



# **GLOBAL ACADEMY OF TECHNOLOGY**

Approved by AICTE, New Delhi, Recognized by the Govt. of Karnataka  
Autonomous Institute affiliated to VTU, Belagavi, NAAC Accredited with 'A' Grade  
Ideal Homes Township, Rajarajeshwari Nagar, Bengaluru-98



## **Department of Mathematics**

# **Mathematics-2(Integrated Course)**

**II Semester**

**Course Code: BMATS24201**

**Version 1.0**

**w.e.f 10.03.2025**

**Editorial Committee**

**Mathematics-2 handling faculties**

**Dept. of Mathematics**

**Approved by**

**Dr.Rupa K**

**Prof. & Head,**

**Dept.of Mathematics, GAT**



# Global Academy of Technology

Approved by AICTE, New Delhi, Recognized by the Govt. of Karnataka  
Autonomous Institute affiliated to VTU, Belagavi, NAAC Accredited with 'A' Grade  
Ideal Homes Township, Rajarajeshwari Nagar, Bengaluru-98



## LABORATORY CERTIFICATE

This is to certify that Mr./Ms. \_\_\_\_\_

bearing USN\_\_\_\_\_ of the Department

\_\_\_\_\_ has satisfactorily completed the course

of **Mathematics-2** (Part of Integrated Course BMATS24101) prescribed

by Global Academy of Technology (Autonomous Institute, under VTU),

Bengaluru for the Academic Year 2024-2025.

Marks	
Maximum	Obtained
20	

Signature of the Faculty

Signature of the HOD

Date:

## Document Log

Name of the document	Python Programming Manual(Mathematics-2) and CIE Evaluation Book
Scheme / Course Type	2024/ Integrated Course
Current version number and date	Version 1.0 / 10.03.2025
Subject code	BMATS24201
Editorial Committee	Python Programming(Mathematics-2) handling faculties
Approved by	Prof & Head, Dept. of Mathematics

## Table of Contents

Sl. No	Particulars		Page No.
1	Vision and Mission of the Institute		1
2	Program Outcomes		2
3	Syllabus Course Outcomes CO-PO Mapping		3
4	Marks Distribution		7
5	Rubrics		8
6	Course Outcomes mapping with Bloom's Taxonomy		9
7	Lab Evaluation Sheet		10
8	<b>CHAPTER 1</b>	Introduction	11
9	<b>CHAPTER 2</b>	About the IDE	12
10	<b>CHAPTER 3</b>	Programs	
	<b>Lab 1</b>	Program to compute surface area, volume and centre of gravity	
	<b>Lab 2</b>	Evaluation of improper integrals	
	<b>Lab 3</b>	Finding gradient, divergent, curl and their geometrical interpretation	
	<b>Lab 4</b>	Computation of basis and dimension for a vector space and Graphical representation of linear transformation	
	<b>Lab 5</b>	Computing the inner product and orthogonality	
	<b>Lab 6</b>	Solution of algebraic and transcendental equations by Regula-Falsi and Newton-Raphson method	
	<b>Lab 7</b>	Interpolation/Extrapolation using Newton's forward and backward difference formula	
	<b>Lab 8</b>	Computation of area under the curve using Trapezoidal, Simpson's (1/3)rd and (3/8)th rule	
	<b>Lab 9</b>	Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method	
	<b>Lab 10</b>	Solution of ODE of first order and first degree by Runge-Kutta 4th order and Milne's predictor-corrector method	

11	<b>CHAPTER 6</b>	Viva Questions	57
----	------------------	----------------	----

## **Vision of the Institute**

Become a premier institution imparting quality education in engineering and management to meet the changing needs of society.

## **Mission of the Institute**

- M1: Create environment conducive for continuous learning through quality teaching and learning processes supported by modern infrastructure.
- M2: Promote research and innovation through collaboration with industries.
- M3: Inculcate ethical values and environmental conscious through holistic education programs.

## Program Outcomes (POs)

### Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## Syllabus

<b>Course Code</b>	<b>BMATS24201</b>	<b>CIE Marks</b>	<b>50</b>
<b>Hours/Week (L: T: P)</b>	<b>2:2:2</b>	<b>SEE Marks</b>	<b>50</b>
<b>No. of Credits</b>	<b>4</b>	<b>Examination Hours</b>	<b>03</b>

### **Course: Mathematics II for CSE Stream (Integrated)**

#### **Course Objectives**

To enable students to apply the knowledge of Mathematics in various fields of engineering by making them to learn the following:

<b>CLO1</b>	Multiple Integrals
<b>CLO2</b>	Vector Calculus
<b>CLO3</b>	Vector Space
<b>CLO4</b>	Numerical methods

<b>Content</b>	<b>No. of Hours/ RBT levels</b>
<b>Module 1</b> Multiple Integrals: Evaluation of double and triple integrals, evaluation of double integrals by change of order of integration, changing into polar coordinates. Problems. Beta and Gamma functions: Definitions, properties, relation between Beta and Gamma functions. Problems.	<b>08 Hours</b> <b>L3</b>
<b>Module 2</b> Scalar and vector fields. Gradient, directional derivative, curl and divergence - physical interpretation, solenoidal and irrotational vector fields. Problems. Curvilinear coordinates: Scale factors, base vectors, Cylindrical polar coordinates, Spherical polar coordinates, transformation between cartesian and curvilinear systems, orthogonality. Problems	<b>08 Hours</b> <b>L3</b>
<b>Module 3</b> Vector spaces: Definition and examples, subspace, linear span, Linearly independent and dependent sets, Basis and dimension. Problems. Linear transformations: Definition and examples, Algebra of transformations, Matrix of a linear transformation. Change of coordinates, Rank and nullity of a linear operator, rank-nullity theorem. Inner product spaces and orthogonality.	<b>08 Hours</b> <b>L3</b>
<b>Module 4</b> Solution of algebraic and transcendental equations: Regula-Falsi and Newton-Raphson methods (only formulae). Problems. Finite differences, Interpolation using Newton's forward and backward difference formulae, Newton's divided difference formula and Lagrange's interpolation formula (All formulae without proof). Problems. Numerical integration: Trapezoidal, Simpson's (1/3)rd and (3/8)th rules (without proof). Problems.	<b>08 Hours</b> <b>L3</b>
<b>Module 5</b> Numerical solution of ordinary differential equations of first order and first degree - Taylor's series method, Modified Euler's method, Runge-Kutta method of fourth order and Milne's predictor-corrector formula (No derivations of formulae). Problems	<b>08 Hours</b> <b>L3</b>

#### **Text books:**



1. Higher Engineering Mathematics, B.S. Grewal, Khanna Publishers, 44th Edition, 2017.
2. David C Lay: "Linear Algebra and its Applications", Pearson Publishers, 4th Ed., 2018.
3. B.V. Ramana, Higher Engineering Mathematics, Tata McGraw-Hill, 2006

### References:

1. E. Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons 10th Edition, 2016
2. N.P.Bali and Manish Goyal, A Textbook of Engineering Mathematics, Laxmi Publications 6th Edition, 2014
3. Gilbert Strang: "Linear Algebra and its Applications", Cengage Publications, 4th Ed., 2022.

### Lab Experiments:

1	Program to compute surface area, volume and centre of gravity
2	Evaluation of improper integrals
3	Finding gradient, divergent, curl and their geometrical interpretation
4	Computation of basis and dimension for a vector space and Graphical representation of linear transformation
5	Computing the inner product and orthogonality
6	Solution of algebraic and transcendental equations by Regula-Falsi and Newton-Raphson method
7	Interpolation/Extrapolation using Newton's forward and backward difference formula
8	Computation of area under the curve using Trapezoidal, Simpson's (1/3)rd and (3/8)th rule
9	Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method
10	Solution of ODE of first order and first degree by Runge-Kutta 4th order and Milne's predictor-corrector method

### Course Outcomes

Upon completion of this course, student will be able to:

<b>CO1</b>	Evaluate double integrals using direct integration, by changing the order of integration and by changing the variables Evaluate triple integrals by iterating the integration with respect to each variable Evaluate integrals using Beta and Gamma functions
<b>CO2</b>	Understand the concept of the gradient of a scalar field and its geometric interpretation Compute the directional derivative of a scalar field, divergence and curl of a vector field. Understand the properties of orthogonal coordinate systems, including scale factors and base vectors Convert vector and scalar fields between Cartesian and cylindrical/spherical coordinate systems
<b>CO3</b>	Determine if a set of vectors forms a subspace of a given vector space Determine if a set of vectors is linearly independent and Construct bases for vector spaces and subspaces. Represent vectors in different coordinate systems and understand the change of basis. Compute inner products, norms, and distances between vectors in an inner product space.
<b>CO4</b>	Solve algebraic and transcendental equations Apply numerical techniques for interpolation of data Evaluate definite integrals using numerical techniques
<b>CO5</b>	<ul style="list-style-type: none"> <li>• Solve ordinary differential equations of first order using numerical techniques.</li> </ul>

### Scheme of Examination:

**Semester End Examination (SEE):**

**SEE Question paper is to be set for 100 marks and the marks scored will be proportionately reduced to 50.** There will be two full questions (with a maximum of three sub questions) from each module carrying 20 marks each. Students are required to answer any **five full questions** choosing at least **one full question from each module**.

**Continuous Internal Evaluation (CIE):**

Component		Marks	Total Marks
CIE	CIE Test-1	30	(Average of 3 CIE + Lab) <b>50</b>
	CIE Test-2	30	
	CIE Test-3	30	
	Lab Record + CIE	20	
SEE	Semester End Examination	100	50
<b>Grand Total</b>			<b>100</b>

<b>CO/PO Mapping</b>				
<b>CO/PO</b>	PO1	PO2	PO3	PO12
<b>CO1</b>	3	2	1	3
<b>CO2</b>	3	2	1	3
<b>CO3</b>	3	2	1	3
<b>CO4</b>	3	2	1	3
<b>CO5</b>	3	2	1	3
<b>Average</b>	3	2	1	3

**Low-1: Medium-2: High-3**

## Python Programming Rubrics

### Rubrics for Evaluation of CIE Evaluation Book

Attribute	Max. Marks	Good	Satisfactory	Poor
		02	01	01
<b>Write-up</b>	<b>02</b>	<ul style="list-style-type: none"> <li>Complete program without errors written.</li> <li>Input and expected output for all test cases.</li> </ul>	<ul style="list-style-type: none"> <li>Complete program without errors written.</li> <li>Input and expected output not for all test cases.</li> </ul>	<ul style="list-style-type: none"> <li>Incomplete code.</li> <li>Indentation missing.</li> <li>All test cases not covered.</li> </ul>
Attribute	Max. Marks	05-06	03-04	00-02
<b>Execution</b>	<b>06</b>	<ul style="list-style-type: none"> <li>Debugs the program independently.</li> <li>Executed the program for all possible inputs.</li> <li>Record of output properly in observation book</li> </ul>	<ul style="list-style-type: none"> <li>Works with little help from faculty.</li> <li>All possible input cases not covered.</li> <li>Incomplete record of output in observation book</li> </ul>	<ul style="list-style-type: none"> <li>Unable to complete the execution of the program within the lab session.</li> </ul>
Attribute	Max. Marks	2	1	0
<b>Viva Voce</b>	<b>02</b>	<ul style="list-style-type: none"> <li>Able to explain the logic of the program.</li> <li>Answered all questions.</li> </ul>	<ul style="list-style-type: none"> <li>Partially understood the logic of the program.</li> <li>Answered few questions.</li> </ul>	<ul style="list-style-type: none"> <li>Not understood the logic of the program.</li> <li>Not answering any questions.</li> </ul>

### Rubrics for Evaluation of Internal Assessment Test

Attribute	Max. Marks	Good	Satisfactory	Poor
		07-10	03-06	00-03
<b>Write-up</b>	<b>10</b>	<ul style="list-style-type: none"> <li>Complete program without errors written.</li> <li>Input and expected output for all test cases.</li> </ul>	<ul style="list-style-type: none"> <li>Complete program without errors written.</li> <li>Input and expected output not for all test cases.</li> </ul>	<ul style="list-style-type: none"> <li>Incomplete code.</li> <li>Indentation missing.</li> <li>All test cases not covered.</li> </ul>
Attribute	Max. Marks	20-30	10-19	00-09
<b>Execution</b>	<b>30</b>	<ul style="list-style-type: none"> <li>Debugs the program independently.</li> <li>Executed the program</li> </ul>	<ul style="list-style-type: none"> <li>Works with little help from faculty.</li> <li>All possible input cases</li> </ul>	<ul style="list-style-type: none"> <li>Unable to complete the execution of the program within the</li> </ul>

		for all possible inputs. • Record of output properly in observation book	not covered. • Incomplete record of output in observation book	lab session.
<b>Attribute</b>	<b>Max. Marks</b>	<b>07-10</b>	<b>03-06</b>	<b>00-03</b>
<b>Viva Voce</b>	<b>10</b>	Able to explain the logic of the program. • Answered all questions.	• Partially understood the logic of the program. • Answered few questions.	• Not understood the logic of the program. • Not answering any questions.

### Course Outcomes mapping with Bloom's Taxonomy

<b>Sl. No.</b>	<b>Program Name</b>	<b>CO Mapping</b>	<b>RBT Mapping</b>
1.	Program to compute surface area, volume and centre of gravity	CO1	L3
2.	Evaluation of improper integrals	CO1	L3
3.	Finding gradient, divergent, curl and their geometrical interpretation	CO2	L3
4.	Computation of basis and dimension for a vector space and Graphical representation of linear transformation	CO2	L3
5.	Computing the inner product and orthogonality	CO3	L3
6.	Solution of algebraic and transcendental equations by Regula-Falsi and Newton-Raphson method	CO4	L3
7.	Interpolation/Extrapolation using Newton's forward and backward difference formula	CO5	L3
8.	Computation of area under the curve using Trapezoidal, Simpson's (1/3)rd and (3/8)th rule	CO5	L3

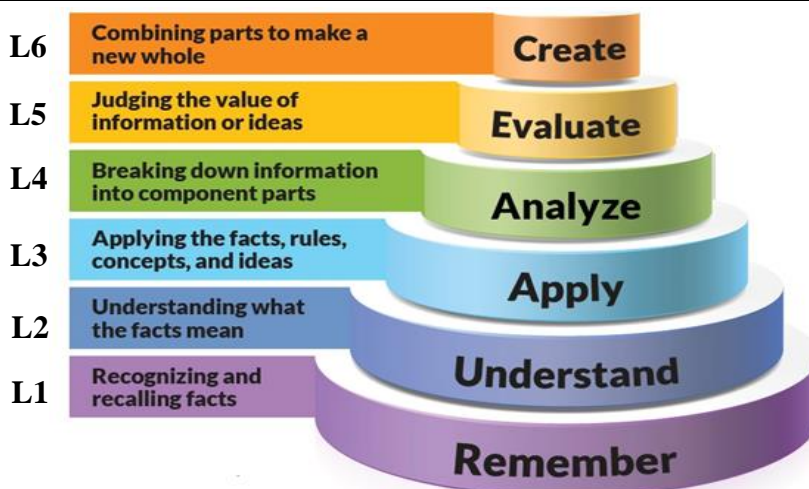


Figure: Blooms Taxonomy Hierarchy

## Evaluation Sheet

Sl. No.	Date	Particulars	CIE Evaluation Book (A)(10 M)			
			Write Up 02 M	Execution 06 M	Viva 02 M	Total 10 M
1		Program to compute surface area, volume and centre of gravity				
2		Evaluation of improper integrals				
3		Finding gradient, divergent, curl and their geometrical interpretation				
4		Computation of basis and dimension for a vector space and Graphical representation of lines and planes				
5		Computing the inner product and orthogonality				
6		Solution of algebraic and transcendental equations by Regula-Falsi and Newton-Raphson method				
7		Interpolation/Extrapolation using Newton's forward and backward difference formula				
8		Computation of area under the curve using Trapezoidal, Simpson's (1/3)rd and (3/8)th rule				
9		Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method				
10		Solution of ODE of first order and first degree by Runge-Kutta 4th order and Milne's predictor-corrector method				
Average Marks of CIE Evaluation Book (A)			<b>Out of 10</b>			
<b>Internal Assessment Marks (B)</b>						
Program no	Change of program (if taken)	Write Up 02 M	Execution 06 M	Viva 02 M	Total 50 M	Reduced to 10 M
<b>Final CIE Marks (A+B)</b>				<b>Out of 20</b>		

- 1.
- 2.

## CHAPTER 1

### INTRODUCTION

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

#### How to Start with Python Programming:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like <https://www.onlinegdb.com/>, <http://ideone.com/> or <http://codepad.org/> that can be used to run Python programs without installing an interpreter.

**Windows:** There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from <http://python.org/>.

**Linux:** Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check which version of Python you're running, type "python" in the terminal emulator. The interpreter should start and print the version number.

**MacOS:** Generally, Python 2.7 comes bundled with macOS. You'll have to manually install Python 3 from <http://python.org/>.

#### Why use Python?

Python, as a high-level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks. The simple syntax rules of the programming language further make it easier for you to keep the code base readable and application maintainable.

1. Web and Internet Development
2. Desktop GUI Applications
3. Scientific and Numeric

4. Software Development
5. Games and 3D Graphics
6. Database Access
7. Games and 3D Graphics

## CHAPTER 2

### ABOUT THE IDE

Jupyter notebook is the most popular IDE used for Python scripting language.

Jupyter notebook offers some of the best features to its users and developers in the following aspects:

- Code completion and inspection
- Advanced debugging
- Support for web programming and frameworks such as Django and Flask

### Jupyter notebook Installation

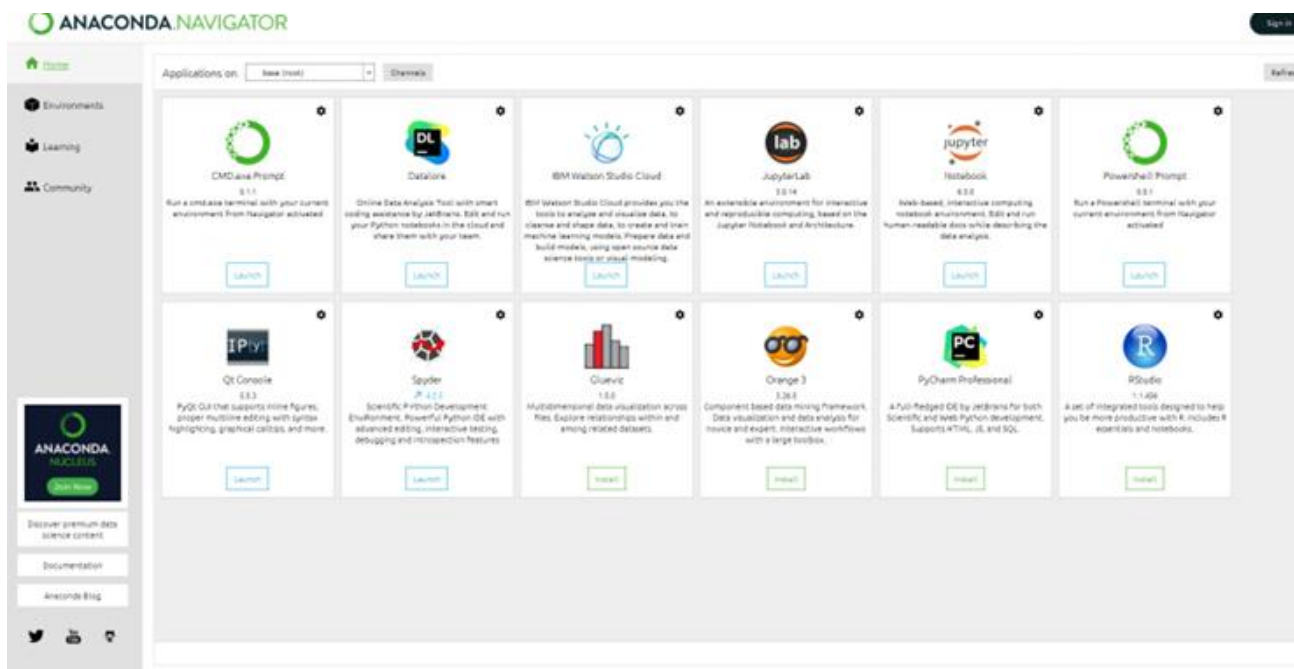
#### Introduction to Anaconda:

Anaconda is a package manager, environment manager, and Python distribution with a collection of 1,500+ open source packages with free community support. Anaconda is free and easy to install and can be used on Windows, macOS, or Linux.

Anaconda can be downloaded from <https://www.anaconda.com/products/individual>

After installing Anaconda, we use Anaconda Navigator to launch applications and easily manage packages, environments and channels without using command-line commands.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in the terminal window.





The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App
- RStudio
- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)

### **Jupyter Notebook :**

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results.

The Jupyter notebook combines two components:

**A web application:** a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

**Notebook documents:** a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects

## Installing Python:

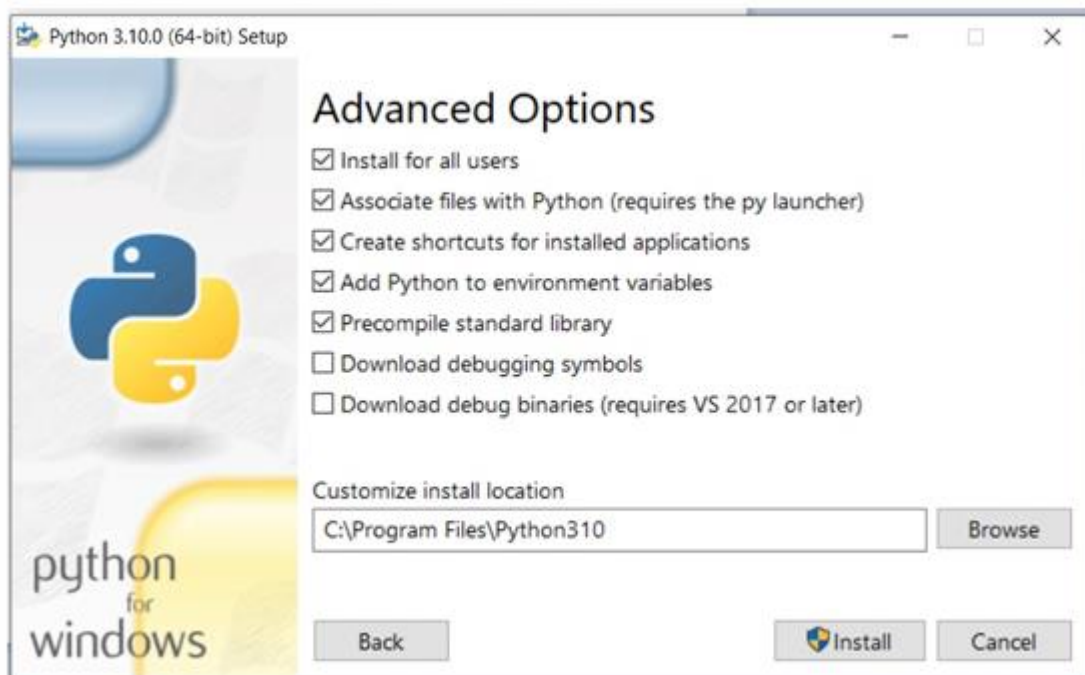
NOTE: Python 3.10 supports Windows 8.1 and newer.

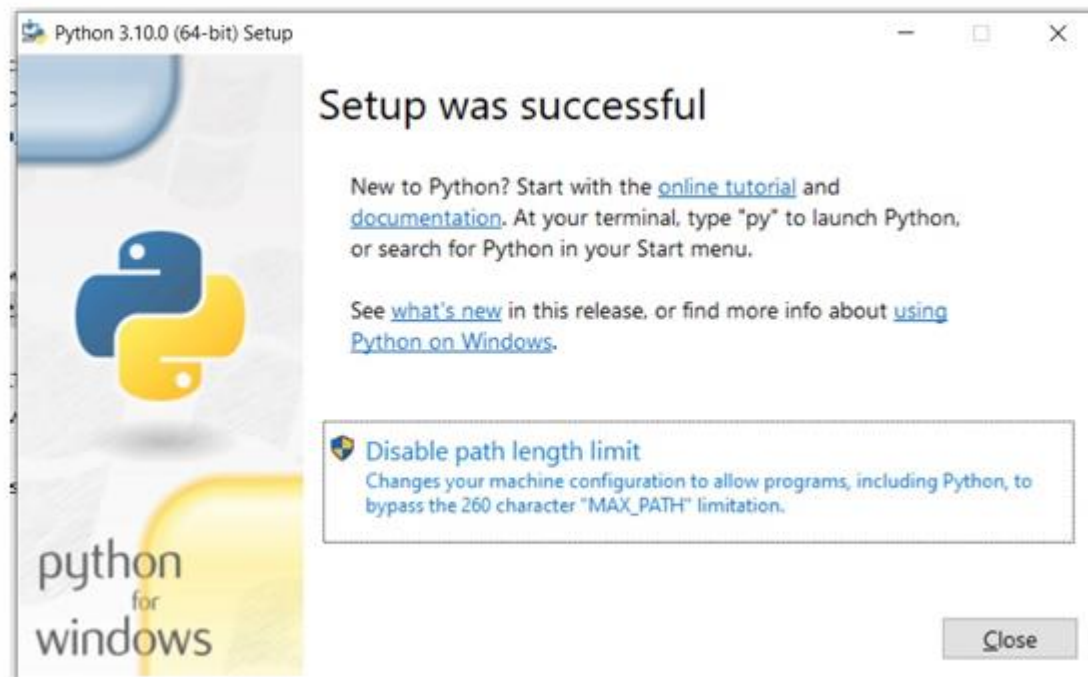
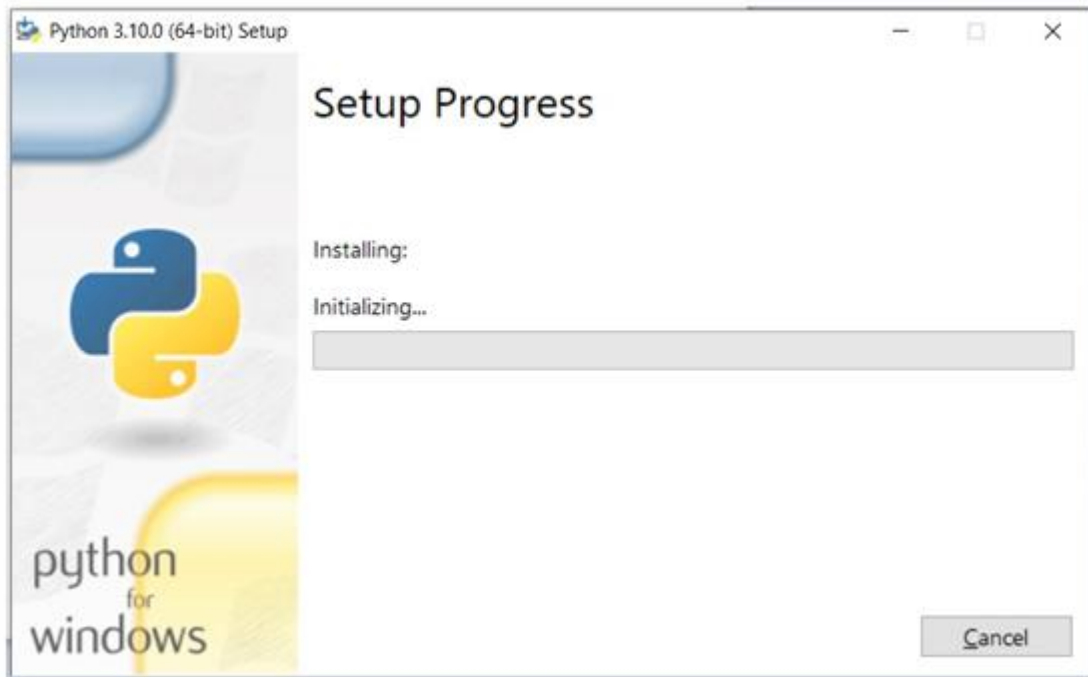
If you require Windows 7 support, please install Python 3.8.

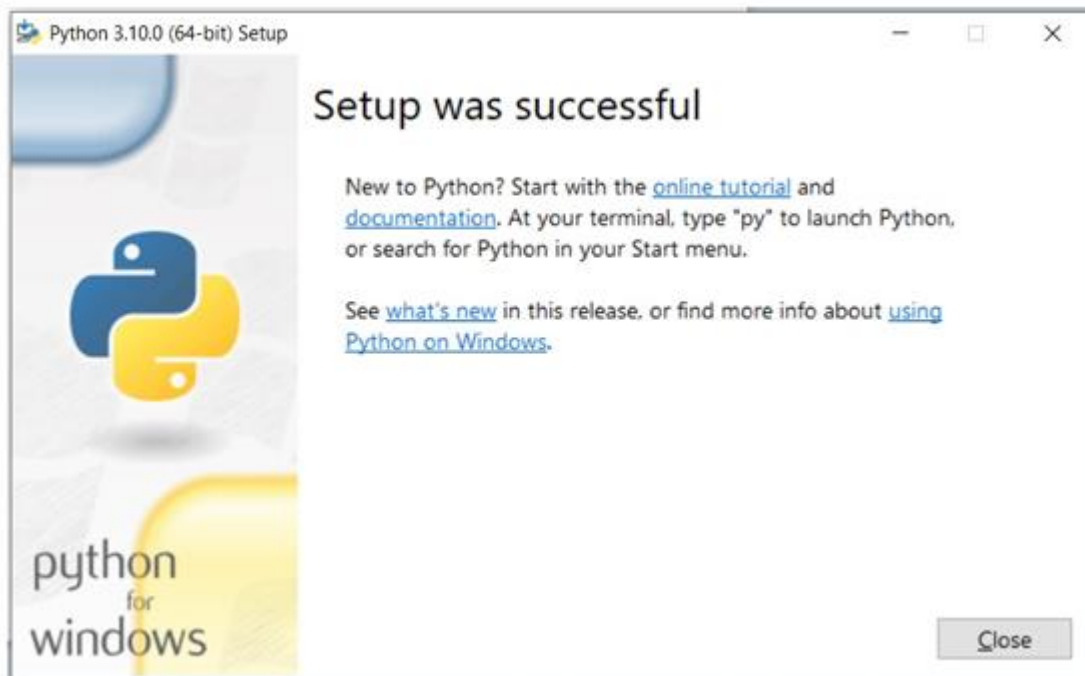
For full installation: download “Python 3.10” installer available for download. The following dialogue box appears.



Select any one of the option and continue.



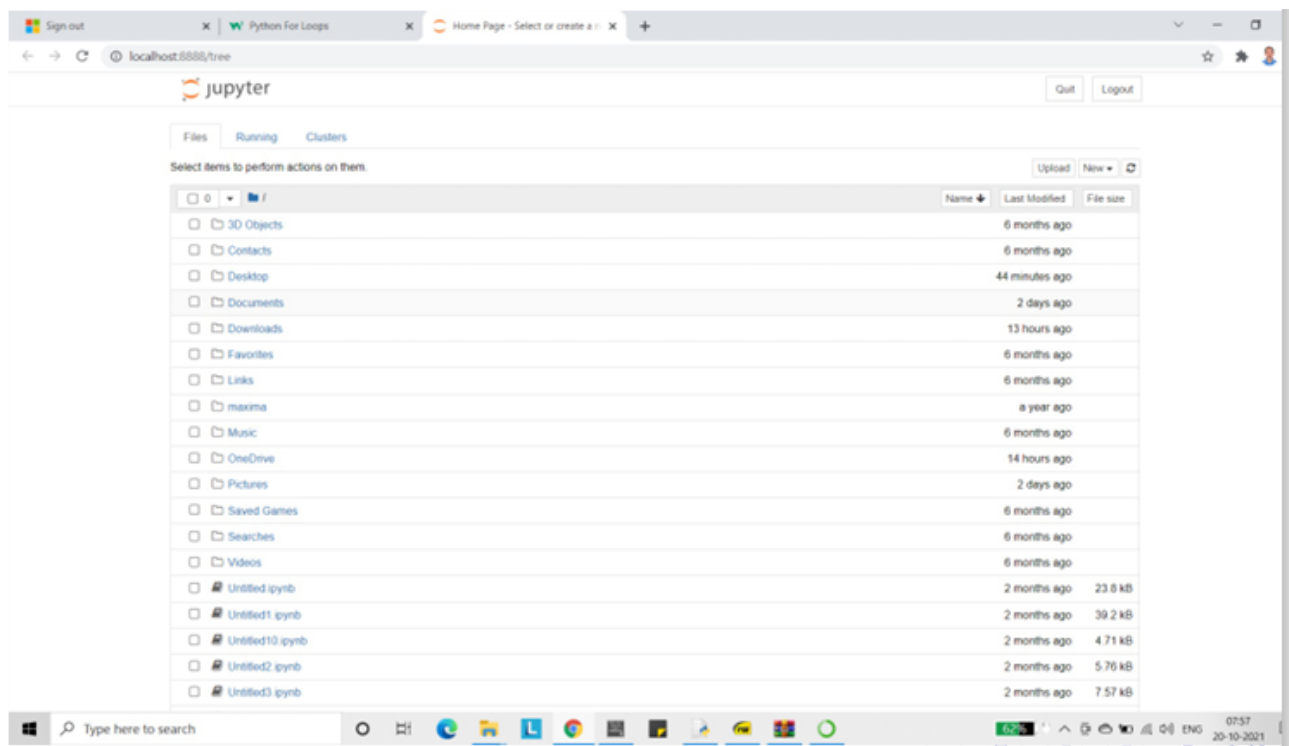




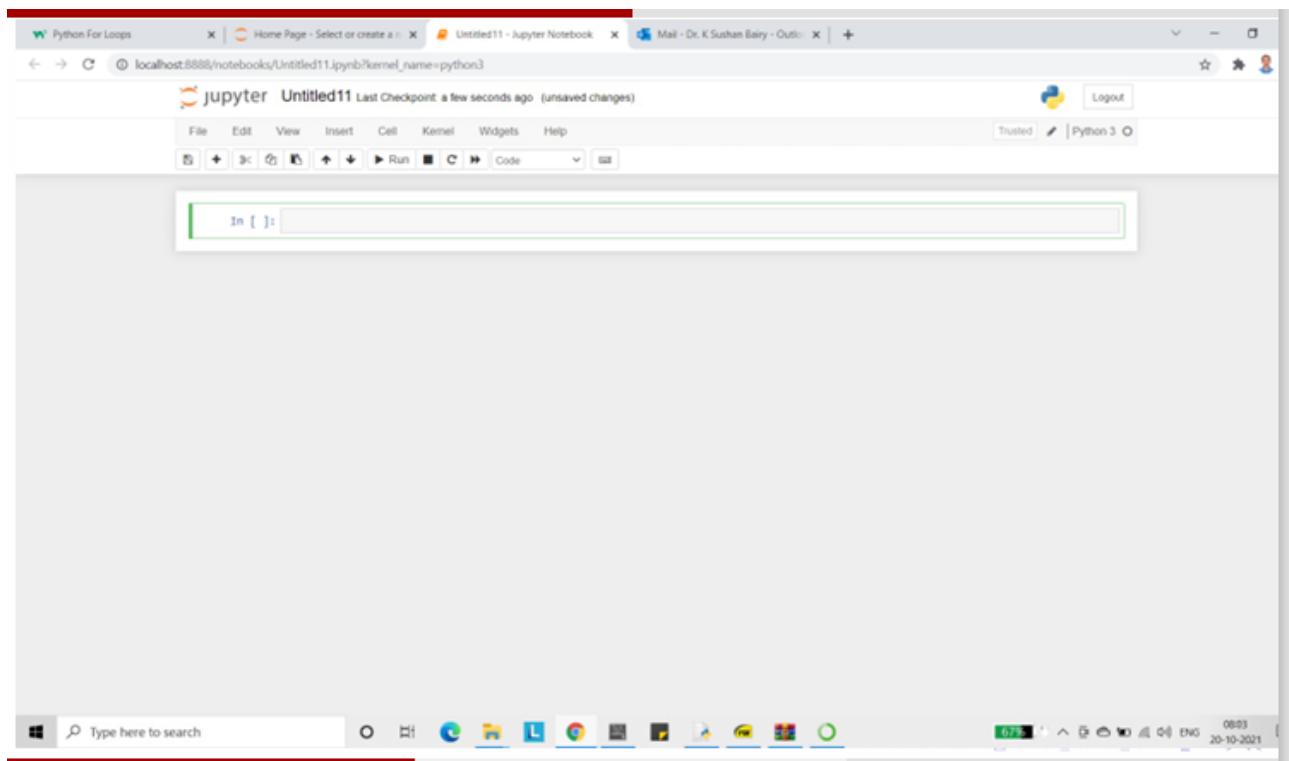
This completes the successful installation of “Python 3.10.0”.

#### INSTALLING PACKAGES FOR PYTHON:

- Open command prompt by searching cmd
- pip (package installer for Python) is used to install packages from Python Package Index and other indexes. First we update the ‘pip’ to latest version and then use pip to install the packages.
- Type: `python-m pip install--upgrade pip`
- `pip install numpy`
- `pip install sympy`
- `pip install matplotlib`
- `pip install statistics`



Click on New and then select Python 3



## Contents: Computer Science and Engineering Stream

- Lab 1. Programme to compute area, volume and center of gravity.
- Lab 2. Evaluation of improper integrals , Beta and Gamma functions.
- Lab 3. Finding gradient, divergent, curl and their geometrical interpretation
- Lab 4. Computation of basis and dimension for a vector space and graphical representation of linear transformation
- Lab 5. Computing the inner product and orthogonality
- Lab 6. Solution of algebraic and transcendental equation by Regula-Falsi and Newton-Raphson method
- Lab 7. Interpolation /Extrapolation using Newton's forward and backward difference formula
- Lab 8. Computation of area under the curve using Trapezoidal, Simpson's  $\left(\frac{1}{3}\right)^{\text{rd}}$  and Simpsons  $\left(\frac{3}{8}\right)^{\text{th}}$  rule
- Lab 9. Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method
- Lab 10. Solution of ODE of first order and first degree by Runge-Kutta 4th order method and Milne's predictor and corrector method

# LAB 1: Programme to compute area, volume and center of gravity

## 1.1 Objectives:

Use python

1. to evaluate double integration.
2. to compute area and volume.
3. to calculate center of gravity of 2D object.

### Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

## 1.2 Double and triple integration

### Example 1:

Evaluate the integral  $\int_0^1 \int_0^x (x^2 + y^2) dy dx$

```
from sympy import *
x,y,z=symbols('x y z')
w1=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(w1)
```

1/3

### Example 2:

Evaluate the integral  $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz) dz dy dx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w2=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w2)
```

81/80



### Example 3:

Prove that  $\int \int (x^2 + y^2) dy dx = \int \int (x^2 + y^2) dx dy$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w3=integrate(x**2+y**2,y,x)
pprint(w3)
w4=integrate(x**2+y**2,x,y)
pprint(w4)
```

## 1.3 Area and Volume

Area of the region R in the cartesian form is  $\int \int_R dx dy$

### Example 4:

Find the area of an ellipse by double integration.  $A=4 \int_0^a \int_0^{(b/a)\sqrt{a^2-x^2}} dy dx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
#a=Symbol('a')
#b=Symbol('b')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)
```

24.0\*pi

## 1.4 Area of the region R in the polar form is $\int \int_R r dr d\theta$

### Example 5:

Find the area of the cardioid  $r = a(1 + \cos\theta)$  by double integration

```
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
#a=4
w3=2*integrate(r,(r,0,a*(1+cos(t))), (t,0,pi))
pprint(w3)
```

## 1.5 Volume of a solid is given by $\int \int \int_V dx dy dz$

### Example 6:

Find the volume of the tetrahedron bounded by the planes  $x=0, y=0$  and  $z=0, \frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
w2=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(w2)
```

$a*b*c/6$

## 1.6 Center of Gravity

Find the center of gravity of cardioid . Plot the graph of cardioid and mark the center of gravity.

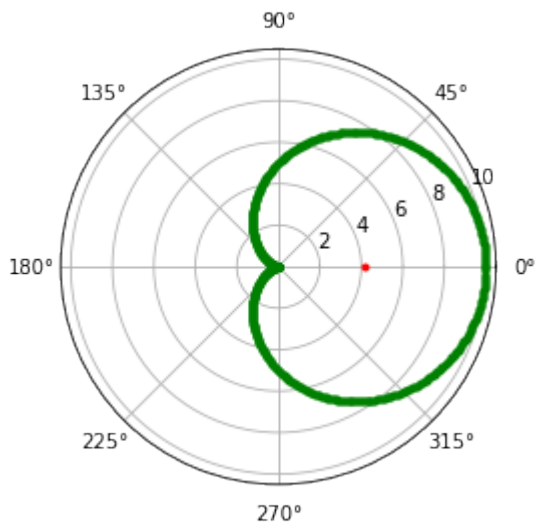
```
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection = 'polar')
a=5

rad = np.arange(0, (2 * np.pi), 0.01)

# plotting the cardioid
for i in rad:
    r = a + (a*np.cos(i))
    plt.polar(i,r,'g.')

plt.polar(0,I,'r.')
plt.show()
```

$5*a/6$



### 1.7 Exercise:

1. Evaluate  $\int_0^1 \int_0^x (x+y) dy dx$

Ans: 0.5

2. Find the  $\int_0^{\log(2)} \int_0^x \int_0^{x+\log(y)} (e^{x+y+z}) dz dy dx$

Ans: -0.2627

3. Find the area of positive quadrant of the circle  $x^2 + y^2 = 16$

Ans:  $4\pi$

4. Find the volume of the tetrahedron bounded by the planes  $x=0, y=0$  and  $z=0$ ,

$$\frac{x}{2} + \frac{y}{3} + \frac{z}{4} = 1$$

Ans: 4

# LAB 2: Evaluation of improper integrals, Beta and Gamma functions

## 2.1 Objectives:

Use python

1. to evaluate improper integrals using Beta function.
2. to evaluate improper integrals using Gamma function.

Syntax for the commands used:

1. gamma

```
math.gamma(x)
```

**Parameters :**

**x :** The number whose gamma value needs to be computed.

2. beta

```
math.beta(x,y)
```

**Parameters :**

**x ,y:** The numbers whose beta value needs to be computed.

3. **Note:** We can evaluate improper integral involving infinity by using `inf`.

**Example 1:**

Evaluate  $\int_0^{\infty} e^{-x} dx$ .

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x),(x,0,float('inf')))
print(simplify(w1))
```

1

**Gamma** function is  $x(n) = \int_0^{\infty} e^{-x} x^{n-1} dx$

**Example 2:**

Evaluate  $\Gamma(5)$  by using definition

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(w1))
```

24

### Example 3:

Evaluate  $\int_0^{\infty} e^{-st} \cos(4t) dt$  . That is Laplace transform of  $\cos(4t)$

```
from sympy import *
t,s=symbols('t,s')
# for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(simplify(w1))
```

$$\begin{cases} \frac{s}{s^2+16} & \text{for } 2|\arg(s)| < \pi \\ \int_0^{\infty} e^{-st} \cos(4t) dt & \text{otherwise} \end{cases}$$

### Example 4:

Find Beta(3,5), Gamma(5)

```
#beta and gamma functions
from sympy import beta, gamma
m=input('m : ');
n=input('n : ');
m=float(m);
n=float(n);
s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is %3.3f'%s)
```

```
m :3
n :5
gamma ( 5.0 ) is 24.000
Beta ( 3.0 5.0 ) is 0.010
```

### Example 5:

Calculate Beta(5/2,7/2) and Gamma(5/2).

```
#beta and gamma functions
# If the number is a fraction give it in decimals. Eg 5/2=2.5
from sympy import beta, gamma
m=float(input('m : '));
n=float(input('n : '));

s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is %3.3f '%s)
```

```

m : 2.5
n : 3.5
gamma ( 3.5 ) is 3.323
Beta ( 2.5 3.5 ) is 0.037

```

### Example 6:

Verify that  $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$  for  $m=5$  and  $n=7$

```

from sympy import beta, gamma
m=5;
n=7;
m=float(m);
n=float(n);
s=beta(m,n);
t=(gamma(m)*gamma(n))/gamma(m+n);
print(s,t)
if (abs(s-t)<=0.00001):
    print('beta and gamma are related')
else:
    print('given values are wrong')

```

```

0.000432900432900433 0.000432900432900433
beta and gamma are related

```

## 2.2 Exercise:

1. Evaluate  $\int_0^{\infty} e^{-t} \cos(2t) dt$   
Ans:  $1/5$
2. Find the value of  $Beta(5/2, 9/2)$   
Ans: 0.0214
3. Find the value of  $Gamma(13)$   
Ans: 479001600
4. Verify that  $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$  for  $m=7/2$  and  $n=11/2$   
Ans: True

# LAB 3: Finding gradient, divergent, curl and their geometrical interpretation

## 1.1 Objectives:

Use python

1. to find the gradient of a given scalar function.
2. to find find divergence and curl of a vector function.

## 1.2 Method I:

To find gradient of  $\phi = x^2y + 2xz - 4$ .

```
#To find gradient of scalar point function.
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N') #Setting the coordinate system
x,y,z=symbols('x y z')
A=N.x**2*N.y+2*N.x*N.z-4 #Variables x,y,z to be used with coordinate
                           system N
delop=Del() #Del operator
display(delop(A)) #Del operator applied to A
gradA=gradient(A) #Gradient function is used
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$\left( \frac{\partial}{\partial x_N} (x_N^2 y_N + 2x_N z_N - 4) \right) \hat{i}_N + \left( \frac{\partial}{\partial y_N} (x_N^2 y_N + 2x_N z_N - 4) \right) \hat{j}_N + \left( \frac{\partial}{\partial z_N} (x_N^2 y_N + 2x_N z_N - 4) \right) \hat{k}_N$$

Gradient of  $N.x**2*N.y + 2*N.x*N.z - 4$  is

$$(2x_N y_N + 2z_N) \hat{i}_N + (x_N^2) \hat{j}_N + (2x_N) \hat{k}_N$$

To find divergence of  $\vec{F} = x^2 y z \hat{i} + y^2 z x \hat{j} + z^2 x y \hat{k}$

```
#To find divergence of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
divA=delop.dot(A)
display(divA)

print(f"\n Divergence of {A} is \n")
display(divergence(A))
```

$$\frac{\partial}{\partial z_N} x_N y_N z_N^2 + \frac{\partial}{\partial y_N} x_N y_N^2 z_N + \frac{\partial}{\partial x_N} x_N^2 y_N z_N$$

Divergence of  $N.x^{**2}*N.y*N.z*N.i + N.x*N.y^{**2}*N.z*N.j + N.x*N.y*N.z^{**2}*N.k$  is

$$6x_N y_N z_N$$

To find curl of  $\vec{F} = x^2 y z \hat{i} + y^2 z x \hat{j} + z^2 x y \hat{k}$

```
#To find curl of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
curlA=delop.cross(A)
display(curlA)

print(f"\n Curl of {A} is \n")
display(curl(A))
```

$$\left( \frac{\partial}{\partial y_N} x_N y_N z_N^2 - \frac{\partial}{\partial z_N} x_N y_N^2 z_N \right) \hat{i}_N + \left( -\frac{\partial}{\partial x_N} x_N y_N z_N^2 + \frac{\partial}{\partial z_N} x_N^2 y_N z_N \right) \hat{j}_N + \left( \frac{\partial}{\partial x_N} x_N y_N^2 z_N - \frac{\partial}{\partial y_N} x_N^2 y_N z_N \right) \hat{k}_N$$

Curl of  $N.x^{**2}*N.y*N.z*N.i + N.x*N.y^{**2}*N.z*N.j + N.x*N.y*N.z^{**2}*N.k$  is

$$(-x_N y_N^2 + x_N z_N^2) \hat{i}_N + (x_N^2 y_N - y_N z_N^2) \hat{j}_N + (-x_N^2 z_N + y_N^2 z_N) \hat{k}_N$$

### 1.3 Method II:

To find gradient of  $\phi = x^2 y z$ .

```
#To find gradient of a scalar point function x^2yz
from sympy.physics.vector import *
from sympy import var, pprint
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given scalar function F=")
display(F)
G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("\n Gradient of F=")
display(G)
```



Given scalar function F=

$$x^2yz$$

Gradient of F=

$$2xyz\hat{\mathbf{v}}_x + x^2z\hat{\mathbf{v}}_y + x^2y\hat{\mathbf{v}}_z$$

To find divergence of  $\vec{F} = x^2y\hat{i} + yz^2\hat{j} + x^2z\hat{k}$ .

```
#To find divergence of F=x^2yi+yz^2j+x^2zk
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v.x+v[1]*v[2]**2*v.y+v[0]**2*v[2]*v.z
G=divergence(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Divergence of F=")
display(G)
```

Given vector point function is

$$x^2y\hat{\mathbf{v}}_x + yz^2\hat{\mathbf{v}}_y + x^2z\hat{\mathbf{v}}_z$$

Divergence of F=

$$x^2 + 2xy + z^2$$

To find curl of  $\vec{F} = xy^2\hat{i} + 2x^2yz\hat{j} - 3yz^2\hat{k}$

```
#To find curl of F=xy^2i+2x^2yzj-3yz^2k
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]**2*v.x+2*v[0]**2*v[1]*v[2]*v.y-3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("curl of F=")
display(G)
```

Given vector point function is

$$xy^2 \hat{\mathbf{v}}_{\mathbf{x}} + 2x^2yz\hat{\mathbf{v}}_{\mathbf{y}} - 3yz^2 \hat{\mathbf{v}}_{\mathbf{z}}$$

curl of  $\mathbf{F}$ =

$$(-2x^2y - 3z^2)\hat{\mathbf{v}}_{\mathbf{x}} + (4xyz - 2xy)\hat{\mathbf{v}}_{\mathbf{z}}$$

#### 1.4 Exercise:

1. If  $u = x + y + z$ ,  $v = x^2 + y^2 + z^2$ ,  $w = yz + zx + xy$ , find  $\text{grad}u$ ,  $\text{grad}v$  and  $\text{grad}w$ .  
Ans:  $\hat{i} + \hat{j} + \hat{k}$ ,  $2(x\hat{i} + y\hat{j} + z\hat{k})$ ,  $(y + z)\hat{i} + (z + x)\hat{j} + (z + x)\hat{k}$ .
2. Evaluate  $\text{div}F$  and  $\text{curl}F$  at the point  $(1,2,3)$ , given that  $\vec{F} = x^2yz\hat{i} + xy^2z\hat{j} + xyz^2\hat{k}$ .  
Ans:  $6xyz$ ,  $x(z^2 - y^2)\hat{i} + y(x^2 - z^2)\hat{j} + z(y^2 - x^2)\hat{k}$ .
3. Prove that the vector  $(yz - x^2)\hat{i} + (4y - z^2x)\hat{j} + (2xz - 4z)\hat{k}$  is solenoidal.
4. Find the vector normal to the surface  $xy^3z^2 = 4$  at the point  $(-1, -1, 2)$ .  
Ans:  $-4\hat{i} - 12\hat{j} + 4\hat{k}$ .
5. If  $\vec{R} = x\hat{i} + y\hat{j} + z\hat{k}$ , show that (i)  $\nabla \cdot \vec{R} = 3$ , (ii)  $\nabla \times \vec{R} = 0$ .

# LAB 4: Computation of basis and dimension for a vector space and graphical representation of linear transformation

## 4.1 Objectives:

Use python

1. to verify the Rank nullity theorem of given linear transformation
2. to compute the dimension of vector space
3. to represent linear transformations graphically

## 4.2 Rank Nullity Theorem

Verify the rank-nullity theorem for the linear transformation  $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$  defined by  $T(x, y, z) = (x + 4y + 7z, 2x + 5y + 8z, 3x + 6y + 9z)$ .

```
import numpy as np
from scipy.linalg import null_space

# Define a linear transformation in terms of matrix
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Find the rank of the matrix A
rank = np.linalg.matrix_rank(A)
print("Rank of the matrix",rank)

# Find the null space of the matrix A
ns = null_space(A)
print("Null space of the matrix",ns)
# Find the dimension of the null space
nullity = ns.shape[1]
print("Null space of the matrix",nullity)
# Verify the rank-nullity theorem
if rank + nullity == A.shape[1]:
    print("Rank-nullity theorem holds.")
else:
    print("Rank-nullity theorem does not hold.")
```

Rank of the matrix 2

Null space of the matrix  $\begin{bmatrix} -0.40824829 \\ 0.81649658 \\ -0.40824829 \end{bmatrix}$

Null space of the matrix 1

Rank-nullity theorem holds.

### 4.3 Dimension of Vector Space

Find the dimension of subspace spanned by the vectors  $(1, 2, 3)$ ,  $(2, 3, 1)$  and  $(3, 1, 2)$ .

```
import numpy as np

# Define the vector space V
V = np.array([
    [1, 2, 3],
    [2, 3, 1],
    [3, 1, 2]])
# Find the dimension and basis of V
basis = np.linalg.matrix_rank(V)
dimension = V.shape[0]
print("Basis of the matrix",basis)
print("Dimension of the matrix",dimension)
```

Basis of the matrix 3

Dimension of the matrix 3

Extract the linearly independent rows in given matrix : Basis of Row space

```
from numpy import *
import sympy as sp
A=[[1,-1,1,1],[2,-5,2,2],[3,-3,5,3],[4,-4,4,4]]
AB=array(A)
S=shape(A)
n=len(A)
for i in range(n):
    if AB[i,i]==0:
        ab=copy(AB)
        for k in range(i+1,S[0]):
            if ab[k,i]!=0:
                ab[i,:]=AB[k,:]
                ab[k,:]=AB[i,:]
                AB=copy(ab)
        for j in range(i+1,n):
            Fact=AB[j,i]/AB[i,i]
            for k in range(i,n):
                AB[j,k]=AB[j,k]-Fact*AB[i,k]
display("REF of given matrix: ",sp.Matrix(AB))
temp = {(0, 0, 0,0)}
result = []
for idx, row in enumerate(map(tuple, AB)):
    if row not in temp:
        result.append(idx)
print("\n Basis are non-zero rows of A:")
display(sp.Matrix(AB[result]))
```

'REF of given matrix: '

$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Basis are non-zero rows of A:

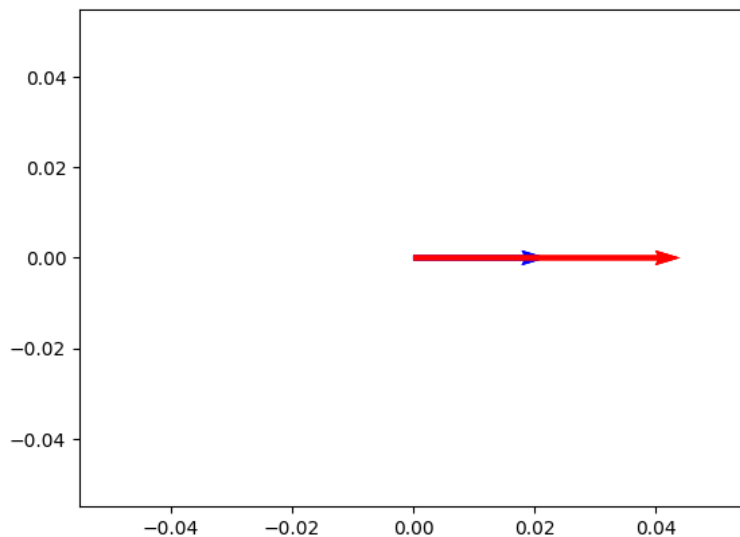
$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

## 4.4 Graphical representation of a transformation

### 4.4.1 Horizontal stretch:

Represent the horizontal stretch transformation  $T : R^2 \rightarrow R^2$  geometrically  
Find the image of vector  $(10, 0)$  when it is stretched horizontally by 2 units.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

```
from math import pi, sin, cos
```

```

import matplotlib.pyplot as plt
import numpy as np

coords = np.array([[0,0],[0.5,0.5],[0.5,1.5],[0,1],[0,0]])
coords = coords.transpose()
coords
x = coords[0,:]
y = coords[1,:]

A = np.array([[2,0],[0,1]])
A_coords = A@coords
x_LT1 = A_coords[0,:]
y_LT1 = A_coords[1,:]

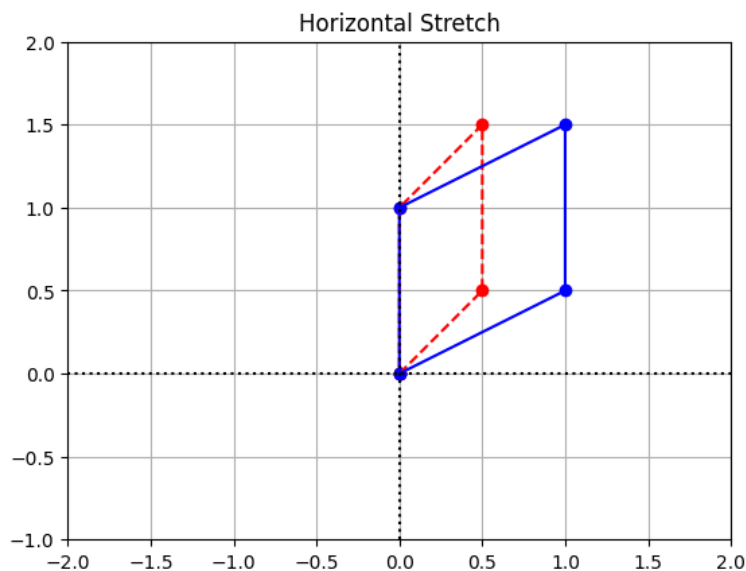
# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points. x and y are original vectors, x_LT1 and y_LT1 are
# images
ax.plot(x,y,'ro')
ax.plot(x_LT1,y_LT1,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT1,y_LT1,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Horizontal Stretch");

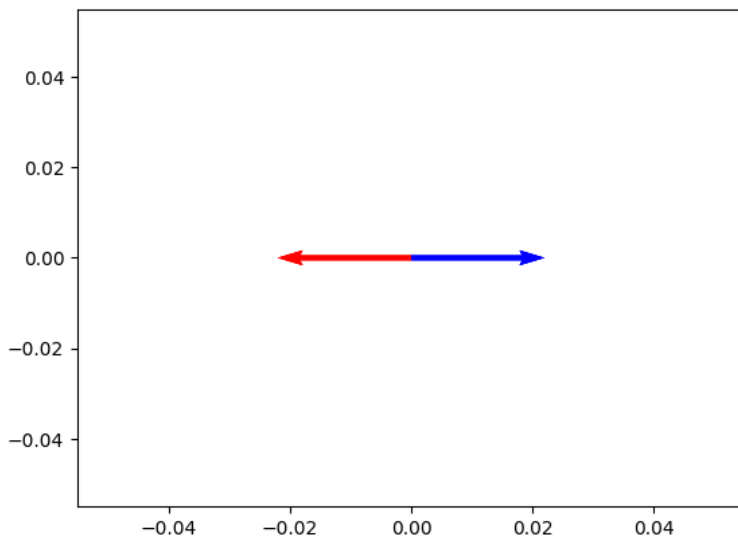
```



#### 4.4.2 Reflection:

Represent the reflection transformation  $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  geometrically.  
Find the image of vector  $(10, 0)$  when it is reflected about  $y$  axis.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[-1,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1:], color=['r'], scale=50)
plt.show()
```



Another example.

```
B = np.array([[-1,0],[0,1]])
B_coords = B@coords

x_LT2 = B_coords[0,:]
y_LT2 = B_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points. x and y are original vectors, x_LT1 and y_LT1 are
# images
ax.plot(x,y,'ro')
ax.plot(x_LT2,y_LT2,'bo')

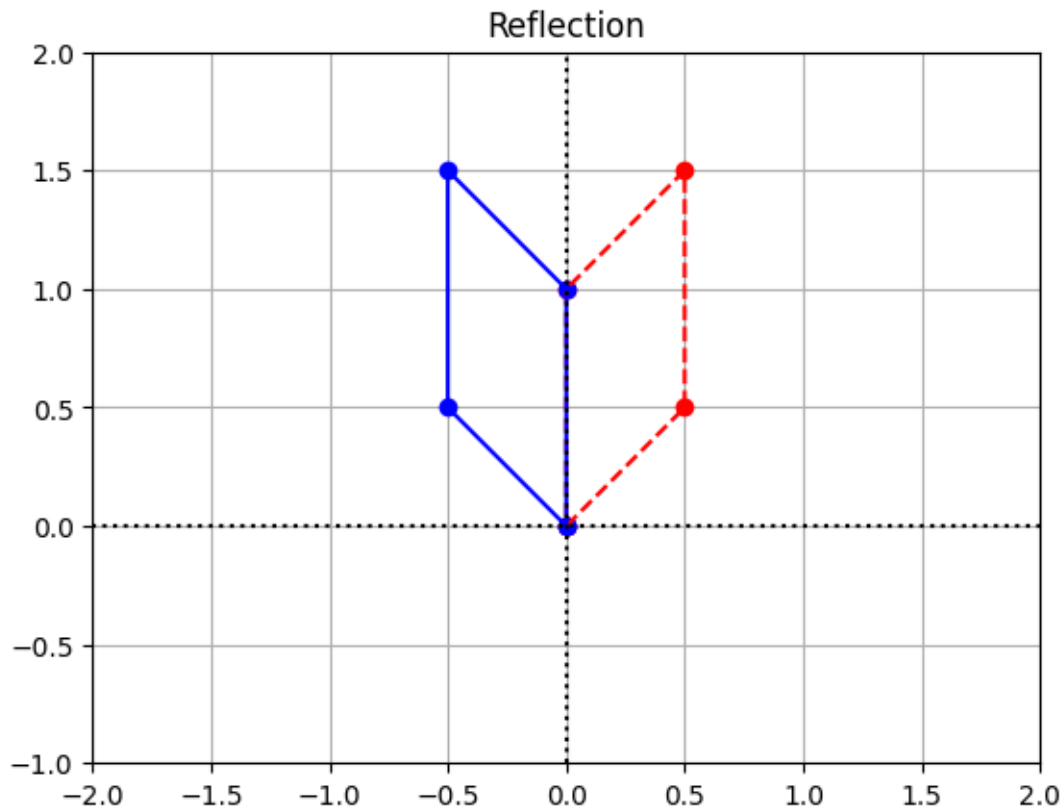
# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT2,y_LT2,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
```

```

ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Reflection");

```



#### 4.4.3 Rotation:

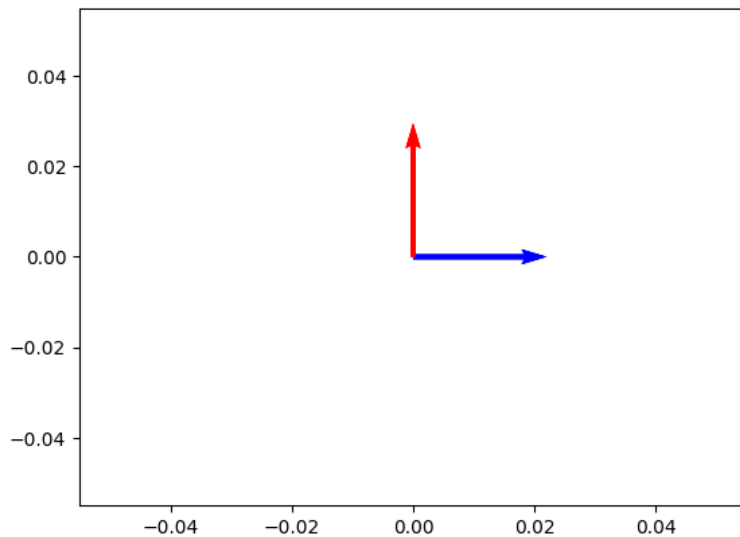
Represent the rotation transformation  $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  geometrically.  
Find the image of vector  $(10, 0)$  when it is rotated by  $\pi/2$  radians.

```

import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()

```





Another example.

```
theta = pi/6
R = np.array([[cos(theta), -sin(theta)], [sin(theta), cos(theta)]])
R_coords = R@coords

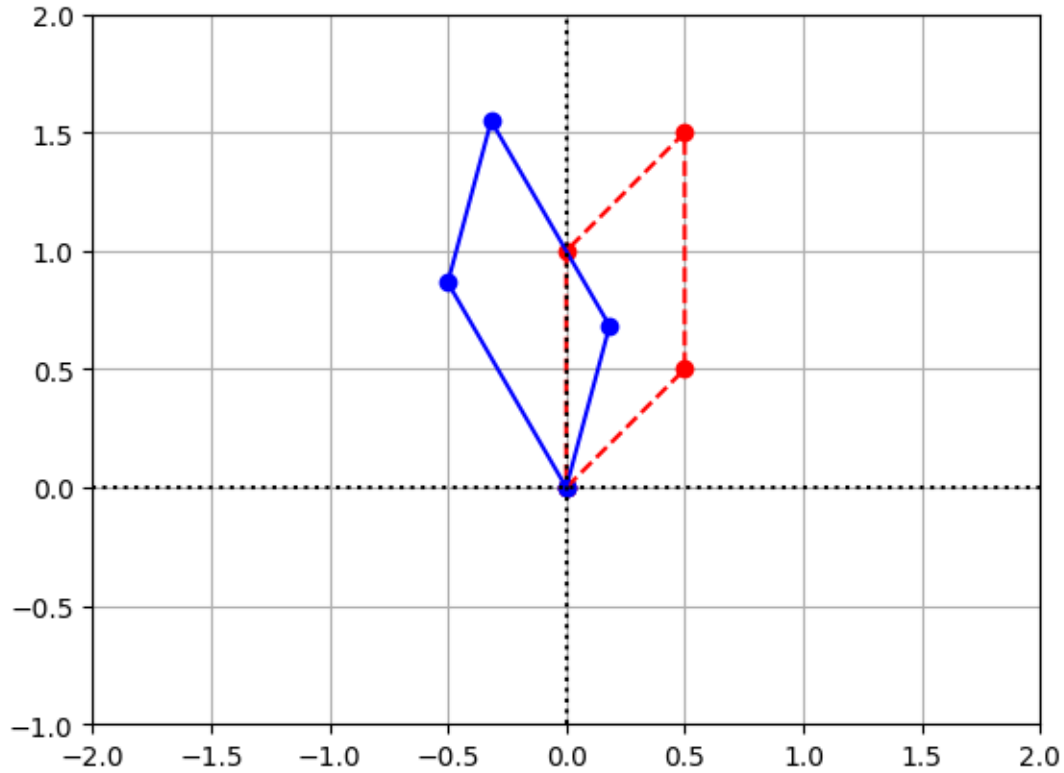
x_LT3 = R_coords[0,:]
y_LT3 = R_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points. x and y are original vectors, x_LT1 and y_LT1 are
# images
ax.plot(x,y,'ro')
ax.plot(x_LT3,y_LT3,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT3,y_LT3,'b')

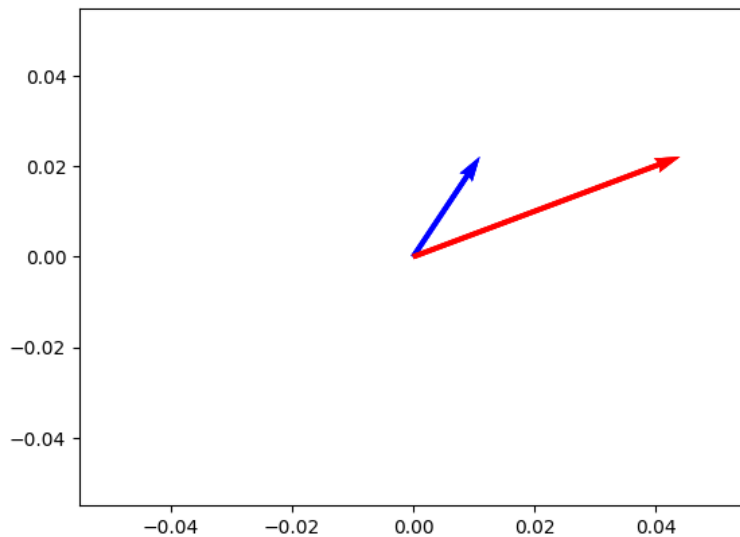
# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



#### 4.4.4 Shear Transformation

Represent the Shear transformation  $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  geometrically.  
Find the image of  $(2, 3)$  under shear transformation.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[1,2],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
print("Image of given vectors is:", V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1:], color=['r'], scale=20)
plt.show()
```



Another example.

```
S = np.array([[1,2],[0,1]])
S_coords = S@coords

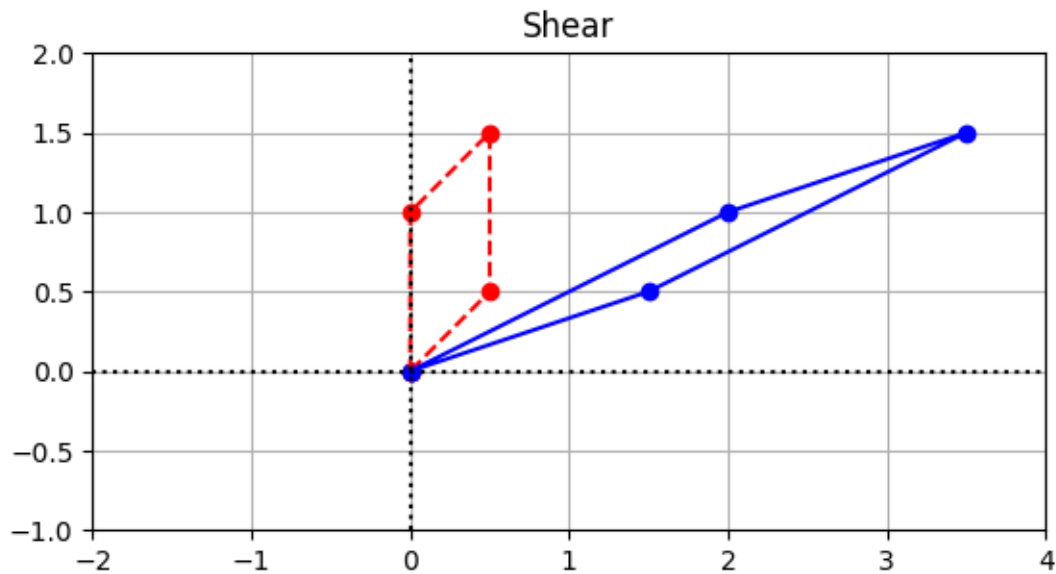
x_LT4 = S_coords[0,:]
y_LT4 = S_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points. x and y are original vectors, x_LT1 and y_LT1 are
# images
ax.plot(x,y,'ro')
ax.plot(x_LT4,y_LT4,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT4,y_LT4,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,4,-1,2])
ax.set_aspect('equal')
ax.set_title("Shear");
```

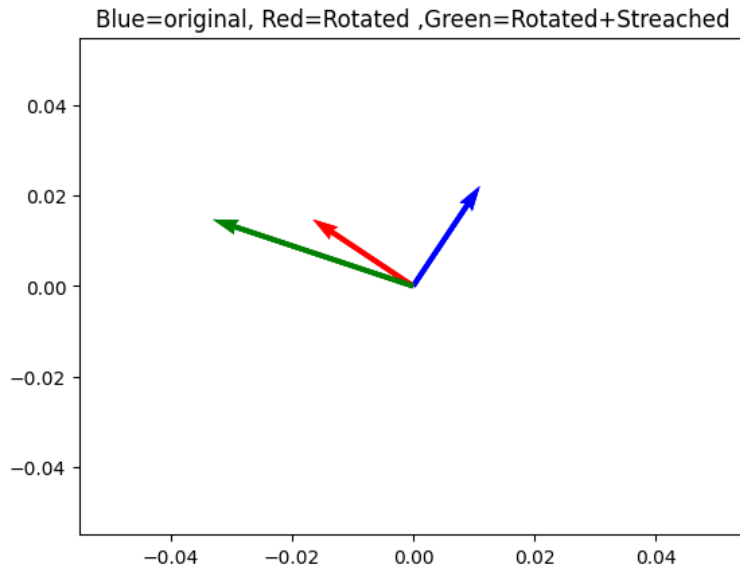


#### 4.4.5 Composition

Represent the composition of two 2D transformations.

Find the image of vector  $(10, 0)$  when it is rotated by  $\pi/2$  radians then stretched horizontally 2 units.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,0]])
B=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V3=B*V2
V2=np.array(V2)
V3=np.array(V3)
print("Image of given vectors is:", V3)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1:], color=['r'], scale=20)
plt.quiver(*origin, V3[0,:], V3[1:], color=['g'], scale=20)
plt.title('Blue=original, Red=Rotated ,Green=Rotated+Stretched')
plt.show()
```



Another example.

```
C = np.array([[ -cos(theta), sin(theta)], [sin(theta), cos(theta)]])
C_coords = C@coords

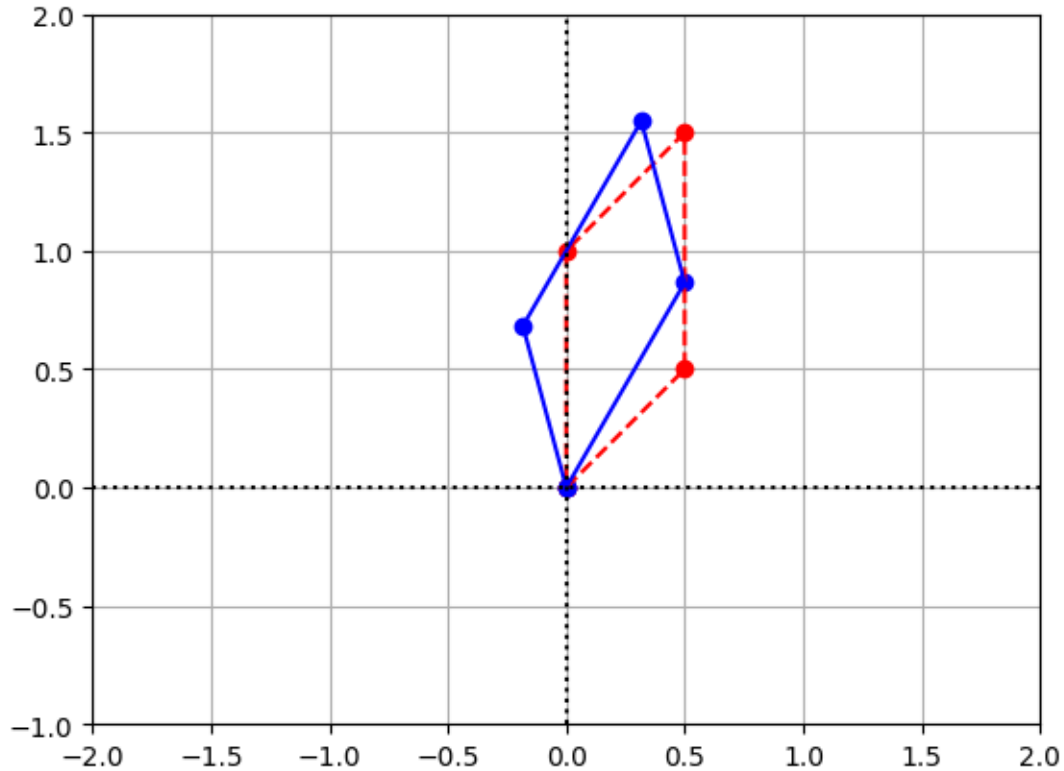
x_LT5 = C_coords[0,:]
y_LT5 = C_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points. x and y are original vectors, x_LT1 and y_LT1 are
# images
ax.plot(x,y, 'ro')
ax.plot(x_LT5,y_LT5, 'bo')

# Connect the points by lines
ax.plot(x,y, 'r', ls="--")
ax.plot(x_LT5,y_LT5, 'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



#### 4.5 Exercise:

1. Verify the rank nullity theorem for the following linear transformation
  - a)  $T : \mathbb{R}^2 \rightarrow \mathbb{R}^3$  defined by  $T(x, y) = (x + 4y, 2x + 5y, 3x + 6y)$ .  
 Ans: Rank=2, Nullity=1, RNT verified
  - b)  $T : \mathbb{R}^3 \rightarrow \mathbb{R}^4$  defined by  $T(x, y, z) = (x + 4y - z, 2x + 5y + 8z, 3x + y + 2z, x + y + z)$ .  
 Ans: Rank=3, Nullity=1, RNT verified
2. Find the dimension of the subspace spanned following set of vectors
  - a)  $S = (1, 2, 3, 4), (2, 4, 6, 8), (1, 1, 1, 1)$   
 Ans: Dimension of subspace is 2
  - b)  $S = (1, -1, 3, 4), (2, 1, 6, 8), (1, 1, 1, 1), (3, 3, 3, 3)$   
 Ans: Dimension of subspace is 3
3. Find the image of  $(1, 3)$  under following  $2D$  transformations
  - a) Horizontal stretch
  - b) Reflection
  - c) Shear
  - d) Rotation

# LAB 5: Computing the inner product and orthogonality

## 5.1 Objectives:

Use python

1. to compute the inner product of two vectors.
2. to check whether the given vectors are orthogonal.

## 5.2 Inner Product of two vectors

Find the inner product of the vectors  $(2, 1, 5, 4)$  and  $(3, 4, 7, 8)$ .

```
import numpy as np

#initialize arrays
A = np.array([2, 1, 5, 4])
B = np.array([3, 4, 7, 8])

#dot product
output = np.dot(A, B)

print(output)
```

77

## 5.3 Checking orthogonality

Verify whether the following vectors  $(2, 1, 5, 4)$  and  $(3, 4, 7, 8)$  are orthogonal.

```
import numpy as np

#initialize arrays
A = np.array([2, 1, 5, 4])
B = np.array([3, 4, 7, 8])

#dot product
output = np.dot(A, B)
print('Inner product is :',output)
if output==0:
    print('given vectors are orthognal ')
else:
    print('given vectors are not orthognal ')
```

Inner product is : 77  
given vectors are not orthognal

## 5.4 Exercise:

1. Find the inner product of  $(1, 2, 3)$  and  $(3, 4, 5)$ .

Ans: 26

2. Find the inner product of  $(1, -1, 2, 1)$  and  $(4, 2, 1, 0)$ .

Ans: 4

3. Check whether the following vectors are orthogonal or not

a)  $(1, 1, -1)$  and  $(2, 3, 5)$ . Ans: True

b)  $(1, 0, 2, 0)$  and  $(4, 2, -2, 5)$ . Ans: True

c)  $(1, 2, 3, 4)$  and  $(2, 3, 4, 5)$  . Ans: False



# LAB 6: Solution of algebraic and transcendental equation by Regula-Falsi and Newton-Raphson method

## 6.1 Objectives:

Use python

1. to solve algebraic and transcendental equation by Regula-Falsi method.
2. to solve algebraic and transcendental equation by Newton-Raphson method.

## 6.2 Regula-Falsi method to solve a transcendental equation

Obtain a root of the equation  $x^3 - 2x - 5 = 0$  between 2 and 3 by regula-falsi method. Perform 5 iterations.

```
# Regula Falsi method
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  %#x^3-2*x-5;      %function
f=lambdify(x,g)
a=float(input('Enter a value :')) #2
b=float(input('Enter b value :')) # 3
N=int(input('Enter number of iterations :')) #5

for i in range(1,N+1):
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%(i,c,f(c)));
```

```
Enter the function  x**3-2*x-5
Enter a value :2
Enter b value :3
Enter number of iterations :5
iteration 1          the root 2.059          function value -0.391

iteration 2          the root 2.081          function value -0.147

iteration 3          the root 2.090          function value -0.055

iteration 4          the root 2.093          function value -0.020

iteration 5          the root 2.094          function value -0.007
```

Using tolerance value we can write the same program as follows:

Obtain a root of the equation  $x^3 - 2x - 5 = 0$  between 2 and 3 by regula-falsi method. Correct to 3 decimal places.

```

# Regula Falsi method while loop2
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  %#x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a value :')) # 2
b=float(input('Enter b value :')) # 3
N=float(input('Enter tolerance :')) # 0.001
x=a;
c=b;
i=0
while (abs(x-c)>=N):
    x=c
    c=((a*f(b)-b*f(a))/(f(b)-f(a)));
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('iteration %d \t the root %0.3f \t function value %0.3f \n'%
          (i,c,f(c)));
print('final value of the root is %0.5f'%c)

```

```

Enter the function  x**3-2*x-5
Enter a value :2
Enter b value :3
Enter tolerance :0.001
iteration 1      the root 2.059      function value -0.391

iteration 2      the root 2.081      function value -0.147

iteration 3      the root 2.090      function value -0.055

iteration 4      the root 2.093      function value -0.020

iteration 5      the root 2.094      function value -0.007

iteration 6      the root 2.094      function value -0.003

final value of the root is 2.09431

```

### 6.3 Newton-Raphson method to solve a transcendental equation

Find a root of the equation  $3x = \cos x + 1$ , near 1, by Newton Raphson method. Perform 5 iterations

```

from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  %#3x-cos(x)-1;    %function
f=lambdify(x,g)
dg = diff(g);

```

```

df=lambdify(x,dg)
x0= float(input('Enter the intial approximation  ')); # x0=1
n= int(input('Enter the number of iterations  ')); #n=5;
for i in range(1,n+1):
    x1 =(x0 - (f(x0)/df(x0)))
    print('itration %d \t the root %0.3f \t function value %0.3f \n'%
          (i, x1,f(x1))); #print all
                             iteration value

x0 = x1

```

```

Enter the function  3*x-cos(x)-1
Enter the initial approximation  1
Enter the number of iterations  5
itration 1          the root 0.620          function value 0.046

itration 2          the root 0.607          function value 0.000

itration 3          the root 0.607          function value 0.000

itration 4          the root 0.607          function value 0.000

itration 5          the root 0.607          function value 0.000

```

## 6.4 Exercise:

1. Find a root of the equation  $3x = \cos x + 1$ , between 0 and 1, by Regula-falsi method. Perform 5 iterations.  
Ans: 0.607
2. Find a root of the equation  $xe^x = 2$ , between 0 and 1, by Regula-falsi method. Correct to 3 decimal places.  
Ans: 0.853
3. Obtain a real positive root of  $x^4 - x = 0$ , near 1, by Newton-Raphson method. Perform 4 iterations.  
Ans: 1.856
4. Obtain a real positive root of  $x^4 + x^3 - 7x^2 - x + 5 = 0$ , near 3, by Newton-Raphson method. Perform 7 iterations.  
Ans: 2.061

# LAB 7: Interpolation /Extrapolation using Newton's forward and backward difference formula

## 7.1 Objectives:

Use python

1. to interpolate using Newton's Forward interpolation method.
  2. to interpolate using Newton's backward interpolation method.
  3. to extrapolate using Newton's backward interpolation method.
1. Use Newtons forward interpolation to obtain the interpolating polynomial and hence calculate  $y(2)$  for the following:
- |    |   |    |    |     |     |
|----|---|----|----|-----|-----|
| x: | 1 | 3  | 5  | 7   | 9   |
| y: | 6 | 10 | 62 | 210 | 502 |

```
from sympy import *
import numpy as np
n = int(input('Enter number of data points: '))
210
x = np.zeros((n))
y = np.zeros((n,n))

# Reading data points
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+'']='))
    y[i][0] = float(input( 'y['+str(i)+'']='))

# Generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i] = y[j+1][i-1] - y[j][i-1]

print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, n-i):
        print('\t\t%0.2f' %(y[i][j]), end='')
    print()

# obtaining the polynomial
t=symbols('t')
f=[] # f is a list type data

p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
```

```

simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ') # y
if inter=='y':
    a=float(input('enter the point ')) #2
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe value of the function at' ,a,'is\n',result);

```

Enter number of data points: 5

Enter data for x and y:

```

x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

```

FORWARD DIFFERENCE TABLE

1.00	6.00	4.00	48.00	48.00	0.00
3.00	10.00	52.00	96.00	48.00	
5.00	62.00	148.00	144.00		
7.00	210.00	292.00			
9.00	502.00				

THE INTERPOLATING POLYNOMIAL IS

```

      3      2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 2

```

The value of the function at 2.0 is  
5.0

2. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate  $y(8)$  for the following data:
- |    |   |    |    |     |     |
|----|---|----|----|-----|-----|
| x: | 1 | 3  | 5  | 7   | 9   |
| y: | 6 | 10 | 62 | 210 | 502 |

```

from sympy import *
import numpy as np
import sys
print("This will use Newton's backward interpolation formula ")
# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))

# Reading data points

```

```

print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+'']='))
    y[i][0] = float(input( 'y['+str(i)+'']='))

# Generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-2,-1):
        y[j][i] = y[j][i-1] - y[j-1][i-1]

print('\nBACKWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, i+1):
        print('\t%0.2f' %(y[i][j]), end='')
    print()

# obtaining the polynomial
t=symbols('t')
f=[]

p=(t-x[n-1])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p+i)/(i+1))

poly=y[n-1][0]
print(poly)
for i in range(n-1):
    poly=poly+y[n-1][i+1]*f[i]
    simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
    a=float(input('enter the point '))
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe value of the function at' ,a,'is\n',result);

```

```

This will use Newton's backward interpolation formula
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

```

BACKWARD DIFFERENCE TABLE

1.00	6.00				
3.00	10.00	4.00			
5.00	62.00	52.00	48.00		
7.00	210.00	148.00	96.00	48.00	
9.00	502.00	292.00	144.00	48.00	0.00
502.0					

THE INTERPOLATING POLYNOMIAL IS

```

      3      2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 8

```

```

The value of the function at 8.0 is
335.0

```

## 7.2 Exercise:

1. Obtain the interpolating polynomial for the following data

```

x:  0  1  2  3
y:  1  2  1 10

```

Ans:  $2x^3 - 7x^2 + 6x + 1$

2. Find the number of men getting wage Rs. 100 from the following table:

```

wage:      50  150  250  350
No. of men:  9   30   35   42

```

Ans: 23 men

3. Using Newton's backward interpolation method obtain  $y(160)$  for the following data

```

x :  100  150  200  250  300
y :   10   13   15   17   18

```

Ans: 13.42

4. Using Newtons forward interpolation polynomial and calculate  $y(1)$  and  $y(10)$ .

```

x :   3    4    5    6    7    8    9
y :  4.8  8.4 14.5 23.6 36.2 52.8 73.9

```

Ans: 3.1 and 100

# LAB 8: Computation of area under the curve using Trapezoidal, Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ and Simpsons $\left(\frac{3}{8}\right)^{\text{th}}$ rule

## 8.1 Objectives:

Use python

1. to find area under the curve represented by a given function using Trapezoidal rule.
2. to find area under the curve represented by a given function using Simpson's  $\left(\frac{1}{3}\right)^{\text{rd}}$  rule.
3. to find area under the curve represented by a given function using Simpson's  $\left(\frac{3}{8}\right)^{\text{th}}$  rule.
4. to find the area below the curve when discrete points on the curve are given.

## 8.2 Trapezoidal Rule

Evaluate  $\int_0^5 \frac{1}{1+x^2}$ .

```
# Definition of the function to integrate
```

```
def my_func(x):  
    return 1 / (1 + x ** 2)
```

```
# Function to implement trapezoidal method
```

```
def trapezoidal(x0, xn, n):  
    h = (xn - x0) / n  
    # Calculating step size  
  
    # Finding sum  
    integration = my_func(x0) + my_func(xn) # Adding first and last terms  
  
    for i in range(1, n):  
        k = x0 + i * h # i-th step value  
        integration = integration + 2 * my_func(k) # Adding areas of the trapezoids  
  
    # Proportioning sum of trapezoid areas  
    integration = integration * h / 2  
    return integration
```

```
# Input section
```

```
lower_limit = float(input("Enter lower limit of integration: "))  
upper_limit = float(input("Enter upper limit of integration: "))  
sub_interval = int(input("Enter number of sub intervals: "))
```

```
# Call trapezoidal() method and get result
```

```
result = trapezoidal(lower_limit, upper_limit, sub_interval)
```

```
# Print result
```

```
print("Integration result by Trapezoidal method is: " , result)
```



```

Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 10
Integration result by Trapezoidal method is: 1.3731040812301099

```

### 8.3 Simpson's $(\frac{1}{3})^{\text{rd}}$ Rule

Evaluate  $\int_0^5 \frac{1}{1+x^2}$ .

```

# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)

# Function to implement the Simpson's one-third rule

def simpson13(x0,xn,n):
    h = (xn - x0) / n                # calculating step size
    # Finding sum
    integration = (my_func(x0) + my_func(xn))
    k = x0
    for i in range(1,n):
        if i%2 == 0:
            integration = integration + 4 * my_func(k)
        else:
            integration = integration + 2 * my_func(k)
        k += h
    # Finding final integration value
    integration = integration * h * (1/3)
    return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result))

```

```

Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 100
Integration result by Simpson's 1/3 method is: 1.404120

```

### 8.4 Simpson's 3/8th rule

Evaluate  $\int_0^6 \frac{1}{1+x^2} dx$  using Simpson's 3/8 th rule, taking 6 sub intervals

```

def simpsons_3_8_rule(f, a, b, n):

```

```

h = (b - a) / n
s = f(a) + f(b)
for i in range(1, n, 3):
    s += 3 * f(a + i * h)
for i in range(3, n-1, 3):
    s += 3 * f(a + i * h)
for i in range(2, n-2, 3):
    s += 2 * f(a + i * h)
return s * 3 * h / 8

def f(x):
    return 1/(1+x**2) # function here

a = 0 # lower limit
b = 6 # upper limit
n = 6 # number of sub intervals

result = simpsons_3_8_rule(f, a, b, n)
print('%3.5f'%result)

```

1.27631

## 8.5 Exercise:

1. Evaluate the integral  $\int_0^1 \frac{x^2}{1+x^3} dx$  using Simpson's  $\frac{1}{3}$  rule.

Ans: 0.23108

2. Use Simpson's  $\frac{3}{8}$  rule to find  $\int_0^{0.6} e^{-x^2} dx$  by taking seven ordinates.

Ans: 0.5351

3. Evaluate using trapezoidal rule  $\int_0^{\pi} \sin^2 x dx$ . Take  $n = 6$ .

Ans:  $\pi/2$

4. A solid of revolution is formed by rotating about the  $x$ -axis, the area between the  $x$ -axis, the lines  $x = 0$  and  $x = 1$ , and a curve through the points with the following co-ordinates:

x	y
0.00	1.0000
0.25	0.9896
0.50	0.9589
0.75	0.9089
1.00	0.8415

Estimate the volume of the solid formed using Simpson's  $\frac{1}{3}$ rd rule. Hint: Required volume is  $\int_0^1 y^2 * \pi dx$ . **\*\*[Ans: 2.8192]\*\***

5. The velocity  $v$ (km/min) of a moped which starts from rest, is given at fixed intervals of time  $t$ (min) as follows:

t:	2	4	6	8	10	12	14	16	18	20
v:	10	18	25	29	32	20	11	5	2	0

Estimate approximately the distance covered in twenty minutes.

#### Answer for 5.

We know that  $ds/dt=v$ . So to get distance (s) we have to integrate.

Here  $h = 2.2$ ,  $v_0 = 0$ ,  $v_1 = 10$ ,  $v_2 = 18$ ,  $v_3 = 25$  etc.

```
# we shall use simpson's 1/3 rule directly to estimate

h=2
y= [0, 10 ,18, 25, 29,32 ,20, 11 ,5 ,2 , 0]
result=(h/3)*((y[0]+y[10])+4*(y[1]+y[3]+y[5]+y[7]+y[9])+2*(y[2]+y[4]+y[6]+y[8]))

print('%3.5f'%result,'km.')
```

309.33333 km.

# LAB 9: Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method

## 9.1 Objectives:

Use python

1. to solve ODE by Taylor series method.
2. to solve ODE by Modified Euler method.
3. to trace the solution curves.

## 9.2 Taylor series method to solve ODE

Solve:  $\frac{dy}{dx} - 2y = 3e^x$  with  $y(0) = 0$  using Taylor series method at  $x = 0.1(0.1)0.3$ .

```
## module taylor
'''X,Y = taylor(deriv,x,y,xStop,h).
4th-order Taylor series method for solving the initial value problem {y
}' = {F(x,{y})}}, where

{y} = {y[0],y[1],...y[n-1]}.
x,y = initial conditions
xStop = terminal value of x
h = increment of x
'''

from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:
        D = deriv(x,y)
        H = 1.0
        for j in range(3):
            H = H*h/(j + 1)
            y = y + D[j]*H
        x = x + h
        X.append(x)
        Y.append(y)

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
# array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y''[0] y''[1] y''[2] ... y''[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y''''[0] y''''[1] y''''[2] ... y''''[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
```

```

D[0] = [2*y[0] + 3*exp(x)]
D[1] = [4*y[0]+ 9*exp(x)]
D[2] = [8*y[0]+ 21*exp(x)]
D[3] = [16*y[0]+ 45*exp(x)]
return D

x = 0.0          # Initial value of x
xStop = 0.3      # last value
y = array([0.0]) # Initial values of y
h = 0.1          # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
      x = %0.2f, y=%0.5f,x = %0.2f, y=%0
      .5f"%(X[0],Y[0],X[1],Y[1],X[2],Y[2]
      ,X[3],Y[3] ))

```

The required values are :at x= 0.00, y=0.00000, x=0.10, y=0.34850,  
x = 0.20, y=0.81079,x = 0.30, y=1.41590

Solve  $y' + 4y = x^2$  with initial conditions  $y(0) = 1$  using Taylor series method at  $x = 0.1, 0.2$ .

```

from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:
        D = deriv(x,y)
        H = 1.0
        for j in range(3):
            H = H*h/(j + 1)
            y = y + D[j]*H
            x = x + h
        X.append(x)
        Y.append(y)

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
# array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y''[0] y''[1] y''[2] ... y''[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y''''[0] y''''[1] y''''[2] ... y''''[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
    D[0] = [x**2-4*y[0]]
    D[1] = [2*x-4*x**2+16*y[0]]
    D[2] = [2-8*x+16*x**2-64*y[0]]
    D[3] = [-8+32*x-64*x**2+256*y[0]]

```

```

    return D

x = 0.0          # Initial value of x
xStop = 0.2      # last value
y = array([1.0]) # Initial values of y
h = 0.1         # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
      x = %0.2f, y=%0.5f"%(X[0],Y[0],X[1],Y[1],X[2],Y[2]))

```

The required values are :at x= 0.00, y=1.00000, x=0.10, y=0.66967,  
x = 0.20, y=0.45026

### 9.3 Euler's method to solve ODE:

To solve the ODE of the form  $\frac{dy}{dx} = f(x, y)$  with initial conditions  $y(x_0) = y_0$ . The iterative formula is given by :  $y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i))$ .

Solve:  $y' = e^{-x}$  with  $y(0) = -1$  using Euler's method at  $x = 0.2(0.2)0.6$ .

```

import numpy as np
import matplotlib.pyplot as plt

# Define parameters
f = lambda x, y: np.exp(-x) # ODE
h = 0.2 # Step size
y0 = -1 # Initial Condition
n=3
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])

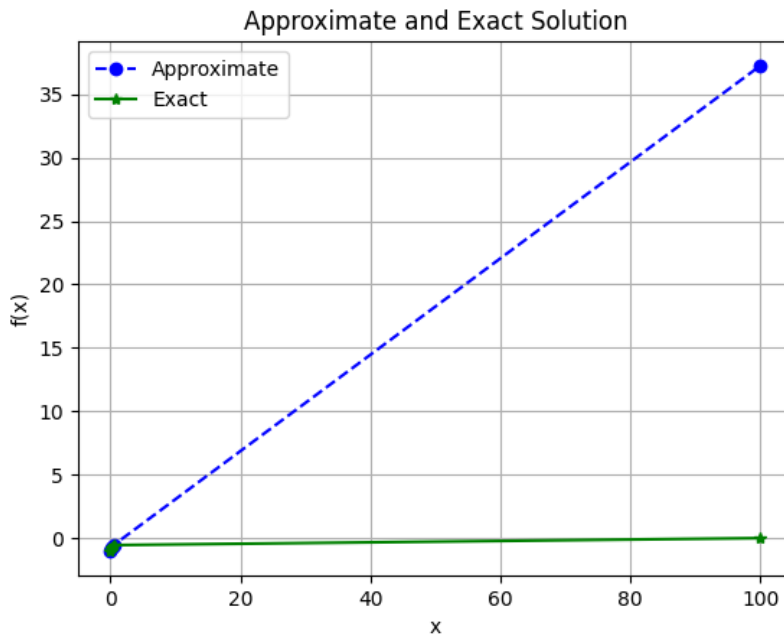
print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
      x = %0.2f, y=%0.5f,x = %0.2f, y=%0.5f"%(x[0],y[0],x[1],y[1],x[2],y[2],
      x[3],y[3]))

print("\n\n")

plt.plot(x, y, 'bo--', label='Approximate')
plt.plot(x, -np.exp(-x), 'g*-', label='Exact')
plt.title("Approximate and Exact Solution" )
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()

```

The required values are at  $x = 0.00$ ,  $y = -1.00000$ ,  $x = 0.20$ ,  $y = -0.80000$ ,  
 $x = 0.40$ ,  $y = -0.63625$ ,  $x = 0.60$ ,  $y = -0.50219$



Solve:  $y' = -2y + x^3e^{-2x}$  with  $y(0) = 1$  using Euler's method at  $x = 0.1, 0.2$ .

```
import numpy as np
import matplotlib.pyplot as plt

# Define parameters
f = lambda x, y: -2*y+(x**3)*np.exp(-2*x) # ODE
h = 0.1 # Step size
y0 = 1 # Initial Condition
n=2
# Explicit Euler Method

y[0] = y0
x[0]=0

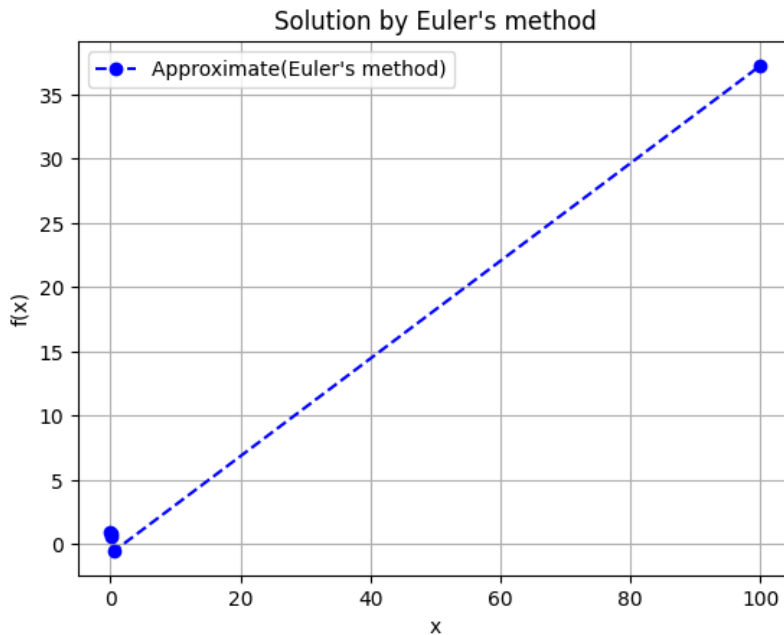
for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])

print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f, x\n\n"
      =%0.2f, y=%0.5f\n\n"%(x[0],y[0],x[1],y[1],x[2],y[2]))

plt.plot(x, y, 'bo--', label="Approximate(Euler's method)")

plt.title("Solution by Euler's method")
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

The required values are at  $x=0.00$ ,  $y=1.00000$ ,  $x=0.10$ ,  $y=0.80000$ ,  $x=0.20$ ,  $y=0.64008$



## 9.4 Modified Euler's method

The iterative formula is:

$$y_1^{(n+1)} = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{(n)})], \quad n = 0, 1, 2, 3, \dots,$$

where,  $y_1^{(n)}$  is the  $n^{th}$  approximation to  $y_1$ .

The first iteration will use Euler's method:  $y_1^{(0)} = y_0 + hf(x_0, y_0)$ .

Solve  $y' = -ky$  with  $y(0) = 100$  using modified Euler's method at  $x = 100$ , by taking  $h = 25$ .

```
import numpy as np
import matplotlib.pyplot as plt

def modified_euler(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)

    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i+1] = x[i] + h
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i+1], y[i] + k1)
        y[i+1] = y[i] + 0.5 * (k1 + k2)

    return x, y
```



```

def f(x, y):
    return -0.01 * y          # ODE dy/dx = -ky

x0 = 0.0
y0 = 100.0
h = 25
n = 4

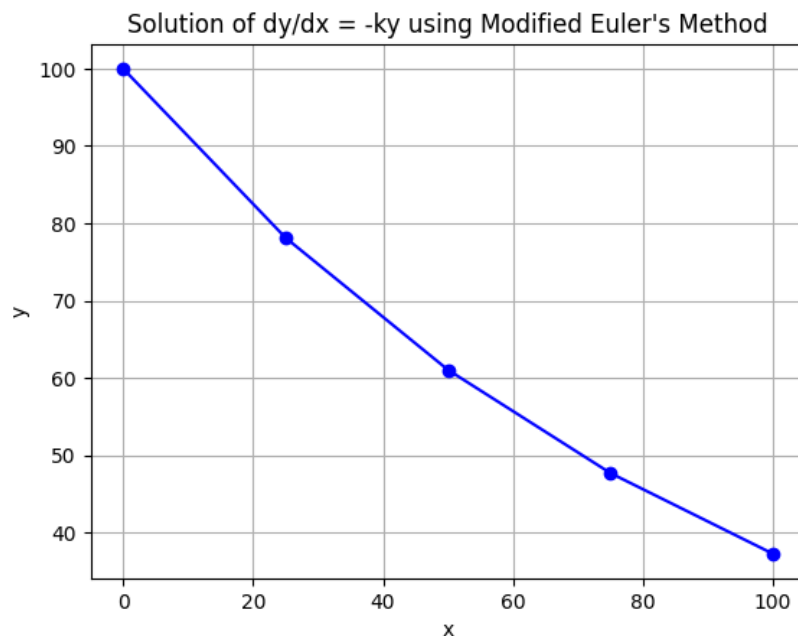
x, y = modified_euler(f, x0, y0, h, n)

print("The required value at x= %0.2f, y=%0.5f"%(x[4],y[4]))
print("\n\n")

# Plotting the results
plt.plot(x, y, 'bo-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx = -ky using Modified Euler\'s Method')
plt.grid(True)
plt.show()

```

The required value at x= 100.00, y=37.25290



## 9.5 Exercise:

- Find  $y(0.1)$  by Taylor Series expansion when  $y' = x - y^2$ ,  $y(0) = 1$ .  
Ans:  $y(0.1) = 0.9138$
- Find  $y(0.2)$  by Taylor Series expansion when  $y' = x^2y - 1$ ,  $y(0) = 1$ ,  $h = 0.1$ .  
Ans:  $y(0.2) = 0.80227$

3. Evaluate by modified Euler's method:  $y' = \ln(x + y)$ ,  $y(0) = 2$  at  $x = 0(0.2)0.8$ .

Ans: 2.0656, 2.1416, 2.2272, 2.3217

4. Solve by modified Euler's method:  $y' = x + y$ ,  $y(0) = 1$ ,  $h = 0.1$ ,  $x = 0(0.1)0.3$ .

Ans: 1.1105, 1.2432, 1.4004

# LAB 10: Solution of ODE of first order and first degree by Runge-Kutta 4th order method and Milne's predictor and corrector method

## 10.1 Objectives:

1. To write a python program to solve first order differential equation using 4th order Runge Kutta method.
2. To write a python program to solve first order differential equation using Milne's predictor and corrector method.

## 10.2 Runge-Kutta method

Apply the Runge Kutta method to find the solution of  $dy/dx = 1 + (y/x)$  at  $y(2)$  taking  $h = 0.2$ . Given that  $y(1) = 2$ .

```
from sympy import *
import numpy as np
def RungeKutta(g,x0,h,y0,xn):

    x,y=symbols('x,y')
    f=lambdify([x,y],g)
    xt=x0+h
    Y=[y0]
    while xt<=xn:
        k1=h*f(x0,y0)
        k2=h*f(x0+h/2, y0+k1/2)
        k3=h*f(x0+h/2, y0+k2/2)
        k4=h*f(x0+h, y0+k3)
        y1=y0+(1/6)*(k1+2*k2+2*k3+k4)
        Y.append(y1)
        #print('y(%3.3f'%xt,') is %3.3f'%y1)
        x0=xt
        y0=y1
        xt=xt+h
    return np.round(Y,2)
RungeKutta('1+(y/x)',1,0.2,2,2)
```

array([2. , 2.62, 3.27, 3.95, 4.66, 5.39])

## 10.3 Milne's predictor and corrector method

Apply Milne's predictor and corrector method to solve  $dy/dx = x^2 + (y/2)$  at  $y(1.4)$ . Given that  $y(1)=2$ ,  $y(1.1)=2.2156$ ,  $y(1.2)=2.4649$ ,  $y(1.3)=2.7514$ . Use corrector formula thrice.

```
# Milne's method to solve first order DE
# Use corrector formula thrice
x0=1
y0=2
```

```

y1=2.2156
y2=2.4649
y3=2.7514
h=0.1
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
def f(x,y):
    return x**2+(y/2)

y10=f(x0, y0)
y11=f(x1,y1)
y12=f(x2,y2)
y13=f(x3,y3)
y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
print('predicted value of y4 is %3.3f'%y4p)
y14=f(x4,y4p);
for i in range(1,4):
    y4=y2+(h/3)*(y14+4*y13+y12);
    print('corrected value of y4 after \t iteration \t %d is \t %3.5f\t '%
          (i,y4))
    y14=f(x4,y4);

```

predicted value of y4 is 3.079

corrected value of y4 after	iteration 1 is	3.07940
corrected value of y4 after	iteration 2 is	3.07940
corrected value of y4 after	iteration 3 is	3.07940

In the next program, function will take all the inputs from the user and display the answer.

Apply Milne's predictor and corrector method to solve  $dy/dx = x^2 + (y/2)$  at  $y(1.4)$ . Given that  $y(1)=2$ ,  $y(1.1)=2.2156$ ,  $y(1.2)=2.4649$ ,  $y(1.3)=2.7514$ . Use corrector formula thrice.

```

from sympy import *
def Milne(g,x0,h,y0,y1,y2,y3):
    x,y=symbols('x,y')
    f=lambdify([x,y],g)
    x1=x0+h
    x2=x1+h
    x3=x2+h
    x4=x3+h

    y10=f(x0, y0)
    y11=f(x1,y1)
    y12=f(x2,y2)
    y13=f(x3,y3)
    y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
    print('predicted value of y4',y4p)
    y14=f(x4,y4p)
    for i in range(1,4):
        y4=y2+(h/3)*(y14+4*y13+y12)
        print('corrected value of y4 , iteration \t %d \t '%i,y4)

```

```

        y14=f(x4,y4)
Milne('x**2+y/2',1,0.1,2,2.2156,2.4649,2.7514)

```

predicted value of y4 3.0792733333333335  
 corrected value of y4 , iteration 1 3.0793962222222224  
 corrected value of y4 , iteration 2 3.079398270370371  
 corrected value of y4 , iteration 3 3.079398304506173

Apply Milne's predictor and corrector method to solve  $dy/dx = x - y^2$ ,  $y(0)=2$  obtain  $y(0.8)$ . Take  $h=0.2$ . Use Runge-Kutta method to calculate required initial values.

```

Y=RungeKutta('x-y**2',0,0.2,0,0.8)
print('y values from Runge -Kutta method:',Y)
Milne('x-y**2',0,0.2,Y[0],Y[1],Y[2],Y[3])

```

y values from Runge -Kutta method: [0. 0.02 0.08 0.18 0.3 ]  
 predicted value of y4 0.3042133333333334  
 corrected value of y4 , iteration 1 0.3047636165214815  
 corrected value of y4 , iteration 2 0.3047412758696499  
 corrected value of y4 , iteration 3 0.3047421836520892

## 10.4 Exercise:

- Find  $y(0.1)$  by Runge Kutta method when  $y' = x - y^2$ ,  $y(0) = 1$ .  
Ans:  $y(0.1) = 0.91379$
- Evaluate by Runge Kutta method :  $y' = \log(x + y)$ ,  $y(0) = 2$  at  $x = 0(0.2)0.8$ .  
Ans: 2.155, 2.3418, 2.557, 2.801
- Solve by Milnes method:  $y' = x + y$ ,  $y(0)=1$ ,  $h=0.1$ , Calculate  $y(0.4)$  . Calculate required initial values from Runge Kutta method.  
Ans: 1.583649219