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](#_Toc72824829)

[1 Постановка задачі 3](#_Toc72824830)

[2 Розв’язок 4](#_Toc72824831)

[3 Розв’язок у Mathcad 6](#_Toc72824832)

[4 Лістинг програми 9](#_Toc72824833)

[Висновок: 11](#_Toc72824834)

### 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

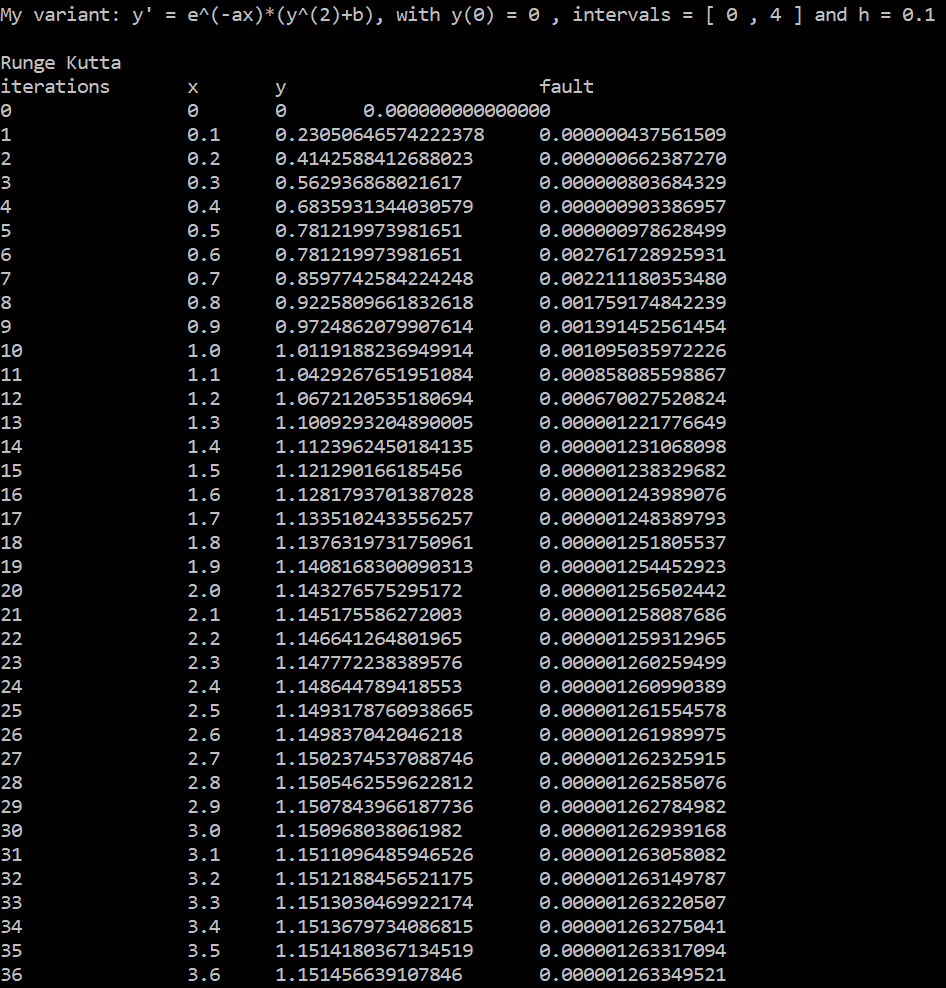
результатами.

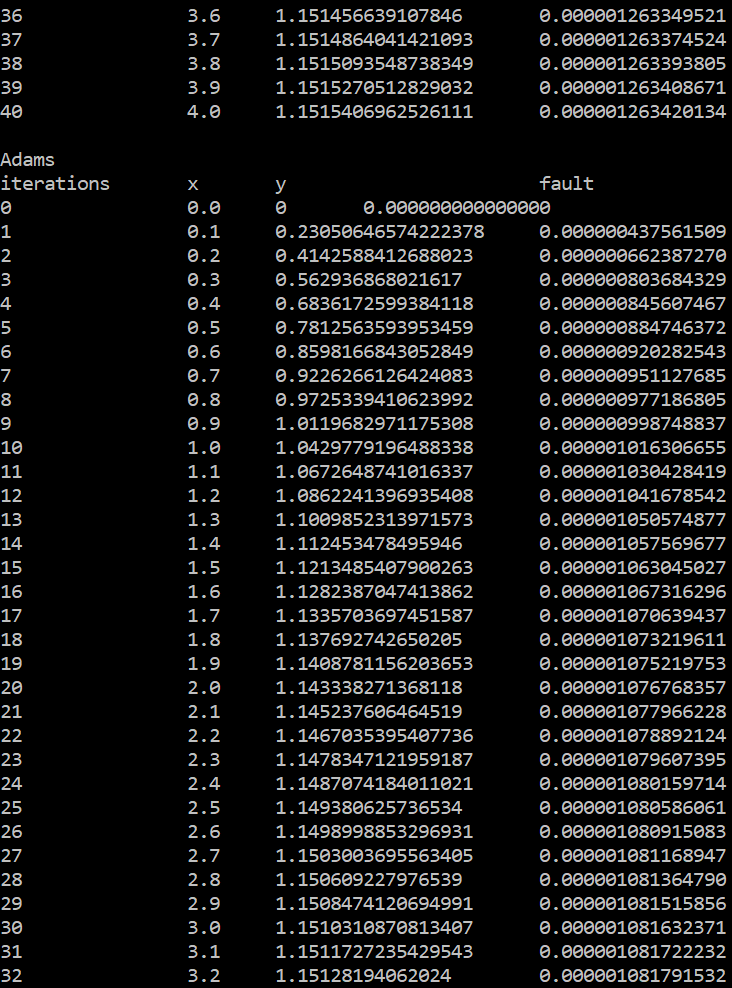
Розв’язати за допомогою Matchad систему рівнянь, побудувати графік 0 y та

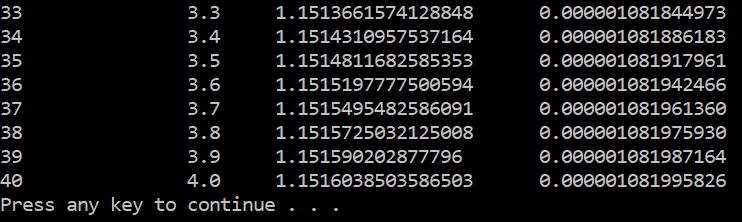
фазовий портрет системи u<2> ( u<1> ) , зробити висновки щодо стійкості системи.

### 2 Розв’язок

Вивід програми:

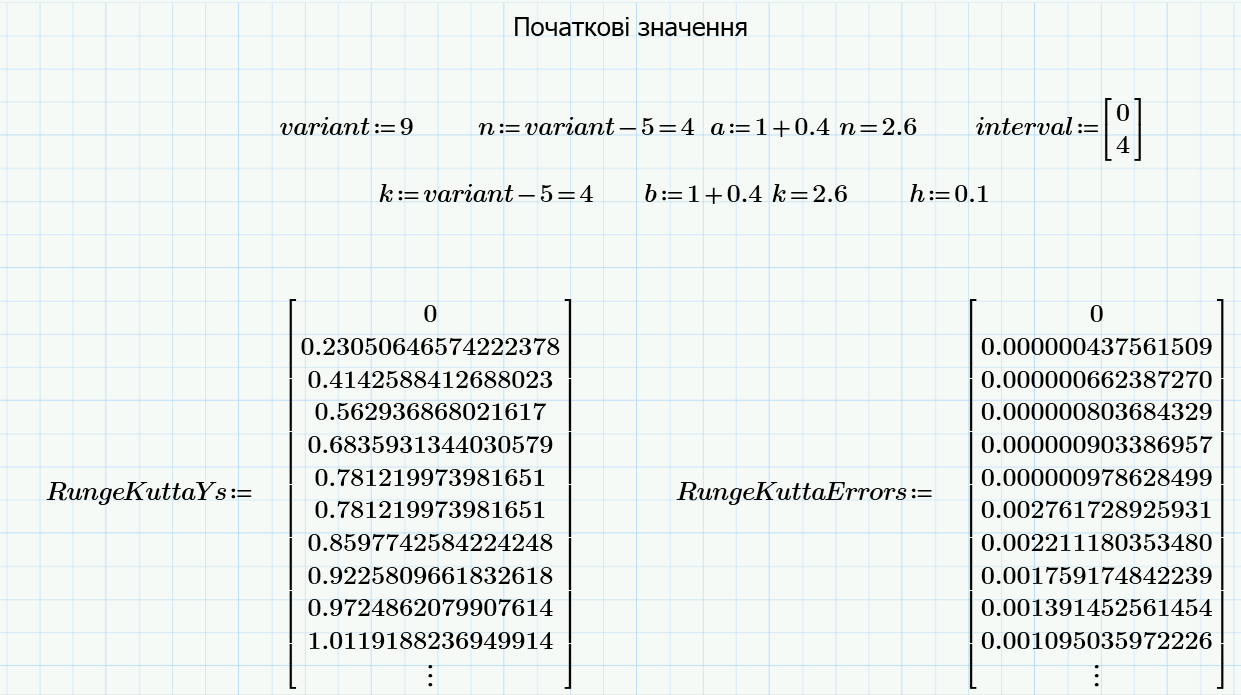


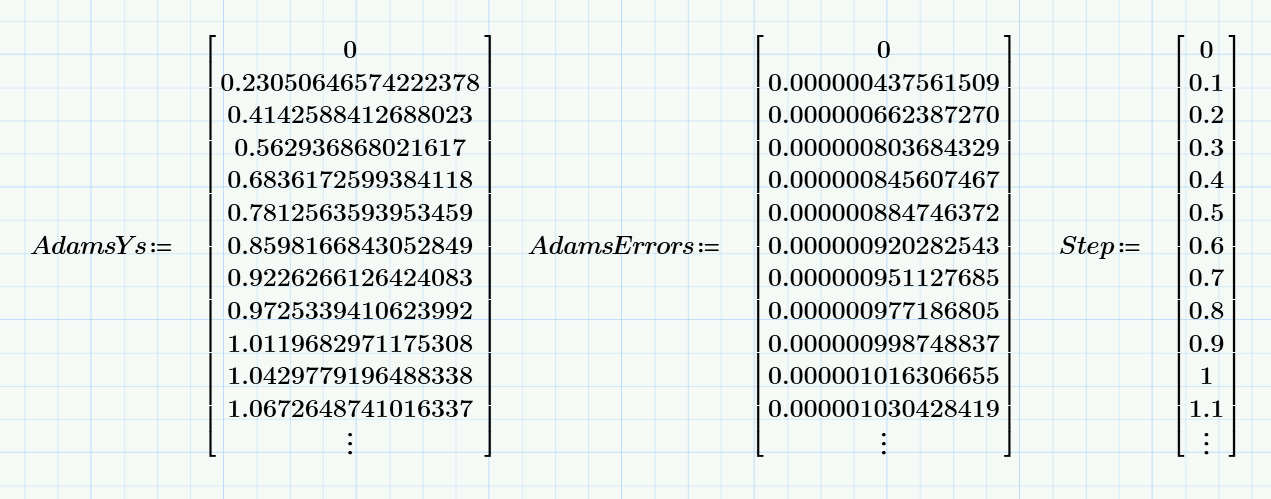


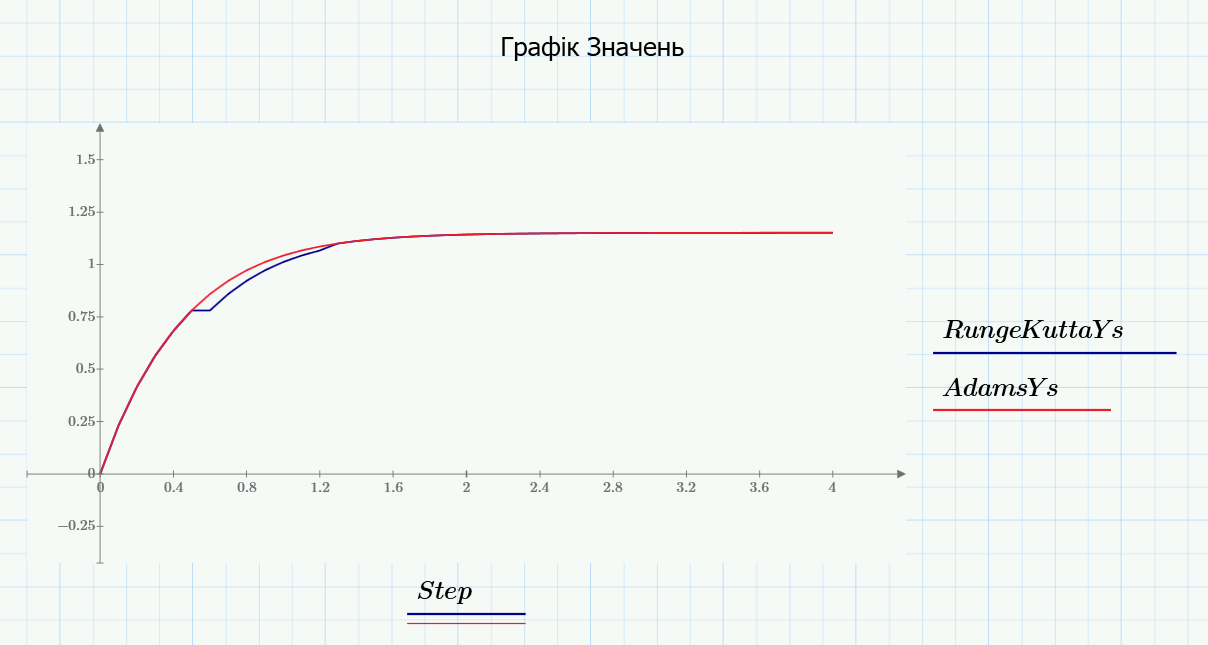


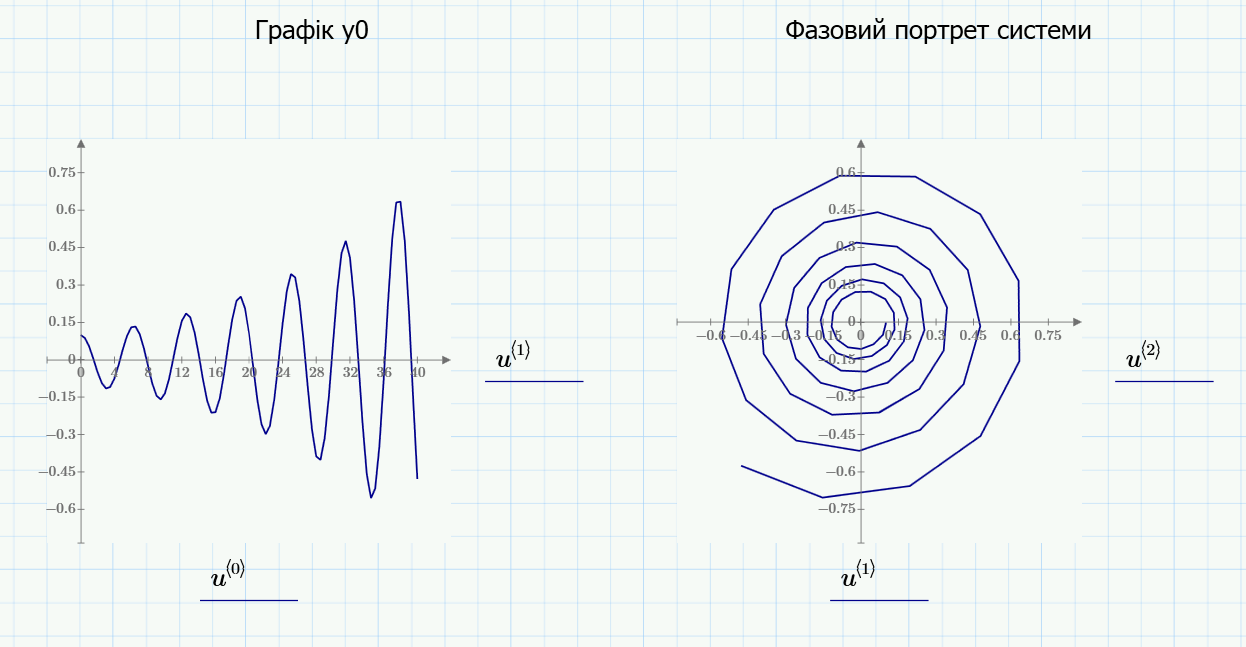
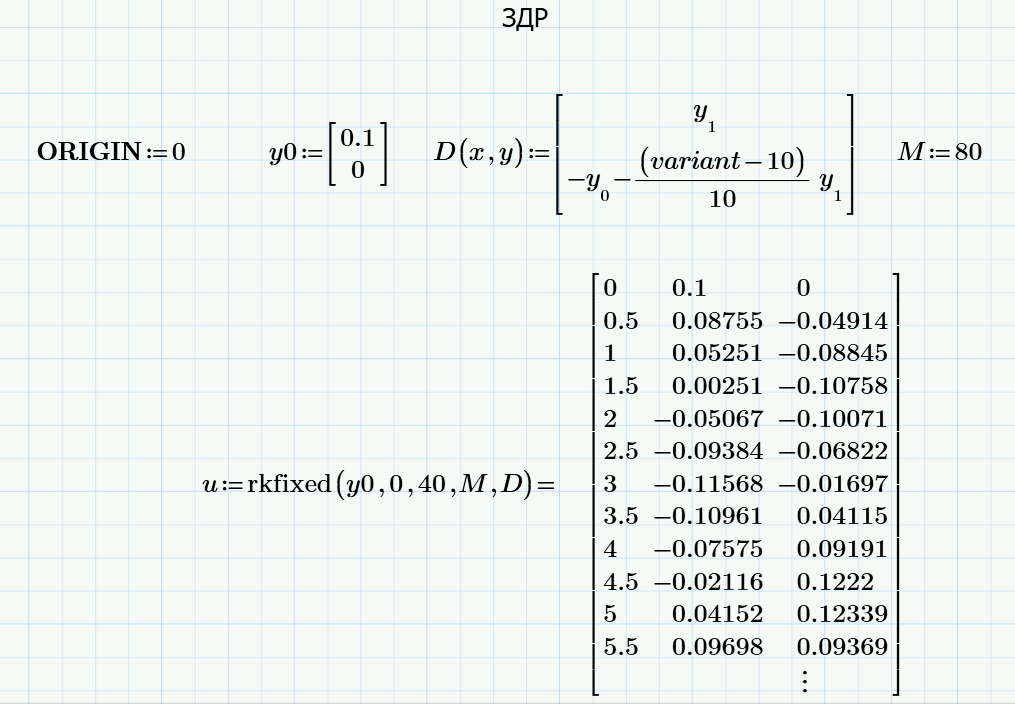
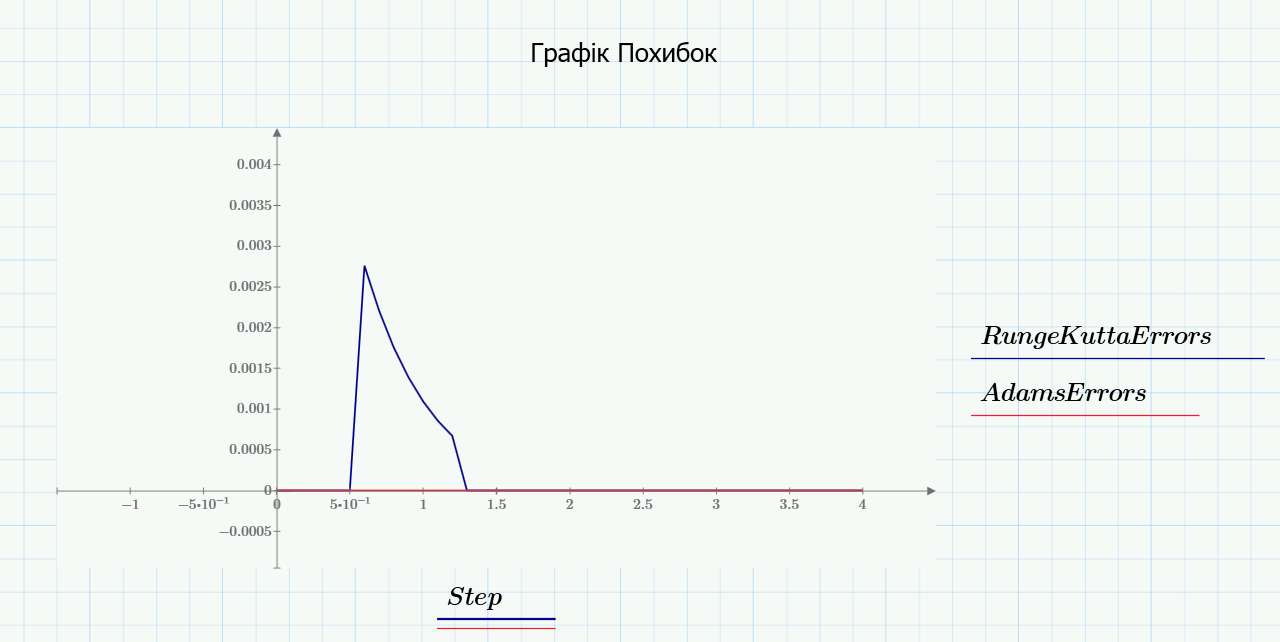
### 3 Розв’язок у Mathcad

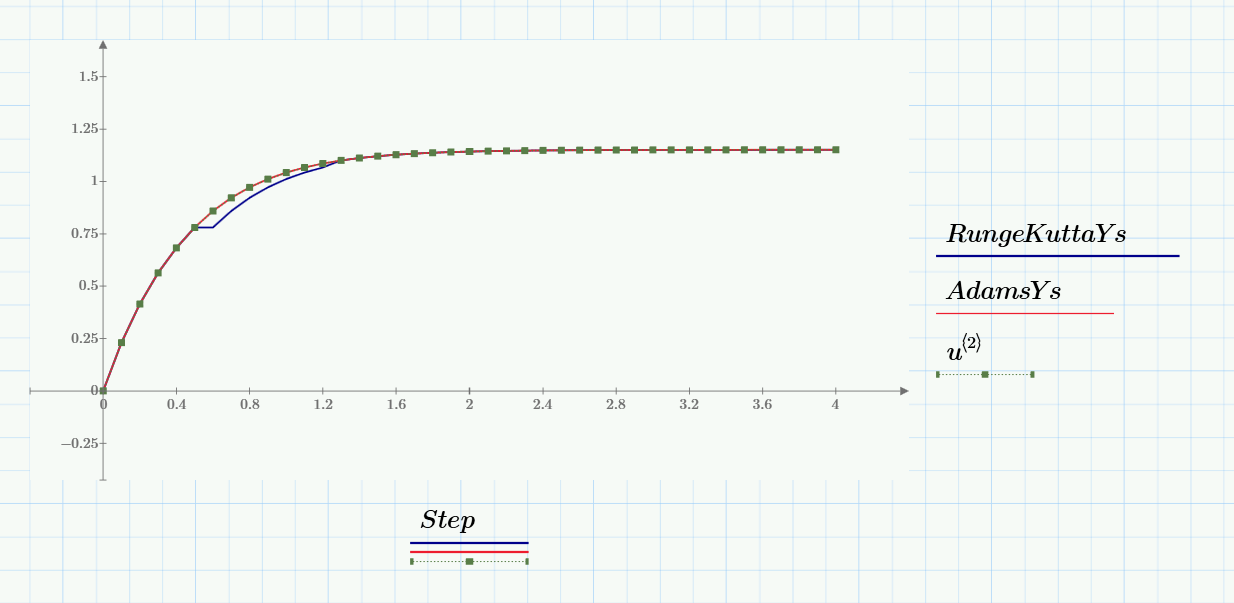
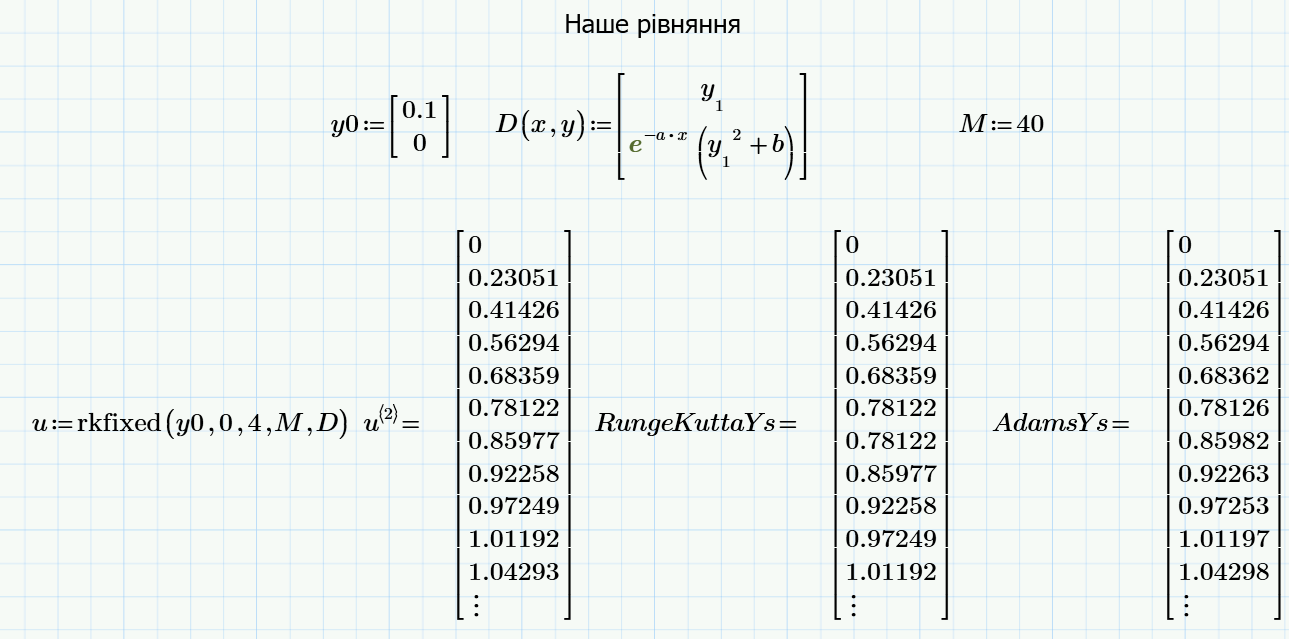
Нижче наведено розв’язок у Mathcad











У технічних розрахунках точність вимірювань характеризують відносною похибкою. Результат вважають гарним, якщо відносна похибка не перевищує 0,1 %. Отже Наш результат є гарним. Також саме у цій лабораторній є умова, що τ не повинно перевищувати декількох сотих, інакше крок потрібно зменшити, яка дотримується

### 4 Лістинг програми

**Lab8.py**

# region Starting Values

**import** math

epsilonValue **=** 0.1

variant **=** 9

N **=** K **=** variant **-** 5

A **=** B **=** 1 **+** 0.4 **\*** N

h **=** 0.1

rungeConstants **=** **[**0.5**,** 1**,** 6**,** 2**]**

adamsConstants **=** **[**24**,** 55**,** 59**,** 37**,** 9**,** 19**,** 5**]**

leftBorder **=** 0

rightBorder **=** 4

yZero **=** 0

yFirstAdams **=** **[]**

ySecondAdams **=** **[]**

square **=** half **=** 2

header **=** "iterations\t x\t y\t\t\t fault"

rounding **=** 1

rungeValue **=** 15

adamsStartValue **=** 3

numOfIter **=** 41

# endregion Starting Values

# region Prime Function

**def** MyPrimeFunction**(**X**,** Y**):**

result **=** **(**Y **\*\*** square **+** B**)** **\*** math**.**e **\*\*** **(-**A **\*** X**)**

**return** result

# endregion Prime Function

# region Runge Kutta

**def** RungeKutta**(**xInnerZero**,** yInnerZero**,** x**,** h**):**

y **=** yInnerZero

countToIterate **=** **(int)(**x **/** h**)**

**for** i **in** **range(**0**,** countToIterate**):**

K1 **=** h **\*** MyPrimeFunction**(**xInnerZero**,** y**)**

K2 **=** h **\*** MyPrimeFunction**(**rungeConstants**[**0**]** **\*** h **+** xInnerZero**,** rungeConstants**[**0**]** **\*** K1 **+** y**)**

K3 **=** h **\*** MyPrimeFunction**(**rungeConstants**[**0**]** **\*** h **+** xInnerZero**,** rungeConstants**[**0**]** **\*** K2 **+** y**)**

K4 **=** h **\*** MyPrimeFunction**(**h **+** xInnerZero**,** K3 **+** y**)**

y **=** **(**K1 **+** rungeConstants**[**3**]** **\*** K2 **+** rungeConstants**[**3**]** **\*** K3 **+** K4**)** **\*** **(**rungeConstants**[**1**]** **/** rungeConstants**[**2**])** **+** y

errorValue **=** **abs((**K2 **-** K3**)** **/** **(**K1 **-** K2**))**

**if** epsilonValue **<** errorValue **:**

h **=** h **/** half

xInnerZero **=** xInnerZero **+** h

**return** y

**def** RungeKuttaFull**():**

tempValueForLeftBorder **=** leftBorder

iterations **=** 0

yFirstRunge **=** **[]**

ySecondRunge **=** **[]**

**print(**header**)**

# do the Runge's rule

**while** tempValueForLeftBorder **<=** rightBorder **+** 0.1**:**

tempOne **=** RungeKutta**(**leftBorder**,** yZero**,** tempValueForLeftBorder**,** h**)**

yFirstRunge**.**append**(**tempOne**)**

tempTwo **=** RungeKutta**(**leftBorder**,** yZero**,** tempValueForLeftBorder**,** h **/** half**)**

ySecondRunge**.**append**(**tempTwo**)**

faultValue **=** **abs((**yFirstRunge**[**iterations**]** **-** ySecondRunge**[**iterations**])** **/** **(**rungeValue**))**

**print(**iterations**,** "\t\t"**,** **round(**tempValueForLeftBorder**,** rounding**),** "\t"**,** tempOne**,** "\t"**,** "%-.15f"**%(**faultValue**))**

tempValueForLeftBorder **=** tempValueForLeftBorder **+** 0.1

# For Adams Method

**if** iterations **<=** adamsStartValue**:**

yFirstAdams**.**append**(**yFirstRunge**[**iterations**])**

ySecondAdams**.**append**(**ySecondRunge**[**iterations**])**

iterations **=** iterations **+** 1

# endregion Runge Kutta

# region Adams

**def** Adams**(**firstValuesFromRunge**,** h**):**

iterations **=** adamsStartValue

**while** iterations **<** **((**rightBorder **-** leftBorder**)** **/** h**)** **+** 4**:**

K4 **=** MyPrimeFunction**(**iterations **\*** h **-** 0.3**,** firstValuesFromRunge**[**iterations**-**3**])**

K3 **=** MyPrimeFunction**(**iterations **\*** h **-** 0.2**,** firstValuesFromRunge**[**iterations**-**2**])**

K2 **=** MyPrimeFunction**(**iterations **\*** h **-** 0.1**,** firstValuesFromRunge**[**iterations**-**1**])**

K1 **=** MyPrimeFunction**(**iterations **\*** h**,** firstValuesFromRunge**[**iterations**])**

firstAdditionalY **=** h **/** adamsConstants**[**0**]** **\*** **(**adamsConstants**[**1**]** **\*** K1 **-** adamsConstants**[**2**]** **\*** K2 **+** adamsConstants**[**3**]** **\*** K3 **-** adamsConstants**[**4**]** **\*** K4**)** **+** firstValuesFromRunge**[**iterations**]**

additionalX **=** h **+** h **\*** iterations

secondAdditionalY **=** firstValuesFromRunge**[**iterations**]** **+** h **/** adamsConstants**[**0**]** **\*** **(**adamsConstants**[**4**]** **\*** MyPrimeFunction**(**additionalX**,** firstAdditionalY**)** **+** adamsConstants**[**5**]** **\*** K1 **-** adamsConstants**[**6**]** **\*** K2 **+** K3**)**

errorValue **=** **abs(**firstAdditionalY **-** secondAdditionalY**)**

**if** epsilonValue **<** errorValue **:**

h **=** h **/** half

**if** firstAdditionalY **!=** secondAdditionalY**:**

firstValuesFromRunge**.**append**(**secondAdditionalY**)**

**else:**

firstValuesFromRunge**.**append**(**firstAdditionalY**)**

iterations **=** iterations **+** 1

**return** firstValuesFromRunge

**def** AdamsFull**():**

**print(**header**)**

yFirstAdamsErrors **=** Adams**(**yFirstAdams**,** h**)**

ySecondAdamsErrors **=** Adams**(**ySecondAdams**,** h**)**

# do the Runge's rule

**for** x **in** **range(**numOfIter**):**

faultValue **=** **abs((**yFirstAdamsErrors**[**x**]** **-** ySecondAdamsErrors**[**x**])** **/** **(**rungeValue**))**

**print(**x**,** "\t\t"**,** **round(**x **\*** 0.1**,** rounding**),** "\t"**,** yFirstAdamsErrors**[**x**],** "\t"**,** "%-.15f"**%(**faultValue**))**

# endregion Adams

**def** RunAll**():**

**print(**"My variant: y' = e^(-ax)\*(y^(2)+b), with y(0) ="**,** yZero**,** ", intervals = ["**,**leftBorder**,**","**,**rightBorder**,**"] and h ="**,** h**)**

**print(**"\nRunge Kutta"**)**

RungeKuttaFull**()**

**print(**"\nAdams"**)**

AdamsFull**()**

RunAll**()**

### Висновок:

Я навчився використовувати різні методи розв’язання задачі Коші (методи Рунге-Кута та Адамса). Метод Рунге-Кутта має різні зручні властивості, які стосуються обчислень, але має також один суттєвий недолік. При побудові цього методу застосовується інформація на відрізку прямої довжиною в один крок, тому подібна інформація має бути отримана знову, що передбачає велику трудоємкість відповідних обчислювальних правил. Якщо відмовитись від умови однокроковості, можна обчислювальні методи будувати таким чином, щоби частина отриманої інформації використовувалась повторно на декількох наступних кроках обчислювального процесу. Такі методи називаються багатокроковими. До них відноситься зокрема метода Адамса (Адамса-Башфорта).

З фазового портрету та графіку y0 можна зрозуміти, що Наша система має нестійкий фокус, а тип коренів – комплексні з додатною дійсною частиною