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 7](#_Toc69189901)

[4 Лістинг програми 12](#_Toc69189902)

[Висновок: 12](#_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:

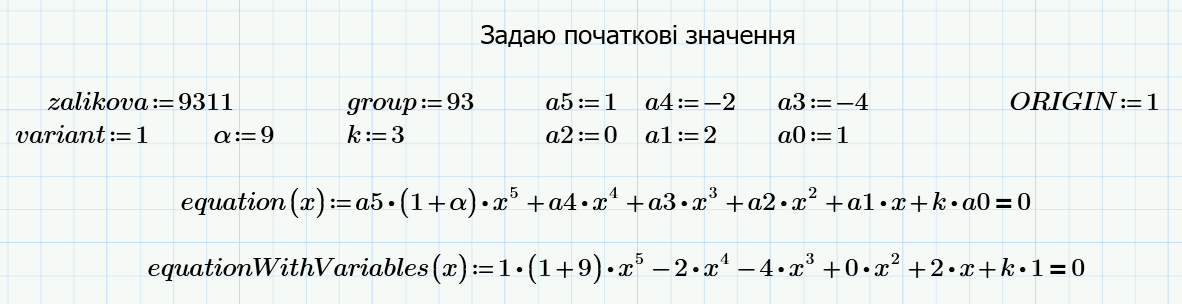
2.1.Методом бісекції.

2.2.Методом хорд.

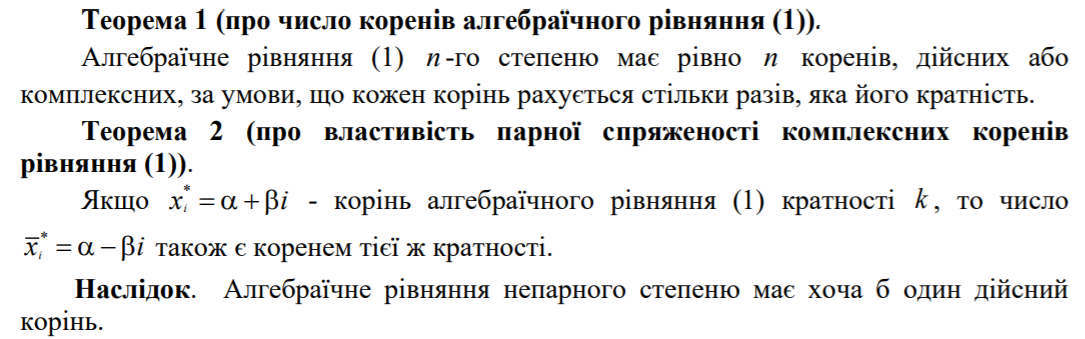
2.3.Методом Ньютона (дотичних)

### 2 Розв’язок

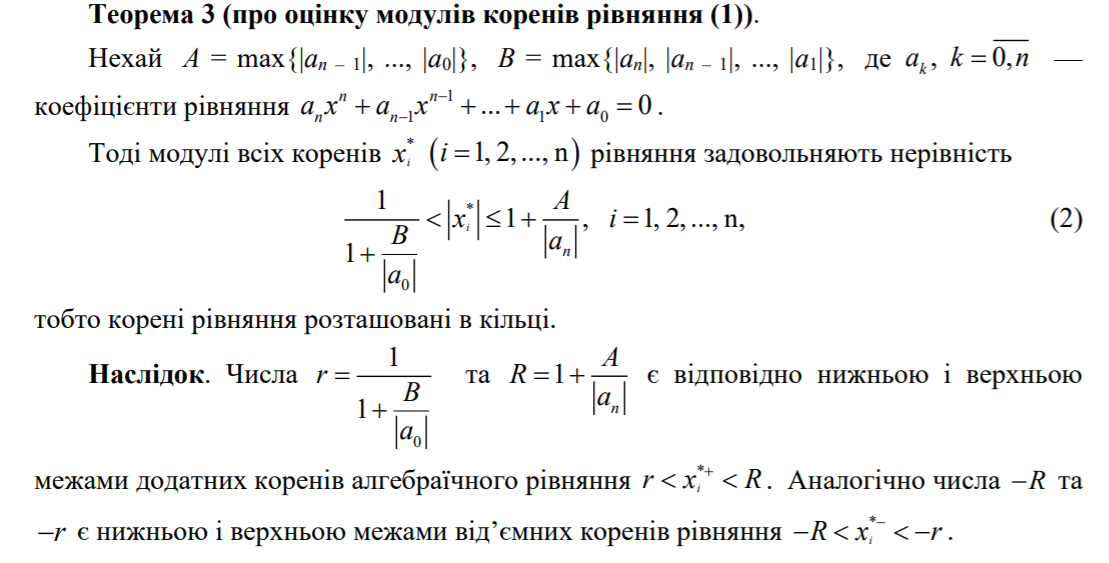
Допрограмовий етап:



Вираз = .

**

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

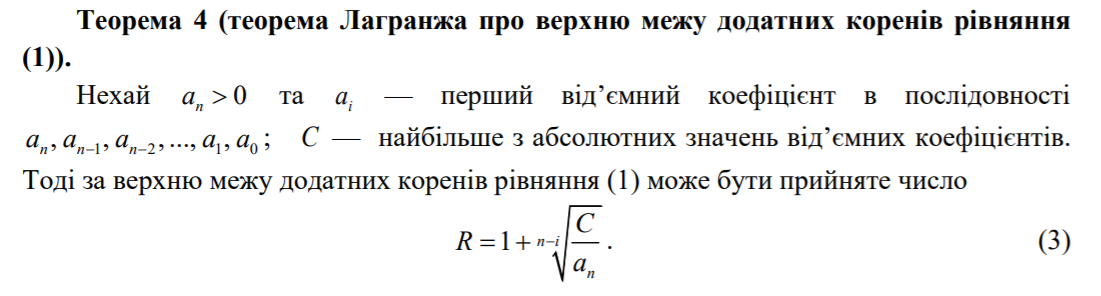


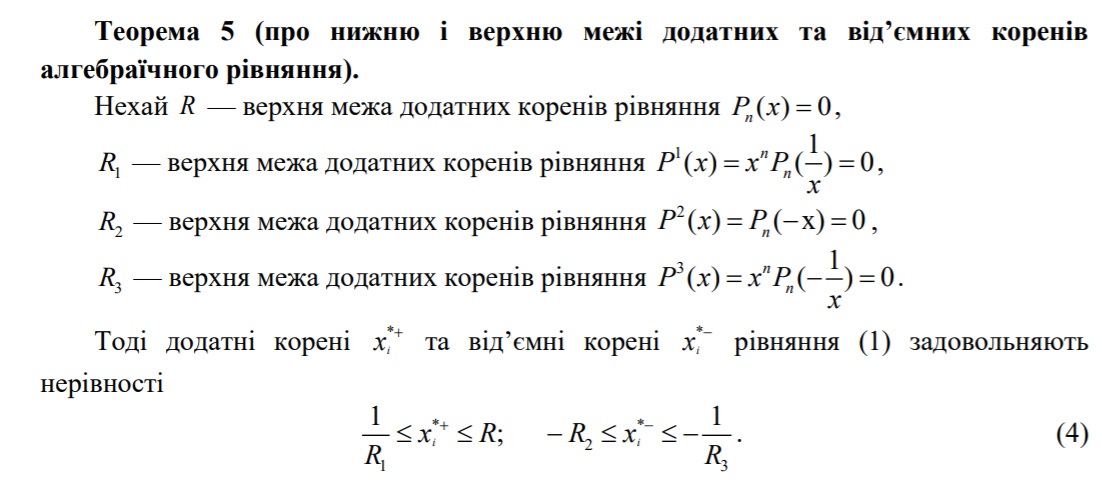
Знаючи про Теорему 3 знайдемо верхні та нижні межі коренів (як від’ємних, так і додатних)

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:

3 Розв’язок у Mathcad

Нижче наведено розв’язок системи 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

**Lab5.py**

# region Starting Values

**import** numpy **as** np

np**.**set\_printoptions**(**suppress**=True)**

**from** math **import** sin

rounding **=** 5

N **=** 5

numberOfUnknown **=** 12

lengthOfRowForMatrix **=** numberOfUnknown **+** 1

differenceBetweenTwoPoints **=** 2

firstIndex **=** 1

secondIndex **=** 3

thirdIndex **=** 7

lastIndex **=** 12

Xarray **=** **[**3**,** 5**,** 7**,** 9**,** 11**]**

# region xValues for Faults

XarrayForFault1 **=** **[**3**,** 3.25**,** 3.5**,** 3.75**,** 4**]**

XarrayForFault2 **=** **[**4.25**,** 4.5**,** 4.75**,** 5**,** 5.25**]**

XarrayForFault3 **=** **[**5.5**,** 5.75**,** 6**,** 6.25**,** 6.5**]**

XarrayForFault4 **=** **[**6.75**,** 7**,** 7.25**,** 7.5**,** 7.75**]**

XarrayForFault5 **=** **[**8**,** 8.25**,** 8.5**,** 8.75**,** 9**]**

XarrayForFault6 **=** **[**9.25**,** 9.5**,** 9.75**,** 10**,** 10.25**]**

XarrayForFault7 **=** **[**10.5**,** 10.75**,** 11**]**

# endregion xValues for Faults

# My Sin Function

**def** MySinFun**(**x**)** **->** **float:**

element **=** sin**(**3 **/** 2 **\*** x**)** **+** **(**x **\*** 3**)** **\*\*** **(**1 **/** 3**)**

**return** element

Yarray **=** **[**MySinFun**(**Xarray**[**0**]),** MySinFun**(**Xarray**[**1**]),** MySinFun**(**Xarray**[**2**]),** MySinFun**(**Xarray**[**3**]),** MySinFun**(**Xarray**[**4**])]**

# region yValues Faults

YarrayForFault1 **=** **[**MySinFun**(**XarrayForFault1**[**0**]),** MySinFun**(**XarrayForFault1**[**1**]),** MySinFun**(**XarrayForFault1**[**2**]),** MySinFun**(**XarrayForFault1**[**3**]),** MySinFun**(**XarrayForFault1**[**4**])]**

YarrayForFault2 **=** **[**MySinFun**(**XarrayForFault2**[**0**]),** MySinFun**(**XarrayForFault2**[**1**]),** MySinFun**(**XarrayForFault2**[**2**]),** MySinFun**(**XarrayForFault2**[**3**]),** MySinFun**(**XarrayForFault2**[**4**])]**

YarrayForFault3 **=** **[**MySinFun**(**XarrayForFault3**[**0**]),** MySinFun**(**XarrayForFault3**[**1**]),** MySinFun**(**XarrayForFault3**[**2**]),** MySinFun**(**XarrayForFault3**[**3**]),** MySinFun**(**XarrayForFault3**[**4**])]**

YarrayForFault4 **=** **[**MySinFun**(**XarrayForFault4**[**0**]),** MySinFun**(**XarrayForFault4**[**1**]),** MySinFun**(**XarrayForFault4**[**2**]),** MySinFun**(**XarrayForFault4**[**3**]),** MySinFun**(**XarrayForFault4**[**4**])]**

YarrayForFault5 **=** **[**MySinFun**(**XarrayForFault5**[**0**]),** MySinFun**(**XarrayForFault5**[**1**]),** MySinFun**(**XarrayForFault5**[**2**]),** MySinFun**(**XarrayForFault5**[**3**]),** MySinFun**(**XarrayForFault5**[**4**])]**

YarrayForFault6 **=** **[**MySinFun**(**XarrayForFault6**[**0**]),** MySinFun**(**XarrayForFault6**[**1**]),** MySinFun**(**XarrayForFault6**[**2**]),** MySinFun**(**XarrayForFault6**[**3**]),** MySinFun**(**XarrayForFault6**[**4**])]**

YarrayForFault7 **=** **[**MySinFun**(**XarrayForFault7**[**0**]),** MySinFun**(**XarrayForFault7**[**1**]),** MySinFun**(**XarrayForFault7**[**2**])]**

# endregion xValues Faults

# endregion Starting Values

# region Prints

# Print matrix

**def** PrintMatrixAsNp**(**matrixName**,**matrix**):**

**print(**"\n"**,** matrixName**,**"="**)**

npMatrix **=** np**.**array**(**matrix**)**

**print(**npMatrix**.round(**rounding**))**

# Print vector

**def** PrintVectorAsNp**(**vectorName**,** vector**):**

**print(**"\n"**,** vectorName**,**"="**)**

npMatrix **=** np**.**array**(**vector**)**

**print(**npMatrix**.round(**rounding**))**

# Print Lagrange

**def** PrintLagrange**(**Xarray**,** Yarray**):**

PrintVectorAsNp**(**"X"**,** Xarray**)**

PrintVectorAsNp**(**"Y"**,** Yarray**)**

**print(**"\nLagrange Polynom:"**)**

firstPart **=** f"\n{Yarray**[**0**]**} \* ((x - {Xarray**[**1**]**})/({Xarray**[**0**]**} - {Xarray**[**1**]**})) \* ((x - {Xarray**[**2**]**})/({Xarray**[**0**]**} - {Xarray**[**2**]**})) \* ((x - {Xarray**[**3**]**})/({Xarray**[**0**]**} - {Xarray**[**3**]**})) \* ((x - {Xarray**[**4**]**})/({Xarray**[**0**]**} - {Xarray**[**4**]**})) +"

secondPart **=** f"{Yarray**[**1**]**} \* ((x - {Xarray**[**0**]**})/({Xarray**[**1**]**} - {Xarray**[**0**]**})) \* ((x - {Xarray**[**2**]**})/({Xarray**[**1**]**} - {Xarray**[**2**]**})) \* ((x - {Xarray**[**3**]**})/({Xarray**[**1**]**} - {Xarray**[**3**]**})) \* ((x - {Xarray**[**4**]**})/({Xarray**[**1**]**} - {Xarray**[**4**]**})) +"

thirdPart **=** f"{Yarray**[**2**]**} \* ((x - {Xarray**[**0**]**})/({Xarray**[**2**]**} - {Xarray**[**0**]**})) \* ((x - {Xarray**[**1**]**})/({Xarray**[**2**]**} - {Xarray**[**1**]**})) \* ((x - {Xarray**[**3**]**})/({Xarray**[**2**]**} - {Xarray**[**3**]**})) \* ((x - {Xarray**[**4**]**})/({Xarray**[**2**]**} - {Xarray**[**4**]**})) +"

fourthPart **=** f"{Yarray**[**3**]**} \* ((x - {Xarray**[**0**]**})/({Xarray**[**3**]**} - {Xarray**[**0**]**})) \* ((x - {Xarray**[**1**]**})/({Xarray**[**3**]**} - {Xarray**[**1**]**})) \* ((x - {Xarray**[**2**]**})/({Xarray**[**3**]**} - {Xarray**[**2**]**})) \* ((x - {Xarray**[**4**]**})/({Xarray**[**3**]**} - {Xarray**[**4**]**})) +"

fifthPart **=** f"{Yarray**[**4**]**} \* ((x - {Xarray**[**0**]**})/({Xarray**[**4**]**} - {Xarray**[**0**]**})) \* ((x - {Xarray**[**1**]**})/({Xarray**[**4**]**} - {Xarray**[**1**]**})) \* ((x - {Xarray**[**2**]**})/({Xarray**[**4**]**} - {Xarray**[**2**]**})) \* ((x - {Xarray**[**3**]**})/({Xarray**[**4**]**} - {Xarray**[**3**]**}))"

**print(**firstPart**,** secondPart**,** thirdPart**,** fourthPart**,** fifthPart**,** "\n"**)**

# endregion Prints

# Implementing Lagrange Interpolation

**def** Lagrange**(**X\_array**,** Y\_array**,** pointToShow**,** show**)** **->** **float:**

# Create some prerequisites

resultAsYpoint **=** 0

**for** k **in** **range(len(**Y\_array**)):**

# Create some prerequisites

tempPoint1 **=** 0

tempPoint2 **=** 0

tempPointArray **=** **[**tempPoint1**,** tempPoint2**]**

tempPointArray**[**0**]** **=** 1

tempPointArray**[**1**]** **=** 1

**for** m **in** **range(len(**X\_array**)):**

**if** m **==** k**:**

**for** z **in** **range(**2**):**

tempPointArray**[**z**]** **=** tempPointArray**[**z**]** **\*** 1

**else:**

tempPointArray**[**1**]** **=** tempPointArray**[**1**]** **\*** **(**X\_array**[**k**]** **-** X\_array**[**m**])**

tempPointArray**[**0**]** **=** tempPointArray**[**0**]** **\*** **(**pointToShow **-** X\_array**[**m**])**

resultAsYpoint **=** resultAsYpoint **+** Y\_array**[**k**]** **\*** tempPointArray**[**0**]** **/** tempPointArray**[**1**]**

**if** show**:**

**print(**"Coef of"**,**pointToShow**,**"element ="**,**resultAsYpoint**)**

**return** resultAsYpoint

**def** CreateMatrixForCramer**(**Xarray**,** Yarray**)** **->** **[list,** **list]:**

# Create some prerequisites

matrixForCramer **=** **list()**

rightPartForCramer **=** **[**0**]** **\*** **(**lengthOfRowForMatrix **-** 1**)**

# Call Our functions to create matrix

FirstPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**)**

SecondPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**)**

ThirdPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**)**

FourthPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**)**

FifthPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**)**

# Create Right Part

**for** i **in** **range(len(**matrixForCramer**)):**

rightPartForCramer**[**i**]** **=** matrixForCramer**[**i**][-**1**]**

# Clean matrix

matrixForCramer **=** np**.**delete**(**matrixForCramer**,** np**.**s\_**[-**1**:],** axis**=**1**)**

**return** rightPartForCramer**,** matrixForCramer

#region PartsOfEquation

**def** FirstPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**):**

**for** i **in** **range(**1**,** **len(**Xarray**)):**

queue **=** **[**0**]** **\*** lengthOfRowForMatrix

queue**[**i**-**firstIndex**]** **=** differenceBetweenTwoPoints**;** queue**[**i**+**secondIndex**]** **=** differenceBetweenTwoPoints **\*\*** 2

queue**[**i**+**thirdIndex**]** **=** differenceBetweenTwoPoints **\*\*** 3**;** queue**[**lastIndex**]** **=** Yarray**[**i**]** **-** Yarray**[**i **-** 1**]**

matrixForCramer**.**append**(**queue**)**

**def** SecondPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**):**

**for** i **in** **range(**1**,** **len(**Xarray**)** **-** 1**):**

queue **=** **[**0**]** **\*** lengthOfRowForMatrix

queue**[**i**]** **=** 1**;** queue**[**i**-**firstIndex**]** **=** **-**1**;** queue**[**lastIndex**]** **=** 0

queue**[**i**+**secondIndex**]** **=** **-**2 **\*** differenceBetweenTwoPoints**;** queue**[**i**+**thirdIndex**]** **=** **-**3 **\*** differenceBetweenTwoPoints **\*\*** 2

matrixForCramer**.**append**(**queue**)**

**def** ThirdPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**):**

**for** i **in** **range(**1**,** **len(**Xarray**)** **-** 1**):**

queue **=** **[**0**]** **\*** lengthOfRowForMatrix

queue**[**i**+**secondIndex**+**1**]** **=** 1**;** queue**[**i**+**secondIndex**]** **=** **-**1**;** queue**[**i**+**thirdIndex**]** **=** **-**3 **\*** differenceBetweenTwoPoints**;** queue**[**lastIndex**]** **=** 0

matrixForCramer**.**append**(**queue**)**

**def** FourthPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**):**

queue **=** **[**0**]** **\*** lengthOfRowForMatrix

queue**[**thirdIndex**]** **=** 1**;** queue**[**lastIndex**-**1**]** **=** 3 **\*** differenceBetweenTwoPoints**;** queue**[**lastIndex**]** **=** 0

matrixForCramer**.**append**(**queue**)**

**def** FifthPartOfEquation**(**Xarray**,** Yarray**,** matrixForCramer**):**

queue **=** **[**0**]** **\*** lengthOfRowForMatrix

queue**[**secondIndex**+**1**]** **=** 1**;** queue**[**lastIndex**]** **=** 0

matrixForCramer**.**append**(**queue**)**

#endregion PartsOfEquation

# Do Cramer Method

**def** Cramer**(**matrixForComputations**,** matrixForCramer**,** rightPartForCramer**)** **->** **list:**

# Create some prerequisites

SIPcoeffs **=** **list()**

**for** k **in** **range(**0**,** **len(**rightPartForCramer**)):**

**for** h **in** **range(**0**,** **len(**rightPartForCramer**)):**

# Get values for right part

matrixForComputations**[**h**][**k**]** **=** rightPartForCramer**[**h**]**

**if** k **>** 0**:**

matrixForComputations**[**h**][**k **-** 1**]** **=** matrixForCramer**[**h**][**k **-** 1**]**

# Add value to the vector

SIPcoeffs**.**append**(**np**.**linalg**.**det**(**matrixForComputations**)** **/** np**.**linalg**.**det**(**matrixForCramer**))**

# Rounding values

SIPcoeffs**[**k**]** **=** **round(**SIPcoeffs**[**k**],** rounding**)**

**return** SIPcoeffs

# Printing Lagrange Equation

PrintLagrange**(**Xarray**,** Yarray**)**

# region Faults

**for** i **in** **range(**N**):**

Lagrange**(**Xarray**,** Yarray**,** Xarray**[**i**],** **True)**

**print(**"Fault of element"**,** Xarray**[**i**],** "="**,** **abs(**MySinFun**(**Xarray**[**i**])** **-** Lagrange**(**Xarray**,** Yarray**,** Xarray**[**i**],** **False)))**

**print(**"\n"**)**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault1**,** YarrayForFault1**,** XarrayForFault1**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault1**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault1**[**i**])** **-** Lagrange**(**XarrayForFault1**,** YarrayForFault1**,** XarrayForFault1**[**i**],** **False)))**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault2**,** YarrayForFault2**,** XarrayForFault2**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault2**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault2**[**i**])** **-** Lagrange**(**XarrayForFault2**,** YarrayForFault2**,** XarrayForFault2**[**i**],** **False)))**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault3**,** YarrayForFault3**,** XarrayForFault3**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault3**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault3**[**i**])** **-** Lagrange**(**XarrayForFault3**,** YarrayForFault3**,** XarrayForFault3**[**i**],** **False)))**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault4**,** YarrayForFault4**,** XarrayForFault4**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault4**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault4**[**i**])** **-** Lagrange**(**XarrayForFault4**,** YarrayForFault4**,** XarrayForFault4**[**i**],** **False)))**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault5**,** YarrayForFault5**,** XarrayForFault5**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault5**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault5**[**i**])** **-** Lagrange**(**XarrayForFault5**,** YarrayForFault5**,** XarrayForFault5**[**i**],** **False)))**

**for** i **in** **range(**N**):**

Lagrange**(**XarrayForFault6**,** YarrayForFault6**,** XarrayForFault6**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault6**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault6**[**i**])** **-** Lagrange**(**XarrayForFault6**,** YarrayForFault6**,** XarrayForFault6**[**i**],** **False)))**

**for** i **in** **range(len(**XarrayForFault7**)):**

Lagrange**(**XarrayForFault7**,** YarrayForFault7**,** XarrayForFault7**[**i**],** **True)**

**print(**"Fault of element"**,** XarrayForFault7**[**i**],** "="**,** **abs(**MySinFun**(**XarrayForFault7**[**i**])** **-** Lagrange**(**XarrayForFault7**,** YarrayForFault7**,** XarrayForFault7**[**i**],** **False)))**

# endregion Faults

rightPartForCramer**,** matrixForCramer **=** CreateMatrixForCramer**(**Xarray**.**copy**(),** Yarray**.**copy**())**

matrixForComputations **=** **list(map(list,** matrixForCramer**))**

# Coefficients of spline interpolation polynomials

SIPcoeffs **=** Cramer**(**matrixForComputations**,** matrixForCramer**,** rightPartForCramer**)**

PrintVectorAsNp**(**"Coefficients of spline interpolation polynomial"**,** SIPcoeffs**)**

### Висновок:

Я навчився використовувати різні методи інтерполяції, визначати сплайн коефіцієнти. Покращив навички роботи з графіками у маткад.