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

Домінський Валентин

Викладач:

доц. Рибачук Л.В.

Київ – 2021

### Зміст

[Зміст 2](#_Toc69189898)

[1 Постановка задачі 3](#_Toc69189899)

[2 Розв’язок 4](#_Toc69189900)

[3 Розв’язок у Mathcad 6](#_Toc69189901)

[4 Лістинг програми 14](#_Toc69189902)

[Висновок: 14](#_Toc69189903)

### 1 Постановка задачі

1.Допрограмовий етап: визначити кількість дійсних коренів рівняння, відокремити корені рівняння (письмово) (див. теореми про верхню та нижню границі, Гюа, метод поліномів Штурма). Результатом є висновок: перший корінь належить проміжку […], другий корінь належить проміжку […] і т.д.

2.Програмний етап: уточнити корені рівняння:

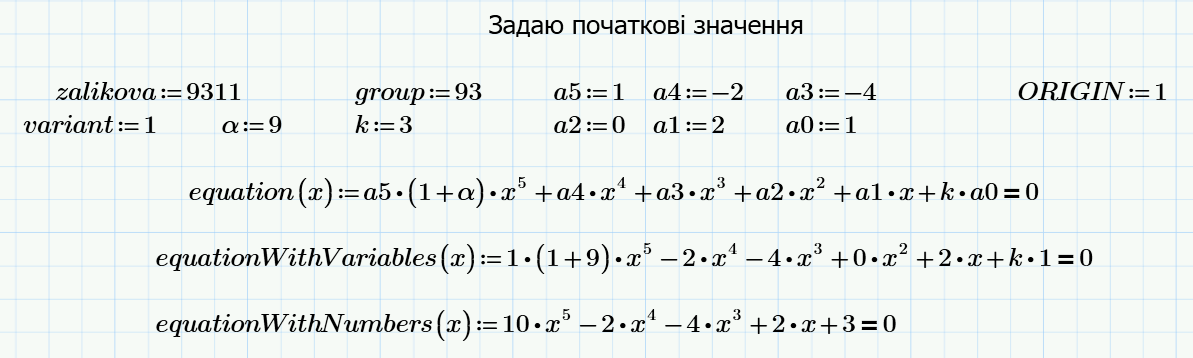
2.1.Методом бісекції.

2.2.Методом хорд.

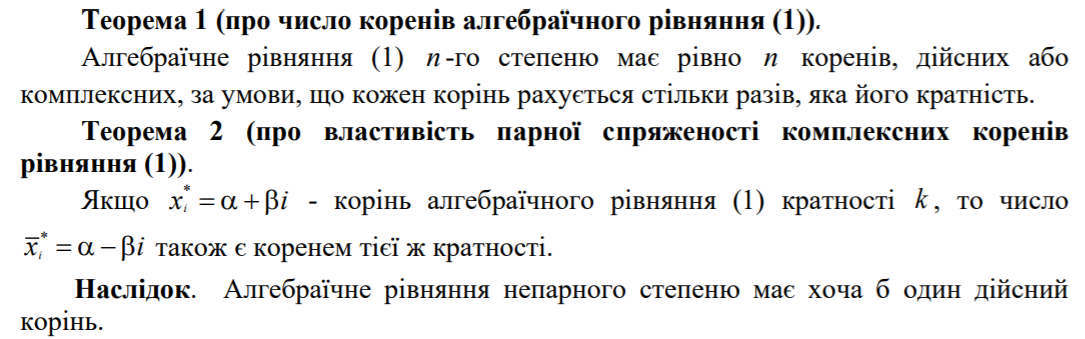
2.3.Методом Ньютона (дотичних)

### 2 Розв’язок

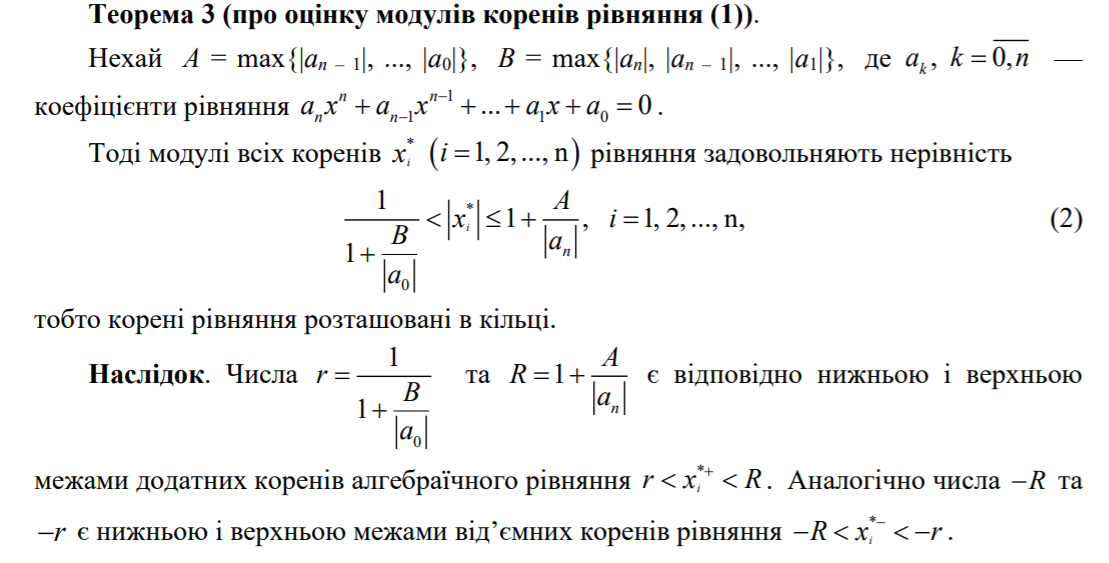
Допрограмовий етап:



Вираз = .

**

Оскільки Наше рівняння має непарний степінь (а точніше = 5), то в той же час буде існувати хоча б один дійсний корінь.

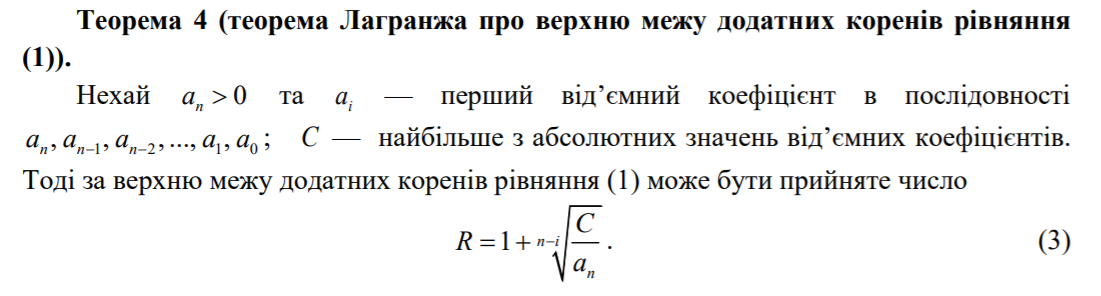


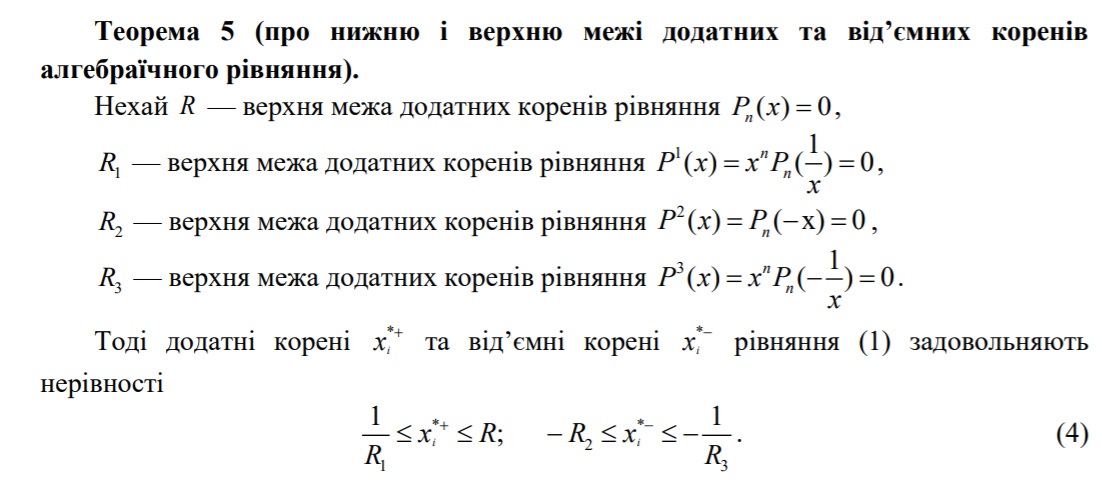
Знаючи про Теорему 3 знайдемо верхні та нижні межі коренів (як від’ємних, так і додатних)

Тобто всі корені лежать всередині цього кільця. За наслідком з теореми 3 це означає,

що додатні корені задовольняють нерівності:

а від’ємні — нерівності:





Знайдемо верхню межу додатних коренів:

(перший від’ємний коефіцієнт послідовності), то , а

Знайдемо нижню межу додатних коренів. Складемо рівняння:

Для цього рівняння , а

Звідси:

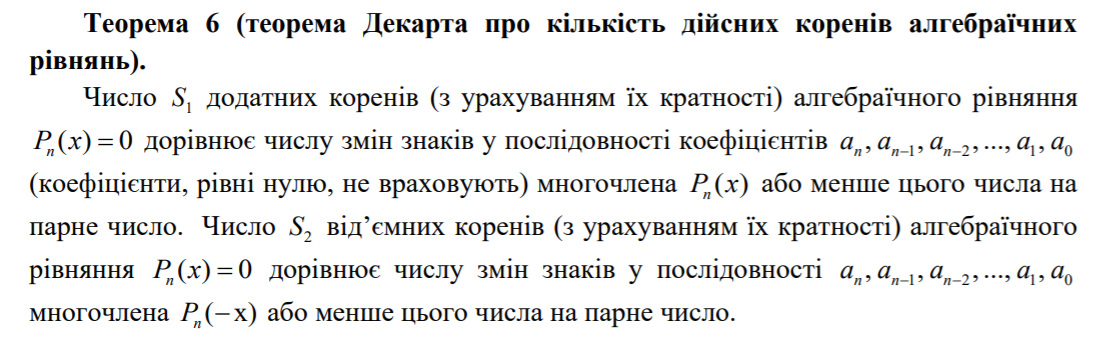
Уточнимо межі від'ємних коренів. Складемо рівняння:

Для цього рівняння , а

Складемо рівняння:

Для цього рівняння , а

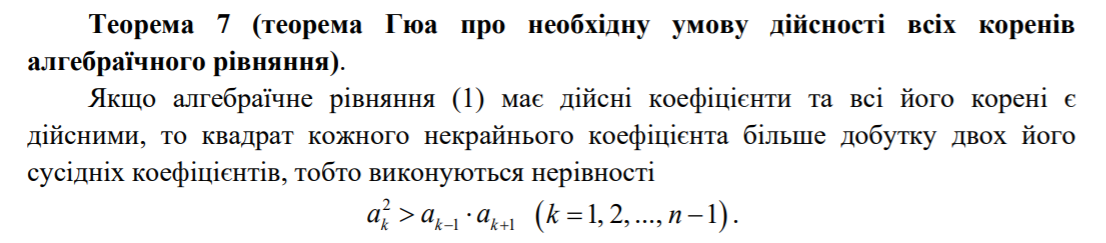
Звідси знаходимо:



Визначимо число додатних і від'ємних коренів. Виписуємо коефіцієнти многочлена

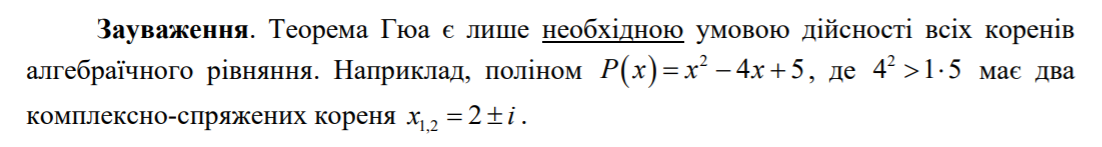
Оскільки число змін знаків C:\Users\Valentin\AppData\Local\Microsoft\Windows\Clipboard\HistoryData\{7ABEC192-DB27-45D4-8BA1-ED2D5FC39BF2}\{2FBA4CA8-5DD3-48BC-9ABC-377A5DBF212E}\ResourceMap\{54523DA6-3F4F-459A-893F-4ABF8922D698}= 2 , то число додатних коренів дорівнює 2 або менше на парне число, тобто їх взагалі немає.

Оскільки число змін знака = 3 , то число від’ємних коренів дорівнює 3 або менше на парне число, тобто дорівнює 1.



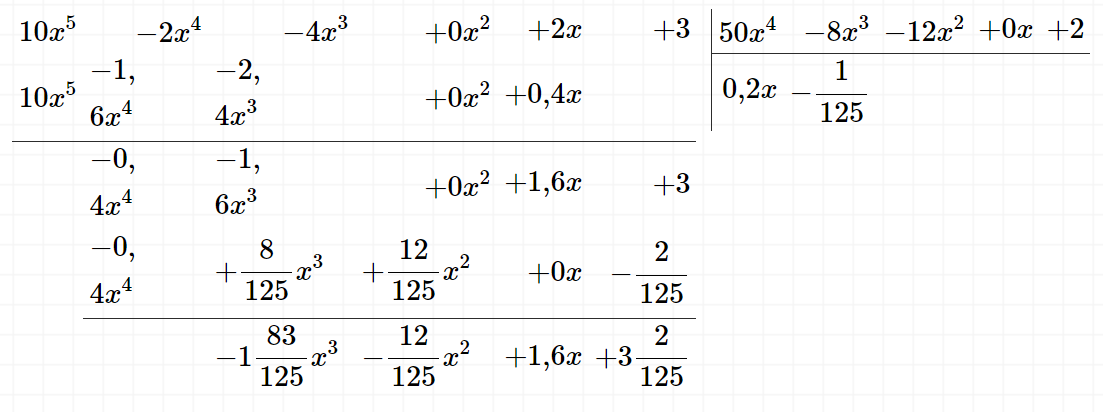
Використаємо цю теорему з k = 3.

Але:



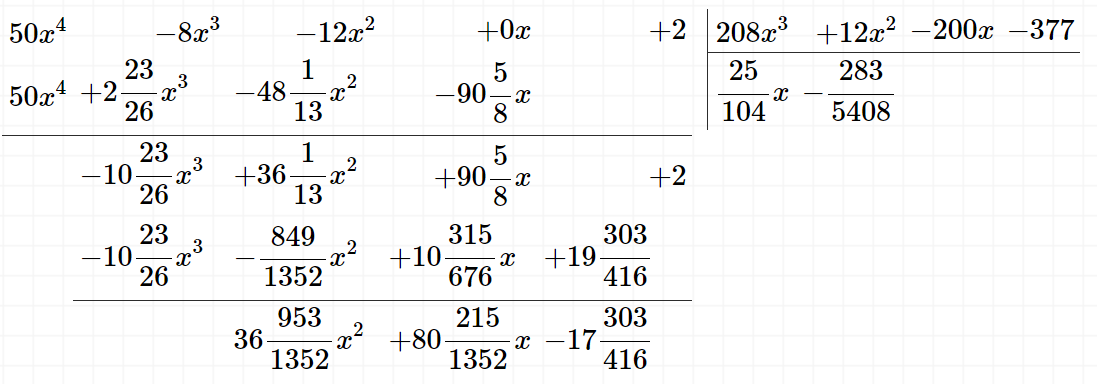
Перейдемо до методу відокремлення коренів Штурма:

Для знаходження наступних многочленів будемо брати залишок від ділення з протилежним знаком:

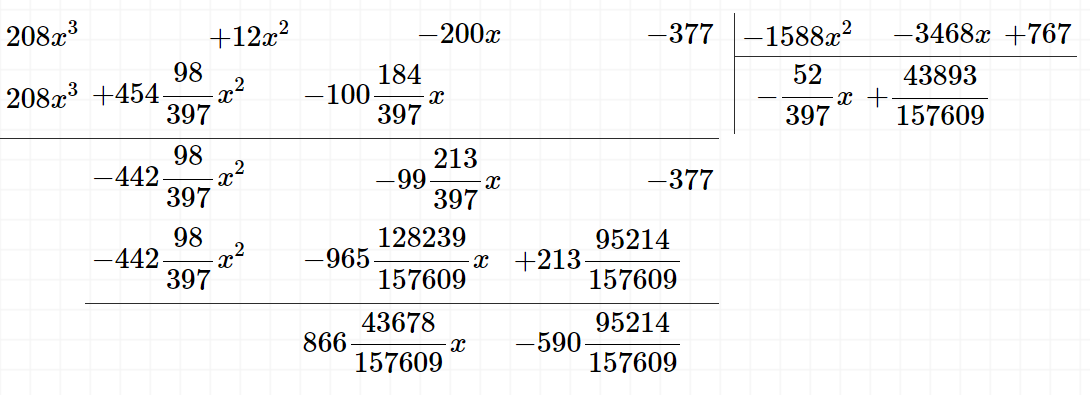


Тепер домножимо цю відповідь на (-125) (щоб отримати рівняння зі зворотнім знаком) і отримаємо:

Таким самим чином шукаємо наступні:



Тепер домножимо цю відповідь на (-(5408/125)) (щоб отримати рівняння зі зворотнім знаком) і отримаємо:

**

Тепер домножимо цю відповідь на (-(157609/676)) (щоб отримати рівняння зі зворотнім знаком) і отримаємо:

Тоді

Система многочленів Штурма:

Тепер знайдемо знаки Наших многочленів:

При :

При :

Отже кількість дійсних коренів для Нашого многочлена Штурма :

Тепер локалізуємо корені:

Оскільки число додатних коренів дорівнює 2 або менше на парне число, тобто їх взагалі немає, то корінь є від’ємним, отже він лежить у промІжку

Візьмемо крок 0.3, отже

При :

При :

При :

Отже корінь лежить між -1 та -0.7. Давайте тепер зменшимо крок до 0.1

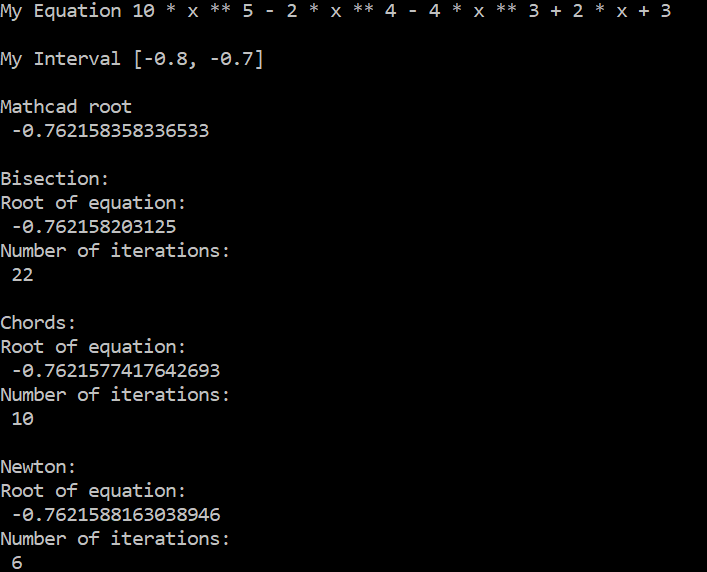
При :

При :

При :

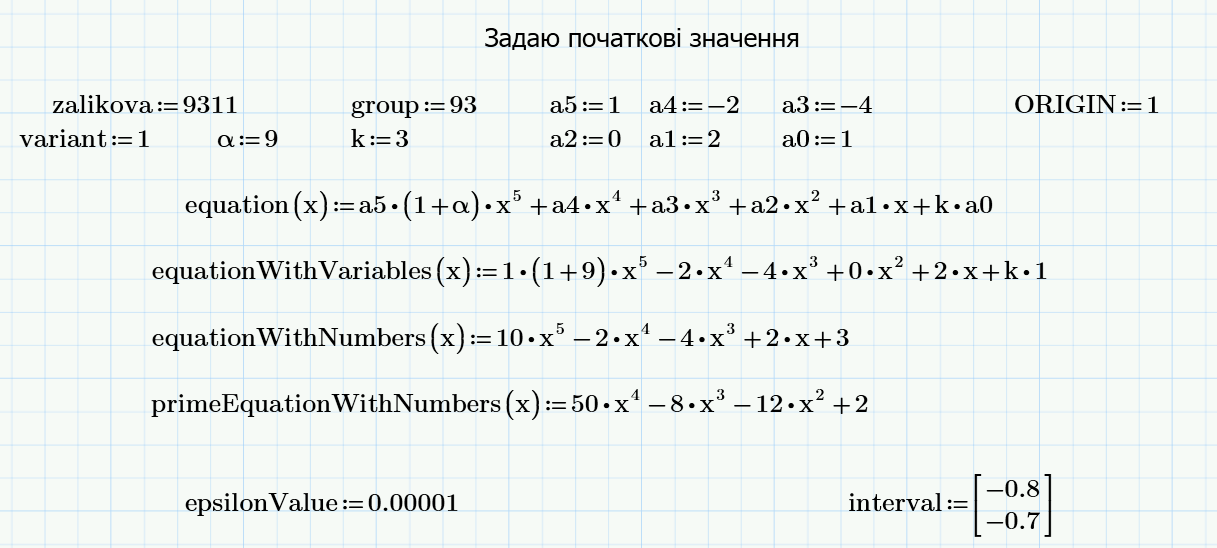
Отже корінь лежить між -0.8 та -0.7.

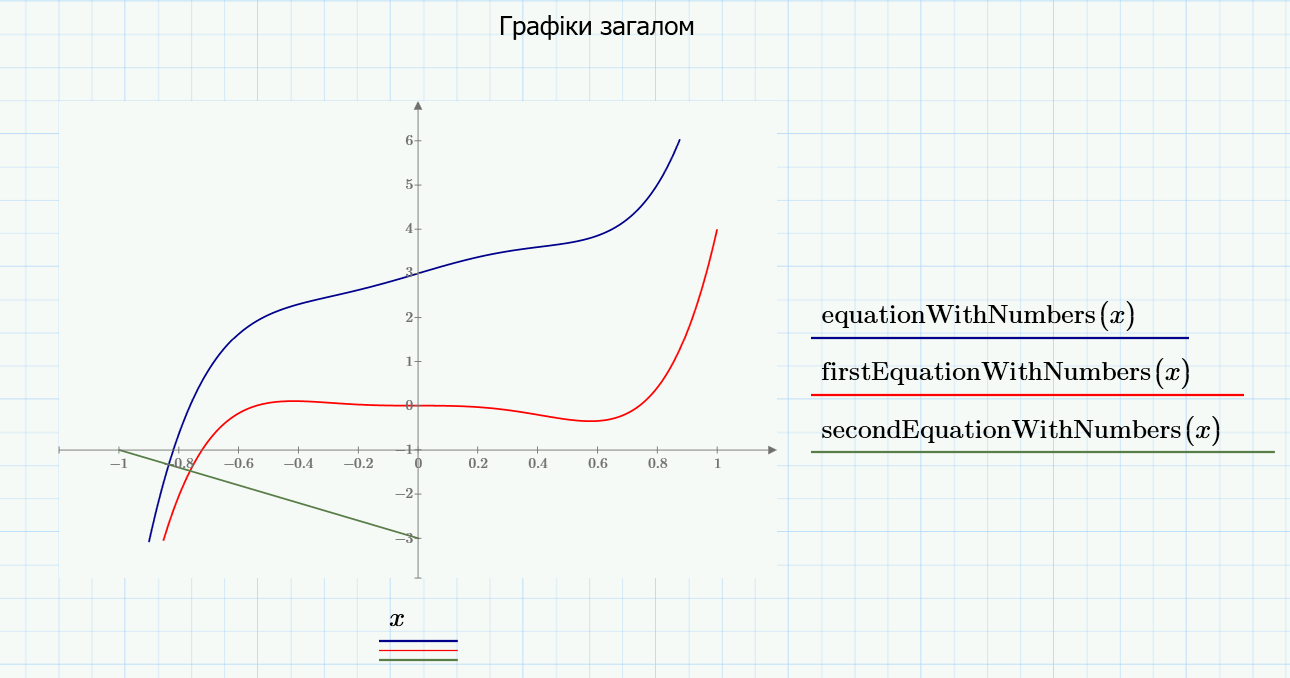
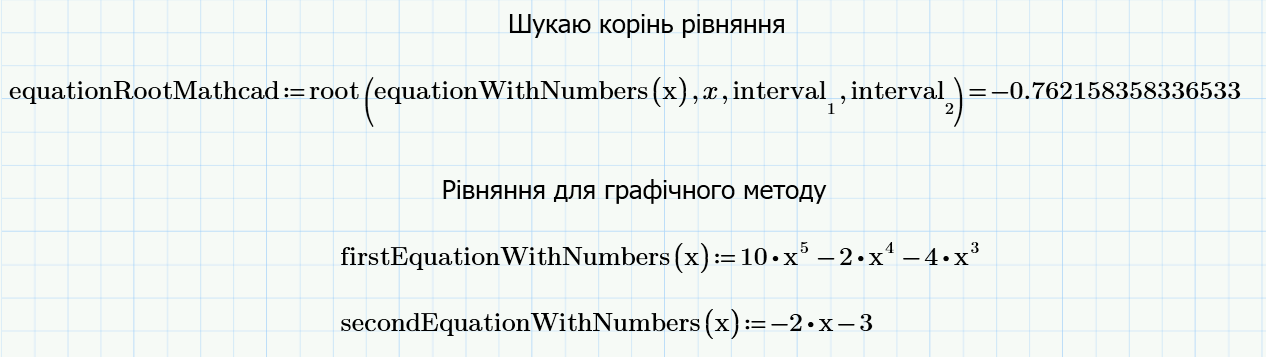
Вивід програми:

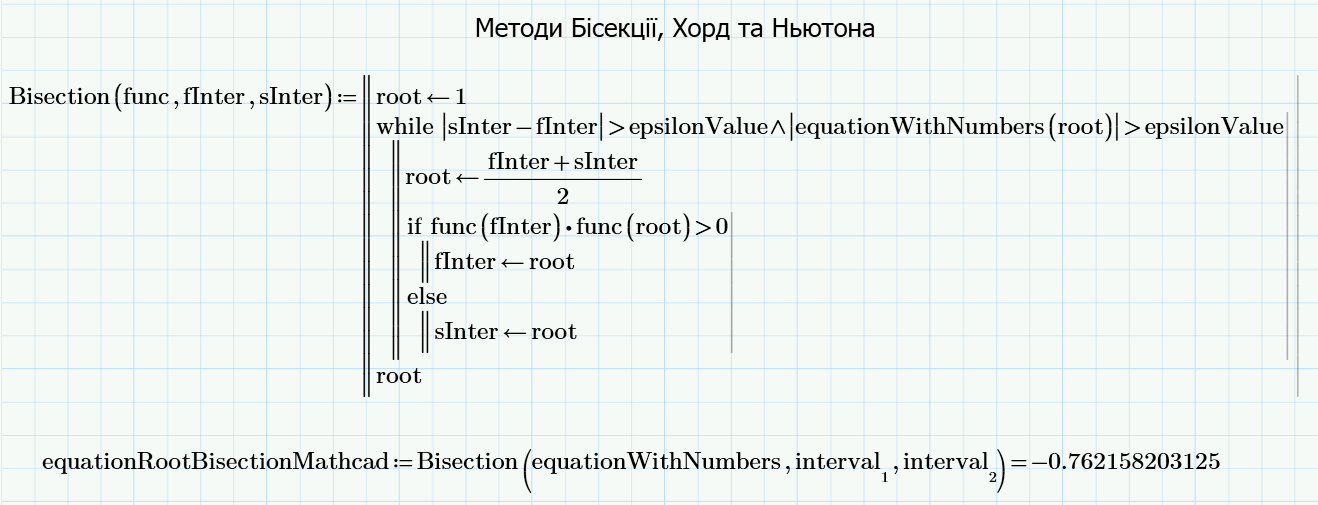
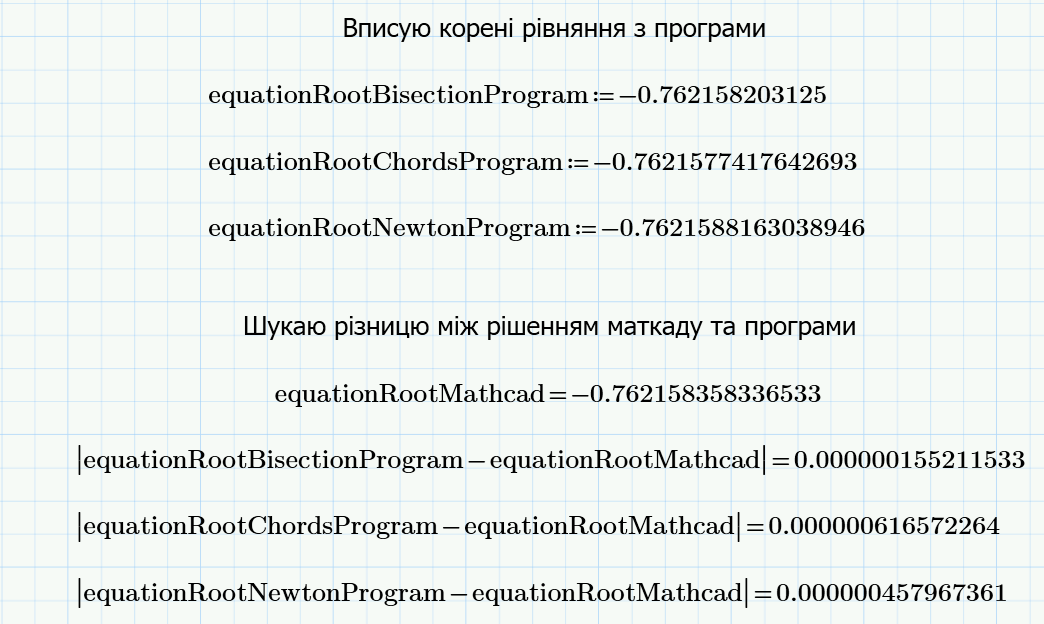
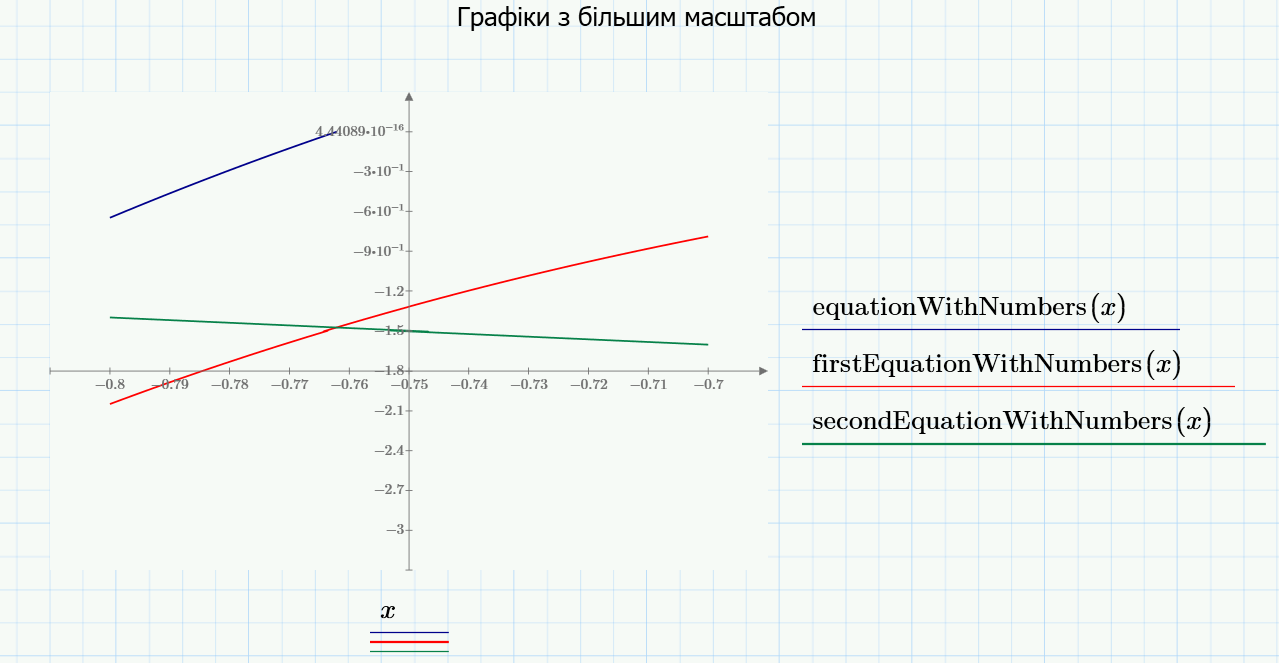


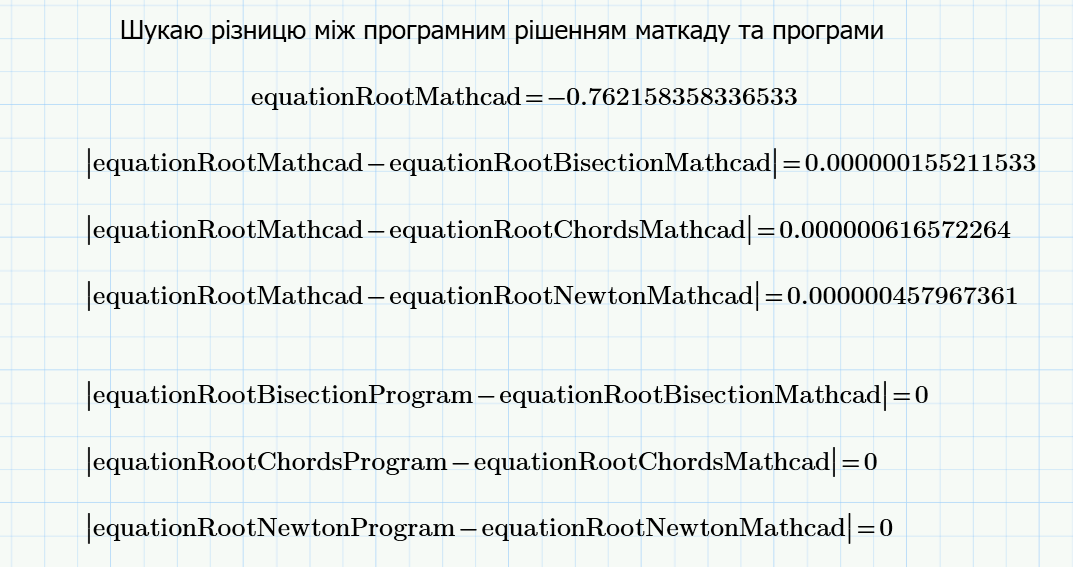
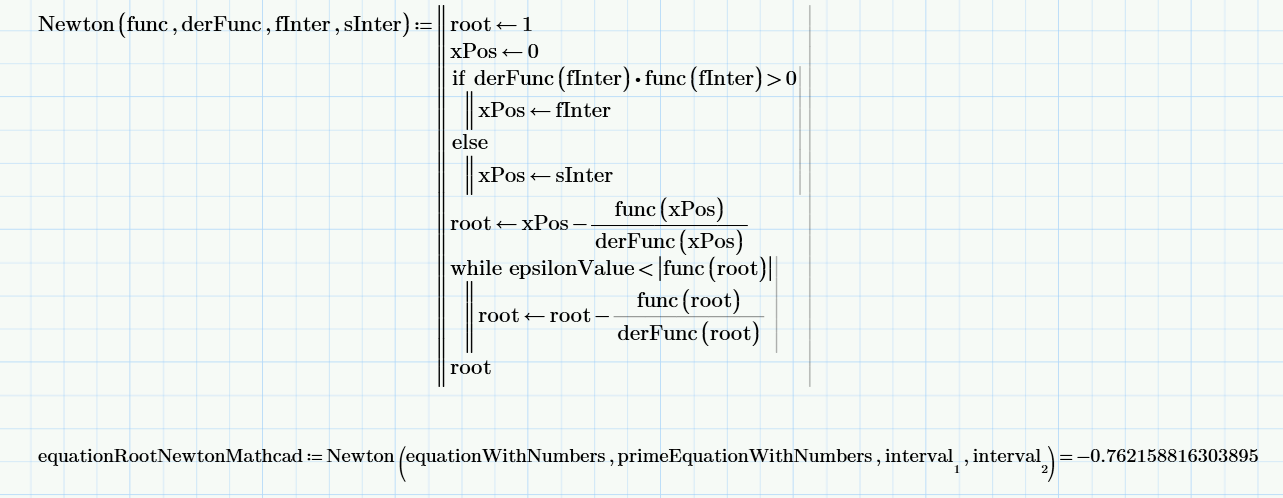
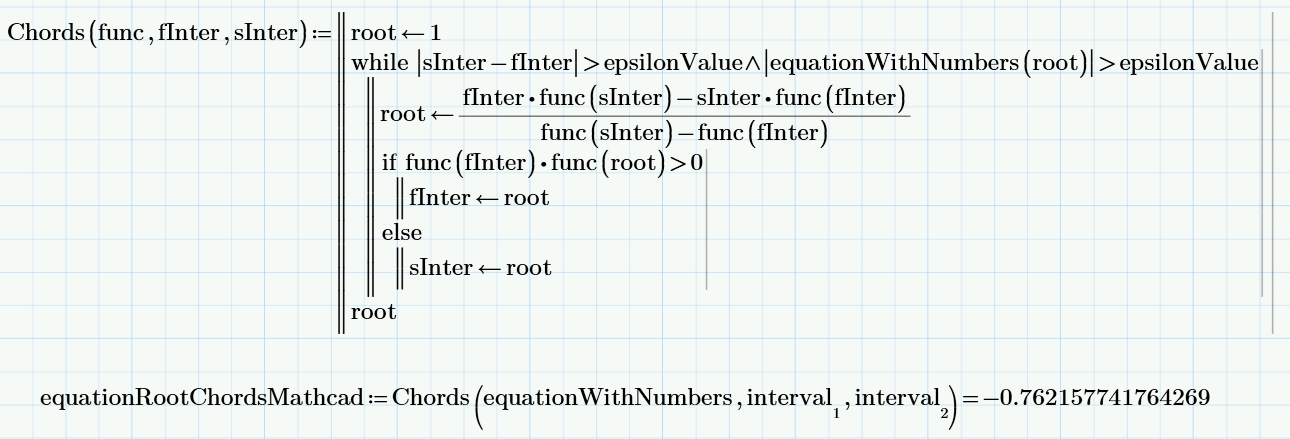
### 3 Розв’язок у Mathcad

Нижче наведено розв’язок у Mathcad









У технічних розрахунках точність вимірювань характеризують відносною похибкою. Результат вважають гарним, якщо відносна похибка не перевищує 0,1 %. Отже Наш результат є гарним

### 4 Лістинг програми

**Lab6.py**

# region Starting Values

epsilonValue **=** 0.00001

interval **=** **[-**0.8**,** **-**0.7**]**

polynomEquation **=** "10 \* x \*\* 5 - 2 \* x \*\* 4 - 4 \* x \*\* 3 + 2 \* x + 3"

rootFromMathcad **=** **-**0.762158358336533

half **=** 2

# endregion Starting Values

#region Default Functions

**def** MyFunction**(**x**):**

func **=** 10 **\*** x **\*\*** 5 **-** 2 **\*** x **\*\*** 4 **-** 4 **\*** x **\*\*** 3 **+** 2 **\*** x **+** 3

**return** func

**def** MyPrimeFunction**(**x**):**

func **=** 50 **\*** x **\*\*** 4 **-** 8 **\*** x **\*\*** 3 **-** 12 **\*** x **\*\*** 2 **+** 2

**return** func

#endregion Default Functions

#region Methods Functions

**def** BisectionAndChords**(**intervals**,** index**):**

numberOfIterations **=** 0

**for** i **in** intervals**:**

# shouldn't be 0

finalNumber **=** 1

firstInterval **=** intervals**[**0**]**

secondInterval **=** intervals**[**1**]**

# completion criterion

**while** epsilonValue **<** **abs(**MyFunction**(**finalNumber**))** **and** epsilonValue **<** **abs(**firstInterval **-** secondInterval**):**

# if 0, then do bisection

**if** index **==** 0**:**

finalNumber **=** **(**secondInterval **+** firstInterval**)** **/** half

# else do chords

**else:**

finalNumber **=** **(**MyFunction**(**secondInterval**)** **\*** firstInterval **-** MyFunction**(**firstInterval**)** **\*** secondInterval**)** **/** **(**MyFunction**(**secondInterval**)** **-** MyFunction**(**firstInterval**))**

**if** MyFunction**(**secondInterval**)** **\*** MyFunction**(**finalNumber**)** **<=** 0**:**

firstInterval **=** finalNumber

**else:**

secondInterval **=** finalNumber

numberOfIterations **=** numberOfIterations **+** 1

**return** finalNumber**,** numberOfIterations

**def** Newton**(**intervals**):**

numberOfIterations **=** 0

**for** i **in** intervals**:**

initialXPos **=** 0

firstInterval **=** intervals**[**0**]**

secondInterval **=** intervals**[**1**]**

# if same symbols (like + && +)

**if** MyFunction**(**firstInterval**)** **\*** MyPrimeFunction**(**firstInterval**)** **>** 0**:**

initialXPos **=** secondInterval

# if different symbols (like - && +)

**else:**

initialXPos **=** firstInterval

finalNumber **=** initialXPos **-** MyFunction**(**initialXPos**)** **/** MyPrimeFunction**(**initialXPos**)**

numberOfIterations **=** numberOfIterations **+** 1

# completion criterion

**while** epsilonValue **<** **abs(**MyFunction**(**finalNumber**)):**

finalNumber **=** finalNumber **-** MyFunction**(**finalNumber**)** **/** MyPrimeFunction**(**finalNumber**)**

numberOfIterations **=** numberOfIterations **+** 1

**return** finalNumber**,** numberOfIterations

#endregion Methods Functions

#region Results

**print(**"My Equation"**,** polynomEquation**)**

**print(**"\nMy Interval"**,** interval**)**

**print(**"\nMathcad root\n"**,** rootFromMathcad**)**

**print(**"\nBisection:"**)**

bisectionRoots**,** bisectionIterations **=** BisectionAndChords**(**interval**,** 0**)**

**print(**"Root of equation:\n"**,** bisectionRoots**,** "\nNumber of iterations:\n"**,** bisectionIterations**)**

**print(**"\nChords:"**)**

chordsRoots**,** chordsIterations **=** BisectionAndChords**(**interval**,** 1**)**

**print(**"Root of equation:\n"**,** chordsRoots**,** "\nNumber of iterations:\n"**,** chordsIterations**)**

**print(**"\nNewton:"**)**

newtonRoots**,** newtonIterations **=** Newton**(**interval**)**

**print(**"Root of equation:\n"**,** newtonRoots**,** "\nNumber of iterations:\n"**,** newtonIterations**)**

#endregion Results

### Висновок:

Я навчився використовувати різні методи розв’язання нелінійних рівнянь (методи бісекції, хорд та Ньютона), відокремлювати корені рівнянь (допрограмний етап), а також знаходити їх з певною точністю. Також можна дійти до висновку, що метод бісекції – найпростіший, але в той же час найдовший, метод хорд дуже схожий на перший, але сильно залежить від інтервала (при великому значенні к-сть ітерацій може бути більшою, ніж у методі бісекції), а метод Ньютона – найскладніший, але в той же час має меншу к-сть ітерацій та одну з найменших похибок