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

[Зміст 2](#_Toc70277394)

[1 Постановка задачі 3](#_Toc70277395)

[2 Розв’язок 4](#_Toc70277396)

[Допрограмовий етап 4](#_Toc70277397)

[Програмовий етап 7](#_Toc70277398)

[3 Розв’язок за допомогою NumPy 8](#_Toc70277399)

[4 Лістинг програми 9](#_Toc70277400)

### 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 \* -3 \* + 7 \* -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, = 27, = 10

r = = = 0.729

R = 1 + = 1 + = 3.7

0.729 < < 3.7, i = 1, 2, …, n

Розкриваємо модуль та отримуємо межі:

0.729 < < 3.7, i = 1, 2, …, n

-3.7 < < -0.729, i = 1, 2, …, n

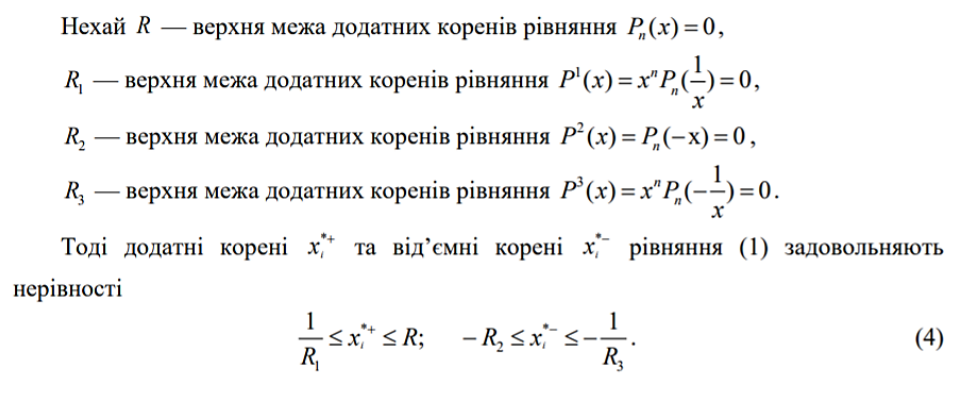
Визначимо точніші межі дійсних коренів для рівняння, використовуючи **теорему 4** (теорема Лагранжа про верхню межу додатних коренів рівняння) та **теорему 5** (про нижню і верхню межі додатних та від’ємних коренів алгебраїчного рівняння).

i = 4, C = 27

Тоді верхня межа додатних коренів:

R = 1 + = 1 + = 3.7

Далі використаємо **теорему 5**:

Побудуємо рівняння = () = 0:

(10) = 0, 10 -3x + 7 - 27 = 0 або 27 - 7 +3x -10 = 0

= 27, i = 3, C = 10

= 1 + = 1 + = 1.60858

Побудуємо рівняння = (-x) = 0:

10 – 3 + 7 – 27 = 0 або -10 – 3 + 7 – 27 = 0 або 10 + 3 - 7 + 27 = 0

= 10, i = 2, C = 7

= 1 + = 1 + = 1.8879

Побудуємо рівняння = (-) = 0:

(10) = 0, -10 -3x + 7 - 27 = 0 або 27 - 7 +3x +10 = 0

= 27, i = 3, C = 7

= 1 + = 1 + = 1.50918

Межі для додатних коренів:

≤ ≤ R, ≤ ≤ 3.7, 0.621666 ≤ ≤ 3.7

Межі для від’ємних коренів:

- ≤ ≤ -, -1.8879 ≤ ≤ -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 = 0 рівне = 3, тобто кількість додатних коренів рівняння рівна 3-м або менше від нього на парне число, тобто рівна 1.

Число знакозмін у рівнянні = 0 рівне = 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:

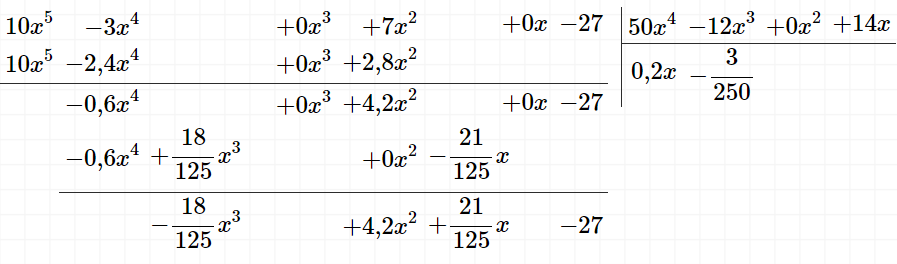
9 > 10 \* 0. Але Теорема Гюа є лише необхідною умовою дійсності всіх коренів алгебраїчного рівняння.

Виконаємо відокремлення коренів для 𝑓(x) = 10 \* -3 \* + 7 \* -27 = 0 методом Штурма

Побудуємо ряд Штурма

(x) = 𝑓(x) = 10 \* - 3 \* + 7 \* -27

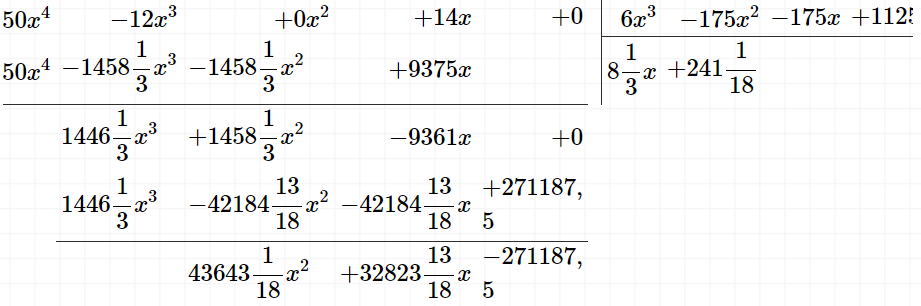
(x) = (x) = 50 \* - 12 \* + 14 \* x

Виконаємо ділення на та знайдемо таким чином

Помножимо залишок на 125/3, та візьмемо з протилежним знаком отримаємо

(x) = 6 - 175 – 7x + 1125

Виконаємо ділення на та знайдемо таким чином



Помножимо залишок на 18/125 та візьмемо з протилежним знаком, отримаємо

(x) = -6083 + 1105x + 39051

Таким же чином = 1255x – 146478 та = 1

|  |
| --- |
| Ряд Штурма |
| (x) = 10 \* - 3 \* + 7 \* -27 |
| (x) = 50 \* - 12 \* + 14 \* x |
| (x) = 6 - 175 – 7x + 1125 |
| (x) = -6083 + 1105x + 39051 |
| (x) = 1255x – 146478 |
| = 1 |

Визначимо знаки цих многочленів при x = − та при x = +

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x | (x) | (x) | (x) | (x) | (x) | (x) | К-сть знакозмін |
| + | + | + | + | - | + | + | 2 |
| − | - | + | - | - | - | + | 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 ≤ ≤ 3.7

Продовжимо локалізувати корені

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x | (x) | (x) | (x) | (x) | (x) | (x) | К-сть знакозмін |
| 0.7 | - | + | + | + | - | + | 3 |
| 0.9 | - | + | + | + | - | + | 3 |
| 1.1 | - | + | + | + | - | + | 3 |
| 1.3 | + | + | + | + | - | + | 2 |

Поліном **10 \* - 3 \* + 7 \* -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)  
 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()