**Lab Session #5**

**Linear and Quadratic Equations in Python**

**Aim:**

To explore various functions available in Numpy and CMath modules in python and solve Linear and Quadratic equations using NumPy, Arrays and CMath (Complex Math Module).

**Problem Definition**:

Develop Python Programs using NumPy and Arrays for following:

1. Solve linear equations with one variable.
2. Solve linear equations with three variables.
3. Solve linear equations taken from user (multiple variables).

Develop Python Programs using CMath for following

1. Solve a quadratic equation.
2. Solve a quadratic equation taken from user.

**Theory:**

**CMath** - Mathematical functions for complex numbers.

This module provides access to mathematical functions for complex numbers. The functions in this module accept integers, floating-point numbers or complex numbers as arguments. They will also accept any Python object that has either a \_\_complex\_\_ () or a \_\_float\_\_ () method: these methods are used to convert the object to a complex or floating-point number, respectively, and the function is then applied to the result of the conversion.

A Python complex number z is stored internally using rectangular or Cartesian coordinates. It is completely determined by its real part z.real and its imaginary part z.imag.

**Calculate phase of a complex number**

The phase of a complex number is defined as the angle between the real axis and the vector representing the complex number. Using the cmath module, we can find the phase of a complex number using the phase () method. The phase method takes a complex number as input and returns a floating point number representing the phase of the complex number

**Polar coordinates of a complex number**

In polar coordinates, a complex number is defined as a tuple consisting of the modulus of the complex number as the first element and phase of the complex number as the second element. We can find the polar coordinates of a complex number using polar () method in python. The polar () method takes a complex number as input and returns a tuple representing the polar coordinates

If we know the modulus and phase of a complex number i.e. If we know the polar coordinates of a complex number, we can obtain the complex number using the rect () method. The rect () method takes the modulus as the first argument and phase of the complex number as the second argument and returns the corresponding complex number.

**Constants**

The cmath module also provides certain mathematical constants like infinity, NaN and pi which are useful in mathematical calculations.

**Trigonometric functions**

For mathematical calculations on complex numbers, the cmath module provides a set of trigonometric functions. All of the trigonometric functions take the complex number as input and also return a complex number which represents the corresponding output for the trigonometric functions.

**Hyperbolic functions**

Just like trigonometric functions, cmath module also provides hyperbolic functions and inverse hyperbolic trigonometric functions for mathematical calculations in python. All these functions take a complex number as input and return a complex number representing the hyperbolic or inverse hyperbolic trigonometric output as per their nature.

**Logarithmic functions**

The cmath module provides two methods namely log () and log10 () for logarithmic calculations on complex numbers. The log () function takes a complex number as first input and an optional argument representing the base of the logarithmic function. When we pass only the complex number as input to the log () function, it returns the natural log of the complex number with base “e”. When we also pass the second argument i.e. base to the log () function, it calculates the logarithm of the complex number with the provided base.

**Power functions**

The cmath module provides two power functions namely exp () and sqrt () for calculations in python. The exp () function takes a complex number as input and returns a complex number representing the exponential value of the input.

The sqrt() function also takes a complex number as input and returns the complex number representing the square root of the input.

**1.**

**Program:**

import numpy as np

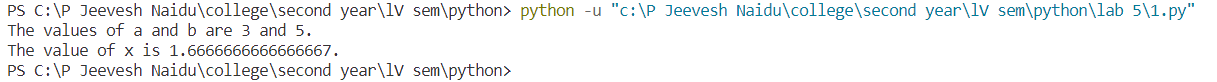
a, b = 3, 5

print(f"The values of a and b are {a} and {b}.")

x = np.divide(b, a)

print(f"The value of x is {x}.")

**Output:**

****

**2.**

**Program:**

import numpy as np

A = np.array([[2, -4, 4], [34, 3, -1], [1, 1, 1]])

B = np.array(([8, 30, 108]))

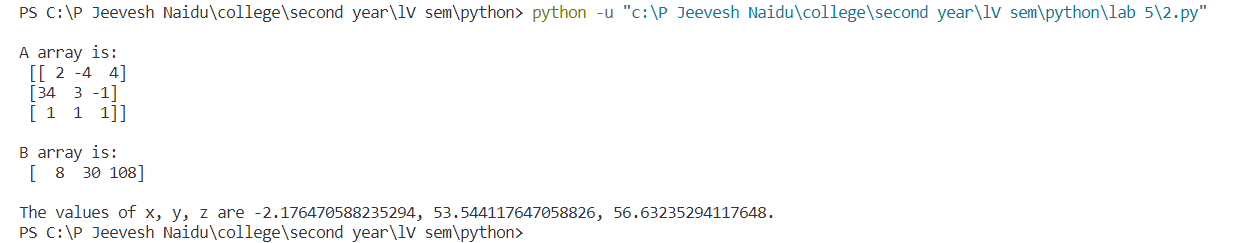
print("\nA array is:\n", A)

print("\nB array is:\n", B)

X = np.linalg.solve(A, B)

print(f"\nThe values of x, y, z are {X[0]}, {X[1]}, {X[2]}.")

**Ouput:**

****

**3.**

**Program:**

import numpy as np

n = int(input("Enter the number of variables."))

i = 0

print("Enter values in the form a1x + a2y + a3z + a4w + ... = b")

equ = []

while (i < n):

    i = i+1

    temp = np.array([int(i) for i in input(f"Enter equation {i}:").split()])

    equ.append(temp)

A = np.array([equ[:-1] for equ in equ])

B = np.array([equ[-1] for equ in equ])

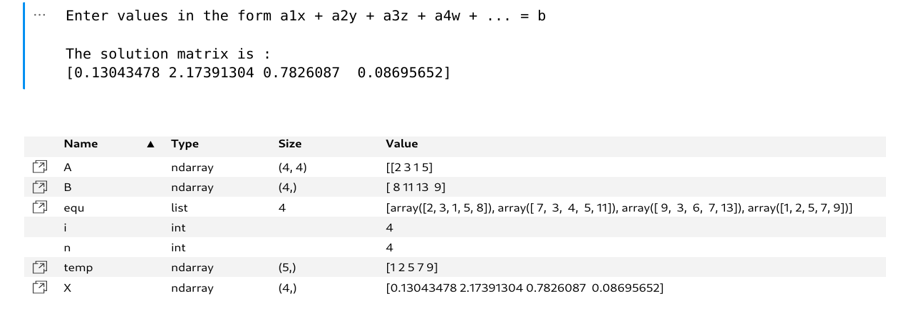
print()

X = np.linalg.solve(A, B)

print("The solution matrix is :")

print(X)

**Output:**

****

**4.**

**Program:**

import cmath

a, b, c = 1, 2, 7

print(f"Values of a, b, c are {a}, {b}, {c}.")

discr = (b\*\*2) - (4\*a\*c)

root1 = ((-b) + cmath.sqrt(discr)) / (2 \* a)

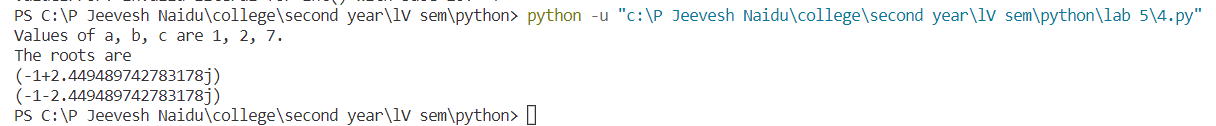
root2 = ((-b) - cmath.sqrt(discr)) / (2 \* a)

print('The roots are')

print(root1)

print(root2)

**Output:**

****

**5.**

**Program:**

import cmath

a, b, c = map(int, input("Enter the values of a,b,c.").split())

discr = (b\*\*2) - (4\*a\*c)

root1 = ((-b) + cmath.sqrt(discr)) / (2 \* a)

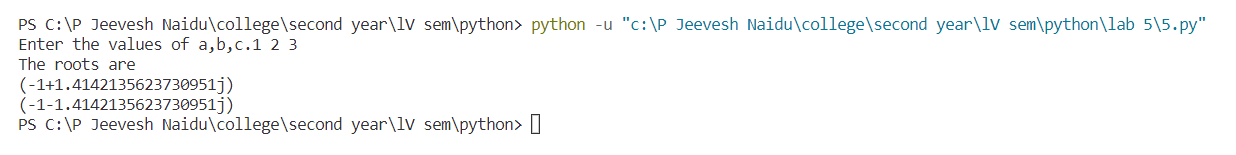
root2 = ((-b) - cmath.sqrt(discr)) / (2 \* a)

print('The roots are')

print(root1)

print(root2)

**Output:**

****