Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Обчислювальної Техніки

Лабораторна робота № 8

з дисципліни «Чисельні методи»

на тему

«**Розв’язання задачі Коші**»

Виконав:

студент гр. ІП-93

Домінський Валентин

Викладач:

доц. Рибачук Л.В.

Київ – 2021

### Зміст

[Зміст 2](#_Toc71617206)

[1 Постановка задачі 3](#_Toc71617207)

[2 Розв’язок 4](#_Toc71617208)

[3 Розв’язок у Mathcad 4](#_Toc71617209)

[4 Лістинг програми 6](#_Toc71617210)

[Висновок: 8](#_Toc71617211)

### 1 Постановка задачі

Методами Рунге-Кутта та Адамса розв'язати задачу Коші. На початку інтервалу у необхідній кількості точок значення для методу Адамса визначити методом Рунге-Кутта четвертого порядку.

Для фіксованого h потрібно навести:

− значення наближеного розв'язку y(x) у тих самих точках, одержані обома методами;

− значення функції помилки ε(x) для обох методів;

графіки:

− обох наближених - на одному малюнку;

− обох помилок - на другому малюнку.

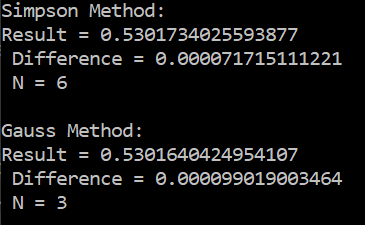
Розв’язати задане рівняння за допомогою Matchad, порівняти із власними

результатами.

Розв’язати за допомогою Matchad систему рівнянь, побудувати графік 0 y та

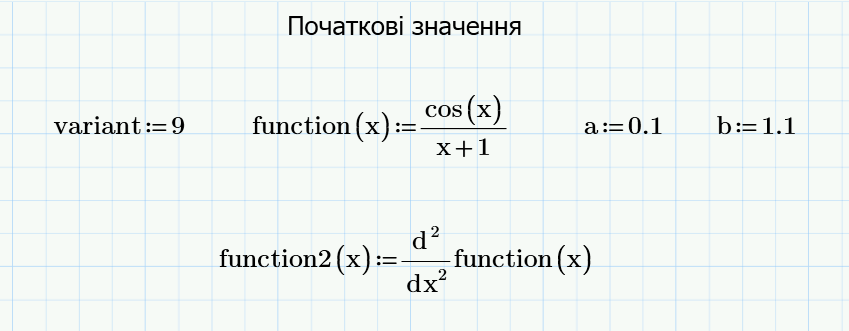
фазовий портрет системи u<2> ( u<1> ) , зробити висновки щодо стійкості системи.2 Розв’язок

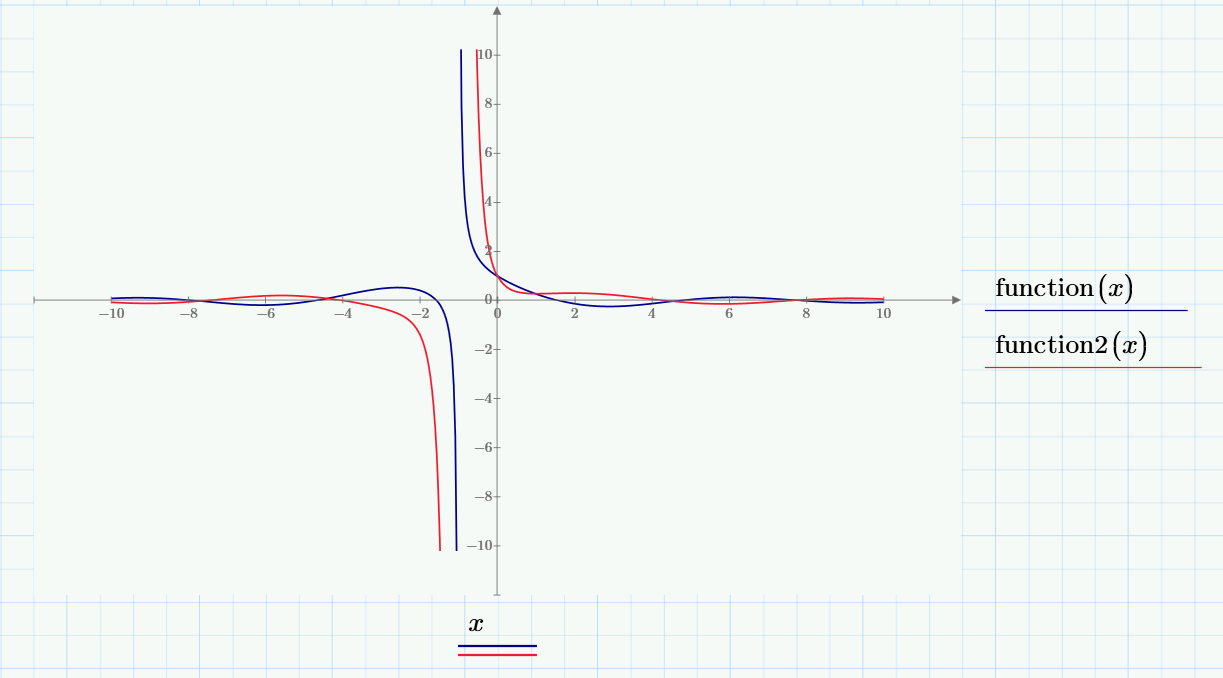
Вивід програми:

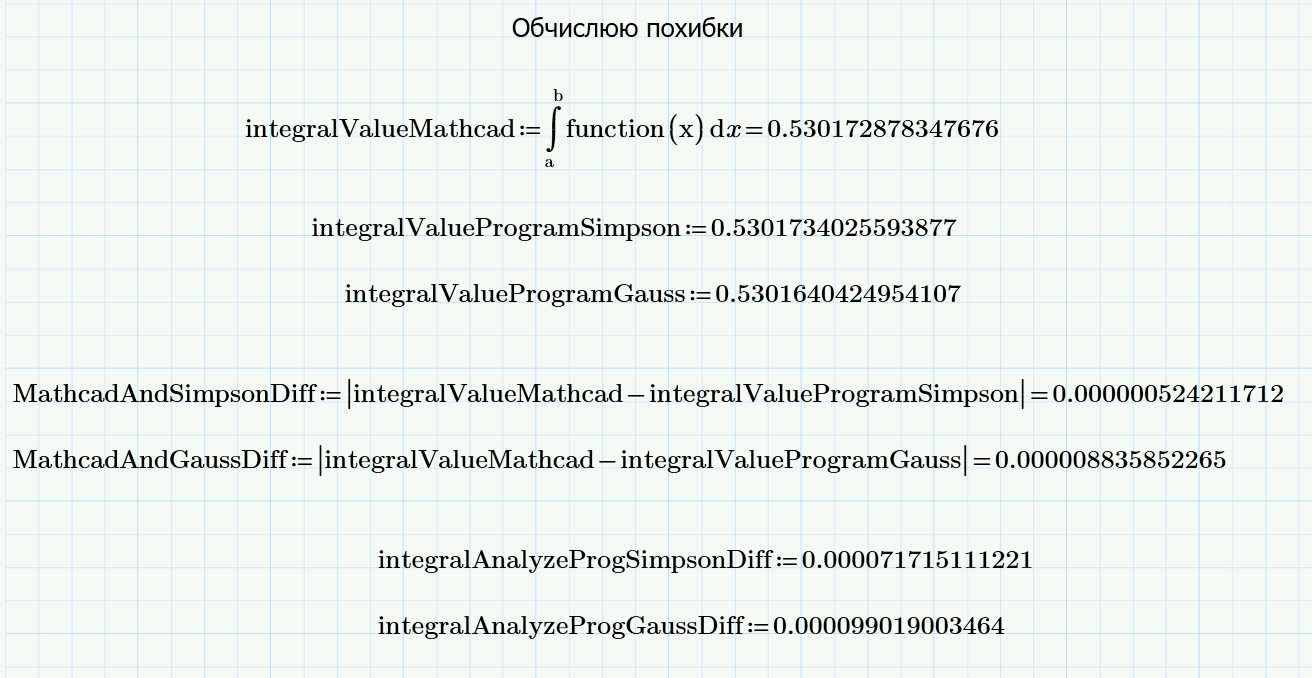


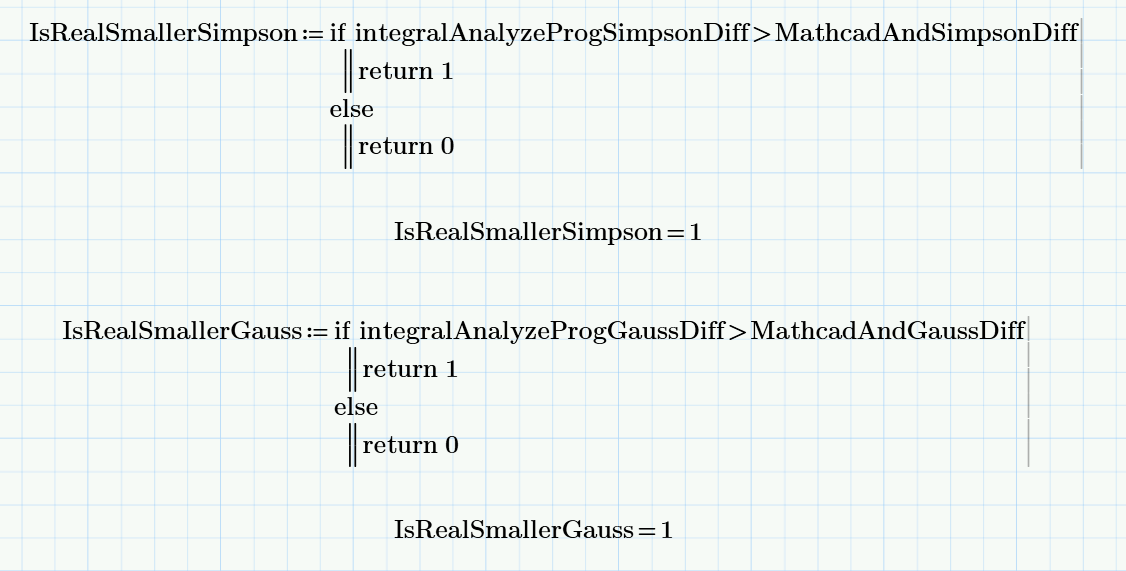
### 3 Розв’язок у Mathcad

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

**Lab7.py**

# region Starting Values

**import** numpy

**import** scipy

numpy**.**set\_printoptions**(**suppress**=True,**

formatter**={**'float\_kind'**:**'{:0.2f}'**.format})**

**import** scipy**.**optimize

**from** math **import** factorial**,** sin**,** cos

epsilonValue **=** 0.0001

leftBoard **=** 0.1

rightBoard **=** 1.1

defaultNforSimpson **=** 1

defaultNforGauss **=** 2

rounding **=** 5

valuesOfCoeffcients **=** **[**

**[** 0.5**,** 2**],**

**[-**0.577350**,** 0.577350**,** 1**,** 1**],**

**[-**0.774597**,** 0**,** 0.774597**,** 0.555555**,** 0.888889**,** 0.555555**],**

**[-**0.861136**,** **-**0.339981**,** 0.339981**,** 0.861136**,** 0.347855**,** 0.652145**,** 0.652145**,** 0.347855**]**

**]**

# endregion Starting Values

#region Default Functions

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

func **=** cos**(**x**)** **/** **(**x **+** 1**)**

**return** func

**def** ReverseMyFunction**(**t**):**

x **=** **((**t **\*** **(**rightBoard **-** leftBoard**))** **/** 2**)** **+** **((**leftBoard **+** rightBoard**)** **/** 2**)**

func **=** MyFunction**(**x**)**

**return** func

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

func **=** **(-**sin**(**x**)/(**x**+**1**))-(**cos**(**x**)/(**x**+**1**)\*\***2**)**

**return** func

**def** MyFourthPrimeFunction**(**x**):**

func **=** **(**cos**(**x**)-(**4**\***sin**(**x**)/(**x**+**1**))-(**12**\***cos**(**x**)/(**x**+**1**)\*\***2**)+(**24**\***sin**(**x**)/(**x**+**1**)\*\***3**)+(**24**\***cos**(**x**)/(**x**+**1**)\*\***4**))/(**x**+**1**)**

**return** func

**def** MySixthPrimeFunction**(**x**):**

func**=(-**cos**(**x**)+(**6**\***sin**(**x**)/(**x**+**1**))+(**30**\***cos**(**x**)/(**x**+**1**)\*\***2**)-(**120**\***sin**(**x**)/(**x**+**1**)\*\***3**)-(**360**\***cos**(**x**)/(**x**+**1**)\*\***4**)+(**720**\***sin**(**x**)/(**x**+**1**)\*\***5**)+(**720**\***cos**(**x**)/(**x**+**1**)\*\***6**))/(**x**+**1**)**

**return** func

**def** MyEigthPrimeFunction**(**x**):**

func**=(**cos**(**x**)-(**8**\***sin**(**x**)/(**x**+**1**))-(**56**\***cos**(**x**)/(**x**+**1**)\*\***2**)+(**336**\***sin**(**x**)/(**x**+**1**)\*\***3**)+(**1680**\***cos**(**x**)/(**x**+**1**)\*\***4**)-(**6720**\***sin**(**x**)/(**x**+**1**)\*\***5**)-(**20160**\***cos**(**x**)/(**x**+**1**)\*\***6**)+**

**(**40320**\***sin**(**x**)/(**x**+**1**)\*\***7**)+(**40320**\***cos**(**x**)/(**x**+**1**)\*\***8**))/(**x**+**1**)**

**return** func

**def** MyTenthPrimeFunction**(**x**):**

func**=(-**cos**(**x**)+(**10**\***sin**(**x**)/(**x**+**1**))+(**90**\***cos**(**x**)/(**x**+**1**)\*\***2**)-(**720**\***sin**(**x**)/(**x**+**1**)\*\***3**)-(**5040**\***cos**(**x**)/(**x**+**1**)\*\***4**)+(**30240**\***sin**(**x**)/(**x**+**1**)\*\***5**)+(**151200**\***cos**(**x**)/(**x**+**1**)\*\***6**)**

**-(**604800**\***sin**(**x**)/(**x**+**1**)\*\***7**)-(**1814400**\***cos**(**x**)/(**x**+**1**)\*\***8**)+(**3628800**\***sin**(**x**)/(**x**+**1**)\*\***9**)+(**3628800**\***cos**(**x**)/(**x**+**1**)\*\***10**))/(**x**+**1**)**

**return** func

**def** MyTwelwethPrimeFunction**(**x**):**

func**=(**cos**(**x**)-(**12**\***sin**(**x**)/(**x**+**1**))-(**132**\***cos**(**x**)/(**x**+**1**)\*\***2**)+(**1320**\***sin**(**x**)/(**x**+**1**)\*\***3**)+(**1180**\***cos**(**x**)/(**x**+**1**)\*\***4**)-(**95040**\***sin**(**x**)/(**x**+**1**)\*\***5**)**

**-(**665280**\***cos**(**x**)/(**x**+**1**)\*\***6**)+(**3991680**\***sin**(**x**)/(**x**+**1**)\*\***7**)+(**19958400**\***cos**(**x**)/(**x**+**1**)\*\***8**)-(**79833600**\***sin**(**x**)/(**x**+**1**)\*\***9**)-(**239500800**\***cos**(**x**)/**

**(**x**+**1**)\*\***10**)+(**479001600**\***sin**(**x**)/(**x**+**1**)\*\***11**)+(**479001600**\***cos**(**x**)/(**x**+**1**)\*\***12**))/(**x**+**1**)**

**return** func

SpecificFunctionsForTheGauss **=** **[**MyFourthPrimeFunction**,** MySixthPrimeFunction**,** MyEigthPrimeFunction**,**

MyTenthPrimeFunction**,** MyTwelwethPrimeFunction**]**

#endregion Default Functions

**def** Simpson**(**firstInterval**,** secondInterval**):**

tempValues **=** **[**0**,** 0**]**

totalAmount **=** MyFunction**(**secondInterval**)** **+** MyFunction**(**firstInterval**)**

analyzeDifference**,** Nvalue **=** SimpsonDifference**(**firstInterval**,** secondInterval**,** defaultNforSimpson**)**

intervalLength **=** **(**secondInterval **-** firstInterval**)** **/** **(**Nvalue **\*** 2**)**

**for** z **in** **range(**1**,** Nvalue **+** 1**):**

tempValues**[**1**]** **=** tempValues**[**1**]** **+** 4 **\*** MyFunction**(**intervalLength **\*** **(**z **\*** 2 **-** 1**)** **+** firstInterval**)**

totalAmount **=** totalAmount **+** tempValues**[**1**]**

**for** z **in** **range(**1**,** Nvalue**):**

tempValues**[**0**]** **=** tempValues**[**0**]** **+** 2 **\*** MyFunction**(**z **\*** 2 **\*** intervalLength **+** firstInterval**)**

totalAmount **=** totalAmount **+** tempValues**[**0**]**

finalResult **=** totalAmount **\*** intervalLength **/** 3

**return** finalResult**,** analyzeDifference**,** Nvalue

**def** SimpsonDifference**(**firstInterval**,** secondInterval**,** Nvalue**):**

valueForAccuracy **=** scipy**.**optimize**.**fmin\_l\_bfgs\_b**(lambda** x**:** **-**MyFourthPrimeFunction**(**x**),**

1.0**,** bounds**=[(**firstInterval**,** secondInterval**)],** approx\_grad**=True)**

difference **=** **(((**secondInterval **-** firstInterval**)** **\*\*** 5**)** **\*** **abs(**valueForAccuracy**[**1**][**0**]))** **/** **((**Nvalue **\*\*** 4**)** **\*** 180**)**

**while** difference **>** epsilonValue**:**

difference **=** **(((**secondInterval **-** firstInterval**)** **\*\*** 5**)** **\*** **abs(**valueForAccuracy**[**1**][**0**]))** **/** **((**Nvalue **\*\*** 4**)** **\*** 180**)**

Nvalue **=** Nvalue **+** 1

**return** difference**,** Nvalue

**def** Gauss**(**firstInterval**,** secondInterval**):**

analyzeDifference**,** Nvalue **=** GaussDifference**(**firstInterval**,** secondInterval**,** defaultNforGauss**)**

tempResult **=** 0

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

tempIndex **=** **int(((len(**valuesOfCoeffcients**[**Nvalue**])/**2**)+**z**)-**1**)**

tempResult **=** tempResult **+** valuesOfCoeffcients**[**Nvalue**-**1**][**tempIndex**]** **\*** ReverseMyFunction**(**valuesOfCoeffcients**[**Nvalue**-**1**][**z**])**

finalResult **=** **((**secondInterval **-** firstInterval**)** **/** 2**)** **\*** tempResult

**return** finalResult**,** analyzeDifference**,** Nvalue

**def** GaussDifference**(**firstInterval**,** secondInterval**,** Nvalue**):**

**for** z **in** **range(len(**SpecificFunctionsForTheGauss**)):**

valueForAccuracy **=** scipy**.**optimize**.**fmin\_l\_bfgs\_b**(lambda** x**:** **-**SpecificFunctionsForTheGauss**[**z**](**x**),**

1.0**,** bounds**=[(**firstInterval**,** secondInterval**)],** approx\_grad**=True)**

difference **=** **(((**secondInterval**-**firstInterval**)\*\*((**Nvalue**+**1**)\***2**))\*((**factorial**(**Nvalue**))\*\***4**))\*abs(**valueForAccuracy**[**1**][**0**])/(((**factorial**(**2**\***Nvalue**))\*\***3**)\*(**2**\***Nvalue**+**1**))**

**if** difference **<** epsilonValue**:**

**break**

Nvalue **=** Nvalue **+** 1

**return** difference**,** Nvalue

**print(**"Simpson Method:"**)**

simpsonResult**,** simpsonDiff**,** simpsonN **=** Simpson**(**leftBoard**,** rightBoard**)**

**print(**"Result ="**,** simpsonResult**,** "\n"**,** "Difference ="**,** "%-.15f"**%(**simpsonDiff**),** "\n"**,** "N ="**,** simpsonN**,** "\n"**)**

**print(**"Gauss Method:"**)**

gaussResult**,** gaussDiff**,** gaussN **=** Gauss**(**leftBoard**,** rightBoard**)**

**print(**"Result ="**,** gaussResult**,** "\n"**,** "Difference ="**,** "%-.15f"**%(**gaussDiff**),** "\n"**,** "N ="**,** gaussN**,** "\n"**)**

### Висновок:

Я навчився використовувати різні методи чисельного інтегрування функцій (методи трапецій, Сімпсона та Гауса. Також можна дійти до висновку, метод Сімпсона – найскладніший серед формул Ньютона – Котеса, але в той же час має меншу к-сть ітерацій та одну з найменших похибок. Також є метод Гауса – найскладніший у лабораторній, оскільки треба мати додаткові дані, але найточніший