна спростити, замінивши похідну на наближене значення. Таке використовується у методі хорд.

З цього робимо заключний висновок, що методи по швидкості розташовуються так: метод Ньютона, метод хорд, метод бісекцій.

3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

```
np_roots = np.roots([10, -3, 0, 7, 0, -27])
np_roots = np_roots[np.isreal(np_roots)]
```

Результат:

```
########## All roots from NumPy ##########
[ 1.18204118+0.j          0.52118821+1.20449926j  0.52118821-1.20449926j
 -0.9622088 +0.63267299j -0.9622088 -0.63267299j]
########## Real roots from NumPy ##########
[1.18204118+0.j]
```

4 Лістинг програми

```
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
from string import Template


template = Template('#' * 10 + ' $string ' + '#' * 10)
epsilon = 10 ** (-6)


def y(x):
    return 10 * x ** 5 - 3 * x ** 4 + 7 * x ** 2 - 27


def y_derivative(x):
    return 50 * x ** 4 - 12 * x ** 3 + 14 * x


def y1(x):
    return 10 * x ** 5 - 3 * x ** 4


def y2(x):
    return 27 - 7 * x ** 2


intervals = [
    [1.1, 1.3]
]


class Polynomial:
    def __init__(self, epsilon:float, intervals:list):
        self.epsilon = epsilon
        self.intervals = intervals

    @classmethod
    def printPolynomial(cls) -> None:
        """
        Print polynomial as NumPy object, normal form
        :return: nothing to return
        """
        print(template.substitute(string='Start polynomial'))
        polynom = np.polynomial.Polynomial([27, 0, 7, 0, -3, 10])
        print(polynom)

    def bisectionMethod(self) -> None:
        """
        Implementation of bisection method
        :return: nothing to return
        """
        answers = []
        iterations = 0
        for interval in self.intervals:
            root = 1000
            a, b = interval[0], interval[1]
            while abs(b - a) > self.epsilon and abs(y(root)) > self.epsilon:
                root = (a + b) / 2
                if y(root) * y(a) <= 0:
                    a, b = a, root
                elif y(root) * y(b) <= 0:
                    a, b = root, b
                iterations += 1
            answers.append(root)
```

```python
        print(template.substitute(string='Bisection method'))
        print(f'Answers: {answers}, iterations: {iterations}')
        self.get_faults(answers, [scipy.optimize.bisect(y, self.intervals[0][0],
self.intervals[0][1])], 'bisection')

    def newtonMethod(self) -> None:
        """
        Implementation of Newton method
        :return: nothing to return
        """
        answers = []
        iterations = 0
        for interval in self.intervals:
            start_x = 0
            a, b = interval[0], interval[1]
            if y(a) * y_derivative(a) > 0:
                start_x = a
            else:
                start_x = b
            root = start_x - y(start_x) / y_derivative(start_x)
            iterations += 1
            while abs(y(root)) > self.epsilon:
                root = root - y(root) / y_derivative(root)
                iterations += 1
            answers.append(root)
        print(template.substitute(string='Newton method'))
        print(f'Answers: {answers}, iterations: {iterations}')
        self.get_faults(answers, [scipy.optimize.newton(y, intervals[0][0])],
'newton')

    def chordsMethod(self) -> None:
        """
        Implementation of chords method
        :return: nothing to return
        """
        answers = []
        iterations = 0
        for interval in self.intervals:
            root = 1000
            a, b = interval[0], interval[1]
            while abs(b - a) > self.epsilon and abs(y(root)) > self.epsilon:
                root = (a * y(b) - b * y(a)) / (y(b) - y(a))
                if y(root) * y(a) <= 0:
                    a, b = a, root
                elif y(root) * y(b) <= 0:
                    a, b = root, b
                iterations += 1
            answers.append(root)
        print(template.substitute(string='Chords method'))
        print(f'Answers: {answers}, iterations: {iterations}')
        self.get_faults(answers, [scipy.optimize.bisect(y, self.intervals[0][0],
self.intervals[0][1])], 'chords')

    def get_faults(self, self_values: list, true_values: list, method: str) ->
None:
        """
        Getting faults for all methods
        :param self_values: roots from my methods
        :param true_values: roots from NumPy
        :param method: string for printing
        :return:
        """
        print(template.substitute(string=f'Fault for {method} method'))
        fault = 0
        for index in range(len(self_values)):
            fault = abs(true_values[index] - self_values[index])
```

```
        print(round(fault, 6))


np_roots = np.roots([10, -3, 0, 7, 0, -27])
print(template.substitute(string='All roots from NumPy'))
print(np_roots)
print(template.substitute(string='Real roots from NumPy'))
np_roots = np_roots[np.isreal(np_roots)]
print(np_roots)

polynomial = Polynomial(epsilon, intervals.copy())
polynomial.printPolynomial()
polynomial.bisectionMethod()
polynomial.newtonMethod()
polynomial.chordsMethod()

# Showing plot with functions
x_axis = np.linspace(-2, 2, num=1000)
x_axis_2 = np.linspace(-5, 5, num=1000)
fig, ax = plt.subplots()
ax.plot(x_axis, y(x_axis), label='y')
ax.plot(x_axis, y1(x_axis), label='y1')
ax.plot(x_axis_2, y2(x_axis_2), label='y2')

ax.legend(loc='lower left', ncol=2)
plt.grid()
plt.show()
```