

**CHRIST (DEEMED TO BE UNIVERSITY)**

DEPARTMENT OF MATHEMATICS

CALCULUS USING PYTHON

**MAT112-2**

E-RECORD

Submitted by:

*Name:* **Sabari K**

*Register No.:* **2340154**

12th April, 2024

**Table of Contents**

# Unit I: Fundamentals of Python Programming

1. Getting Started with Python
2. Variables
3. Data Structures
4. Functions
5. 2-D Plots
6. Scatter Plots
7. 3-D Plots
8. Vector Plots
9. Surface Plots

# Unit II: Symbolic and Numeric Computations

1. Use of Sympy and Numpy packages
2. Basic Algebraic Operations with Polynomials, Rational Functions, Trigonometric Functions, Exponential and Logarithmic Functions
3. Solving Algebraic Equations
4. Limits, Derivatives, and Functions
5. Series Expansions
6. Evaluating Taylor Series

# Unit III: Solving Differential Equations

1. First Order Differential Equations
2. Logistic Growth Model
3. Predator-Prey Model
4. SIR Model

Getting Started with Python

Variable

x=1 y=35.656222554887711 z=-325.e100

print(type(x)) print(type(y)) print(type(z))

<class 'int'>

<class 'float'>

<class 'float'>

x1=float(x) y1=int(y) print(x1) print(y1)

1.0

35

name="Hii world" print(type(name))

<class 'str'>

Write a Program to perform basic arthemetic operations of two numbers entered by the user.

n1=int(input("Enter a number:")) n2=int(input("Enter a number:")) s=n1+n2

d=n1-n2 p=n1\*n2 q=n1/n2 r=n1%n2

print("Sum= ",s) print("Difference= ",d) print("Product= ",p) print("Quotient= ",q) print("Remainder= ",r)

Enter a number: 5 Enter a number:88 Sum= 93

Difference= -83

Product= 440

Quotient= 0.056818181818181816

Remainder= 5

Write a Program to check the biggest of three numbers entered by user

a=int(input("Enter a number:")) b=int(input("Enter a number:")) c=int(input("Enter a number:")) if a>b and a>c:

print(a,"is greatest") elif b>a and b>c:

print(b,"is greatest") elif c>a and c>b:

print(c,"is greatest") else:

print("All are equal.") a=3+4j

print(a.real) print(a.imag)

Data Structure

List a=["gokul","is","dumb"] print(type(a))

#tuple

b=("maybe","its beacause","he eats too much") print(type(b))

for i in range(1,10):

print(i) #dictionary d={"gokul": "Dumb"} print(d) print(type(d))

Function

def my\_function():

print("Hello ")

def starkids(\*kids):

print("The name of the last kid is",kids[2]) starkids("Harry","Hermione","Ron")

2d plots

import numpy as np

import matplotlib.pyplot as plt x=np.linspace(1,50,100) y=np.linspace(1,25,100) z1=np.e\*\*x

z2=np.exp(y) print(x) print(y) print(z1) print(z2) print(type(x))

plt.plot(x,z1,color='red') plt.show()

5.89910195e+16 9.67697762e+16 1.58742630e+17 2.60403853e+17

4.27170487e+17 7.00737040e+17 1.14949982e+18 1.88565720e+18

3.09326109e+18 5.07423308e+18 8.32385003e+18 1.36545717e+19

2.23991695e+19 3.67439423e+19 6.02753283e+19 9.88765761e+19

1.62198657e+20 2.66073173e+20 4.36470527e+20 7.15992968e+20

1.17452588e+21 1.92671033e+21 3.16060528e+21 5.18470553e+21] [2.71828183e+00 3.46400087e+00 4.41429653e+00 5.62529127e+00 7.16850389e+00 9.13507328e+00 1.16411409e+01 1.48347099e+01

1.89043856e+01 2.40905147e+01 3.06993789e+01 3.91212840e+01

4.98536099e+01 6.35301853e+01 8.09587199e+01 1.03168506e+02

1.31471208e+02 1.67538323e+02 2.13499900e+02 2.72070332e+02

3.46708666e+02 4.41822886e+02 5.63030237e+02 7.17488971e+02

9.14321096e+02 1.16515110e+03 1.48479249e+03 1.89212260e+03

2.41119749e+03 3.07267265e+03 3.91561339e+03 4.98980203e+03

6.35867790e+03 8.10308393e+03 1.03260411e+04 1.31588325e+04

1.67687569e+04 2.13690089e+04 2.72312697e+04 3.47017520e+04

4.42216469e+04 5.63531794e+04 7.18128122e+04 9.15135588e+04

1.16618904e+05 1.48611517e+05 1.89380813e+05 2.41334543e+05

3.07540983e+05 3.91910148e+05 4.99424703e+05 6.36434232e+05

8.11030229e+05 1.03352397e+06 1.31705546e+06 1.67836948e+06

2.13880448e+06 2.72555277e+06 3.47326649e+06 4.42610403e+06

5.64033797e+06 7.18767842e+06 9.15950806e+06 1.16722790e+07

1.48743902e+07 1.89549517e+07 2.41549528e+07 3.07814946e+07

3.92259268e+07 4.99869599e+07 6.37001178e+07 8.11752708e+07

1.03444465e+08 1.31822872e+08 1.67986460e+08 2.14070976e+08

2.72798074e+08 3.47636053e+08 4.43004687e+08 5.64536248e+08

7.19408133e+08 9.16766751e+08 1.16826769e+09 1.48876406e+09

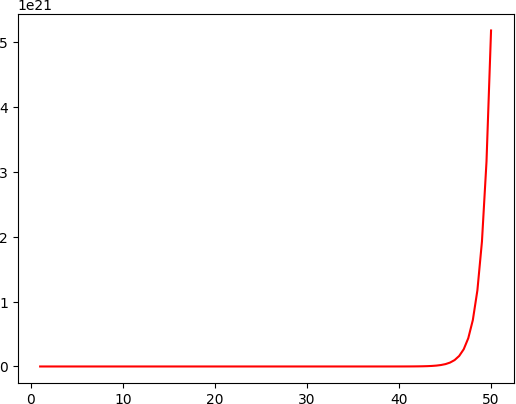
1.89718371e+09 2.41764704e+09 3.08089153e+09 3.92608699e+09

5.00314891e+09 6.37568629e+09 8.12475831e+09 1.03536615e+10

1.31940302e+10 1.68136105e+10 2.14261674e+10 2.73041087e+10

3.47945733e+10 4.43399323e+10 5.65039146e+10 7.20048993e+10]

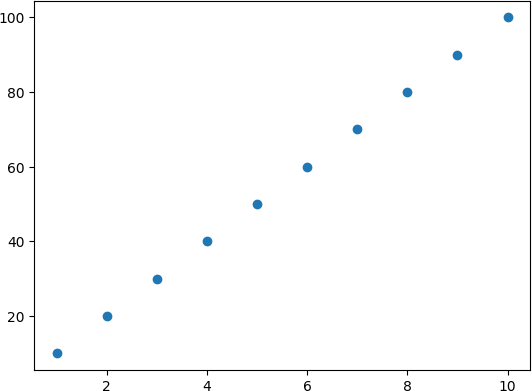
<class 'numpy.ndarray'>



x=[1,2,3,4,5,6,7,8,9,10]

y=[10,20,30,40,50,60,70,80,90,100]

plt.scatter(x,y) plt.show()



3D Plot

from mpl\_toolkits import mplot3d from matplotlib.pyplot import \* from numpy import \* x=linspace(0,50,20)

print(x) y=cos(x) z=sin(x) print(y) print(z)

ax=axes(projection="3d") ax.plot3D(x,y,z,'red')

[ 0. 2.63157895 5.26315789 7.89473684 10.52631579 13.15789474

15.78947368 18.42105263 21.05263158 23.68421053 26.31578947 28.94736842

31.57894737 34.21052632 36.84210526 39.47368421 42.10526316 44.73684211

47.36842105 50. ]

[ 1. -0.87273782 0.52334259 -0.04074393 -0.45222506 0.83009175

-0.99667986 0.90958867 -0.59098499 0.12196123 0.37810463 -0.78193365

0.98674151 -0.9403996 0.65470308 -0.20236868 -0.30147349 0.7285833

-0.97025092 0.96496603]

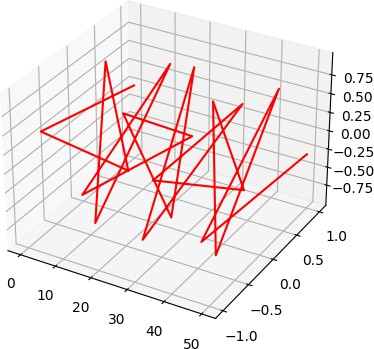
[ 0. 0.48818921 -0.85212237 0.99916962 -0.89190386 0.55762683

-0.08142019 -0.41550988 0.80668255 -0.99253487 0.92576287 -0.62336166

0.16229972 0.34007145 -0.75588615 0.97930941 -0.95347456 0.6849572

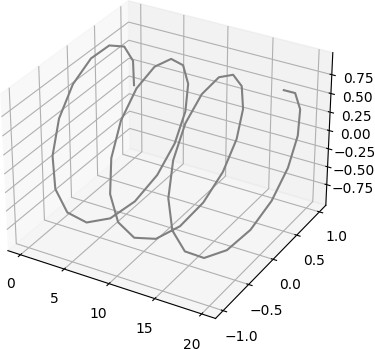
-0.24210155 -0.26237485]

[<mpl\_toolkits.mplot3d.art3d.Line3D at 0x79ee8c24c220>]



x=linspace(0,20,50) y=cos(x)

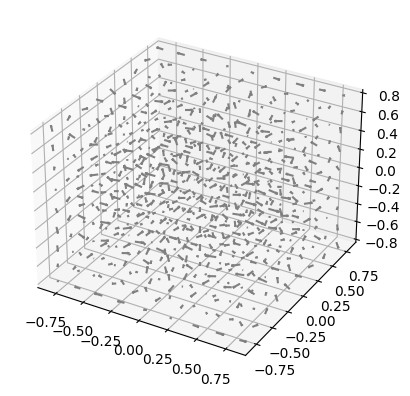
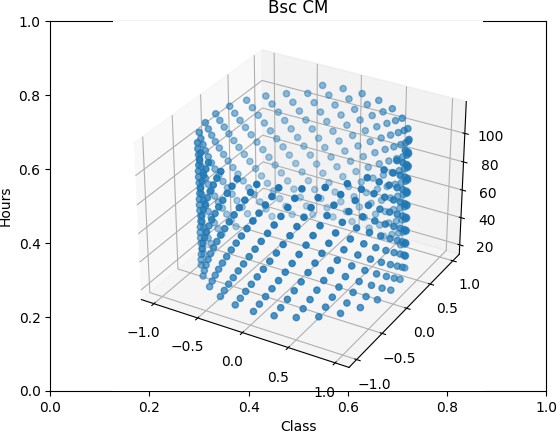
z=sin(x) ax=axes(projection="3d") ax.plot3D(x,y,z,'grey') show()



x=linspace(20,115,450) y=cos(x)

z=sin(x) title("Bsc CM") xlabel("Class") ylabel("Hours")

ax=axes(projection="3d") ax.scatter3D(y,z,x) show()



from numpy import \*

from matplotlib.pyplot import \* from mpl\_toolkits.mplot3d import \* x=arange(-0.8,1,0.2)

print(x) ax=axes(projection="3d")

x,y,z=meshgrid(arange(-0.8,1,0.2),arange(-0.8,1,0.2),arange(- 0.8,1,0.2))

u=sin(x\*pi)\*cos(y\*pi)\*cos(z\*pi) v=-cos(x\*pi)\*sin(y\*pi)\*cos(z\*pi)

w=sqrt(2/3)\*cos(x\*pi)\*cos(y\*pi)\*sin(z\*pi) ax.quiver(x,y,z,u,v,w,color='grey',length=0.1) show()

[-8.00000000e-01 -6.00000000e-01 -4.00000000e-01 -2.00000000e-01

-2.22044605e-16 2.00000000e-01 4.00000000e-01 6.00000000e-01 8.00000000e-01]



**UNIT 2 - ALGEBRA**

**Use of SymPy and NumPy package**

sympy

It is a Python library for symbolic mathematics

X=l print(x+x+4)

6

**Symbol**

To define one symbol to a variable we use **Symbol**

from sympy import Symbol

x=Symbol('x') print(x+x+4)

2\*x + 4

a=Symbol('x') print(a+a+l) a=a+l print(a)

2\*x + 1

**X** + 1

x=Symbol('x')

y=Symbol( 'y')

z=Symbol('z')

print(x,y,z)

X y Z

**symbols**

To define more than one symbol to diff variables we use **symbols**

from sympy import symbols

x,y,z=symbols('a,b,c') print(x.name) print(y.name) print(z.name)

a

b

C

The output for below code is the **symbols** not the variables

x,y,z=symbols('x,y,z') print(x.name) print(y.name) print(z.name)

**X**

y

z

## Basic algebraic operations with polynomials/rational functions



x=Symbol('x')

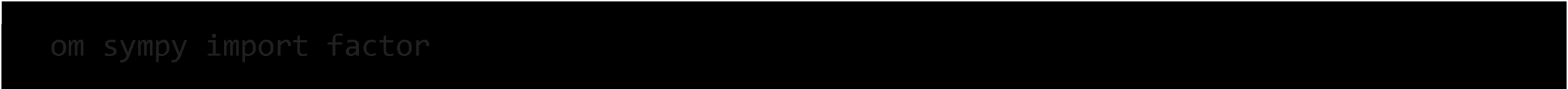
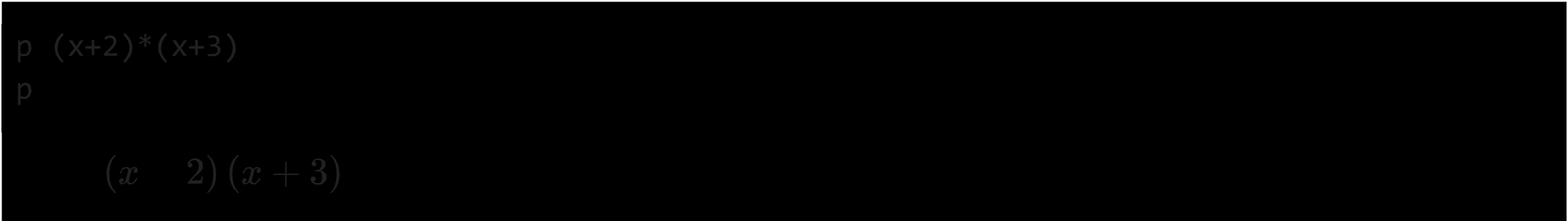
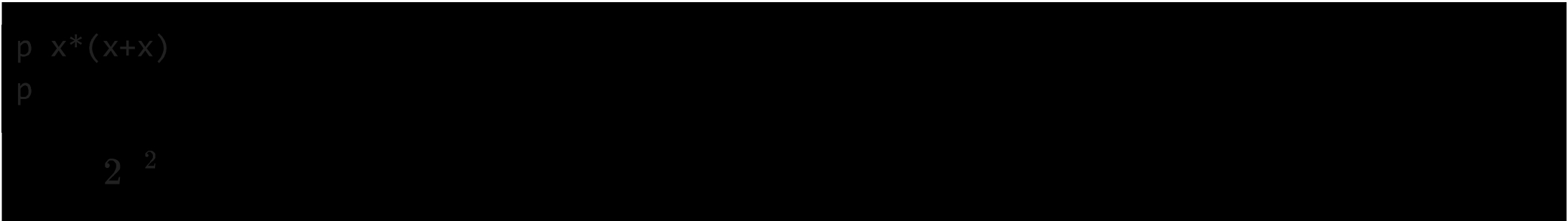
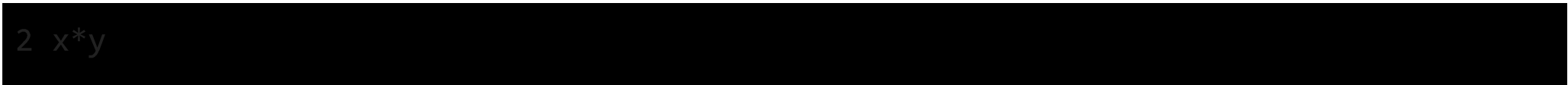
y=Symbol('y') s=x\*y+x\*y

s

*2xy*

Symbol(' X')

y=Symbol('y') s=x\*y+x\*y print(s)



**factor**

It finds the factor of the given expression

expr=x\*\*2-y\*\*2 factor(expr)

*(x-y)(x+y)*

**expand**

It is used to expand the factorised equation and viceversa depending on the equation provided

from sympy import expand

factors=factor(expr) expand(factors)



expr=x\*\*3+3\*x\*\*2\*y+3\*x\*y\*\*2+y\*\*3 factors=factor(expr)

factors



#viceversa case f=expand(factors) f



expand(f)



expr=x+y+x\*y factor(expr)

*xy* + *X* + *y*

**pprint()**

The pprint module provides a capability to "pretty-print" arbitrary Python data structures in a form which can be used as input to the interpreter.

from sympy import pprint

expr=x\*x+2\*x\*y+y\*y expr

y2 + *2xy* + x2

pprint(expr)

2 2

X + 2·X·Y + y

from sympy import init\_printing

init\_printing(order='rev-lex') pprint(expr)

2 2

y + 2·X·Y + X

**collect()**

collects common power of a term in an expression

from sympy import\*

x,y,z=symbols('x,y,z') factor\_list(x\*\*2-y\*\*2)

(1, [(-y + *x,* 1), (y + *x,* 1)])

expr=x\*y+x-3+2\*x\*\*2-z\*x\*\*2+x\*\*3 print(expr) collected\_expr=collect(expr,x) collected\_expr

x\*\*3 - x\*\*2\*z + 2\*x\*\*2 + x\*y + x - 3

-3 + *x* (1 + *y)* + x2 • (2 - *z)* + *x3*

***coeff***

collects the coefficient of given parameters inside the function. Here, the coefficient of x power 2 is printed

collected\_expr.coeff(x,2)

2-z

**cancel**

## Cancels the common terms in numerator and denominator

cancel((x\*\*2+2\*x+1)/(x\*\*2+x))

*l+x*

*X*





## Use of SymPy and NumPy package

from numpy import\* from sympy import\*

a=Rational(5,8) print(type(a))

print("The value of a is: "+str(a))

b=Integer(3.579)

print("The value of bis: "+str(b))

<class 'sympy.core.numbers.Rational'> The value of a is: 5/8

The value of bis: 3

p=pi\*\*3

print("The value of pis: "+str(p))

The value of pis: pi\*\*3

#evalf method evaluates the expression to a floating-point number q=pi.evalf()

print(q) print(type(q))

print("The value of q is: "+str(q))

3.14159265358979

<class 'sympy.core.numbers.Float'>

The value of q is: 3.14159265358979

#equivalent to e 1 or e\*\*l r=exp(l).evalf()

print("The value of r is: "+str(r))

The value of r is: 2.71828182845905

s=(pi+exp(l)).evalf()

print("The value of sis: "+str(s))

The value of sis: 5.85987448204884

## oo standes for infinity

rslt=oo+1000

print("The value of rslt is: "+str(rslt))

The value of rslt is: oo

if 00>9999999:

print("True") else:

print("False")

True

y=Symbol( 'y')

x=Symbol('x')

z=(x+y)+(x-y)

print("The value of z is: "+str(z)) The value of z is: 2\*x

## Trigonometric Simplifications

**Differentiation**

ansl=diff(sin(x)\*exp(x),x) print("Derivative of sin(x)\*e"x ",ansl)

Derivative of sin(x)\*e"x exp(x)\*sin(x) + exp(x)\*cos(x)

## Integrals

**Indefinite Integration**

#Compute(e"x\*sin(x)+e"x\*cos(x))dx ans2=integrate(exp(x)\*sin(x)+exp(x)\*cos(x),x) print("Indefinite Integration is: ",ans2)

Indefinite Integration is: exp(x)\*sin(x)

## Definite Integration

#Compute definite integral of sin(xA2)dx #in b/w interval of? and??

ans3=integrate(sin(x\*\*2),(x,-oo,oo)).evalf() print("Definite Integration is: ",ans3)

Definite Integration is: 1.25331413731550

**Limit** of a **function**

#Find the limit of sin(x) / x given x tends to 0 ans4=limit(sin(x)/x,x,0)

print("Limit is : ",ans4)

Limit is 1

#Solve Quadratic equation like, example: xA27=0 ans5=solve(x\*\*2-2,x)

print("Roots are : ",ans5)

Roots are: [-sqrt(2), sqrt(2)]

a=pi/6 b=3

C=4

**end**

removes **default** '\n'

#returning the value of tangent of pi/6

print("The value of tangent of pi/6 is : ",end="") print(tan(a))

The value of tangent of pi/6 is sqrt(3)/3

#returning the value of hypotenuse of 3 and 4

print("The value of hypotenuse of 3 and 4 is : ",end="") print(hypot(b,c))

The value of hypotenuse of 3 and 4 is 5.0

a=pi/6 b=30

#returning the converted value from radians to degree

print("The converted value from radians to degree is : ",end="") print(math.degrees(a))

The converted value from radians to degree is **30.000000000000004**

#returning the converted value from degree to radians

print("The converted value from degree to radians is : ",end="") print(math.radians(b))

The converted value from degree to radians is 0.5235987755982988

**INVERSE TRIGNOMETRY FUNCTION**

print(asin(l)) #gives sin inverse print(acos(0)) #gives cos inverse print(atan(l)) #gives tan inverse

**pi/2 pi/2 pi/4**



**X=l.0**

**y=l.0** z=complex(x,y) print(z)

print("The arc sine is : ",asin(z)) print("The arc cosine is : ",acos(z)) print("The arc tangent is : ",atan(z)) print("The hyperbolic sine is : ",sinh(z)) print("The hyperbolic cosine is : ",cosh(z)) print("The hyperbolic tangent is : ",tanh(z))

print("The inverse hyperbolic sine is : ",asinh(z)) print("The inverse hyperbolic cosine is : ",acosh(z)) print("The inverse hyperbolic tangent is : ",atanh(z))

 (  +lj)

The arc sine is 0.666239432492515 + 1.06127506190504\*1 The arc cosine is 0.904556894302381 - 1.06127506190504\*1 The arc tangent is 1.01722196789785 + 0.402359478108525\*1

The hyperbolic sine is 0.634963914784736 + 1.29845758141598\*1 The hyperbolic cosine is 0.833730025131149 + 0.988897705762865\*1 The hyperbolic tangent is 1.08392332733869 + 0.271752585319512\*1

The inverse hyperbolic sine is 1.06127506190504 + 0.666239432492515\*1 The inverse hyperbolic cosine is 1.06127506190504 + 0.904556894302381\*1 The inverse hyperbolic tangent is 0.402359478108525 + 1.01722196789785\*1



## Exponential and Logarithms functions

import math as m

## math.log()

method returns the natural logarithm of a number, or the logarithm of number to base. math.log(x, base)// default base is e

# Return the natural logarithm of different numbers print(m.log(2.7183))

print(m.log(2))

print(m.log(l))

1.0000066849139877

0.6931471805599453

0.0

import numpy as np

import matplotlib.pyplot as plt

in\_array = [1, 3, 5, 2\*\*8]

print ("Input array: ", in\_array)

out\_array = np.log(in\_array)

print ("Output array: ", out\_array)

print("\nnp.log(4\*\*4) • " np.log(4\*\*4))

print("np.log(2\*\*8) ", np.log(2\*\*8))

Input array : [1, 3, 5, 256]

Output array [0. 1.09861229 1.60943791 5.54517744]

np.log(4\*\*4) 5.545177444479562

np.log(2\*\*8) 5.545177444479562

in\_array = [1, 1.2, **1.4,** 1.6, 1.8, 2]

out\_array = np.log(in\_array)

print ("out\_array: ", out\_array) plt.plot(in\_array, in\_array,

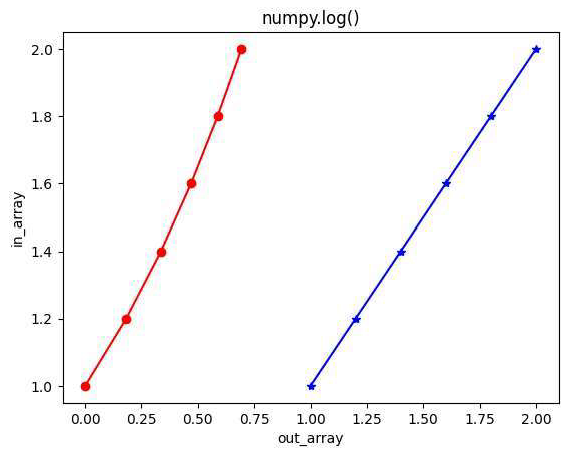
color= 'blue', marker="\*")

# red for numpy.log() plt.plot(out\_array, in\_array,

color= 'red', marker= "o")

plt.title("numpy.log()") plt.xlabel("out\_array") plt.ylabel("in\_array") plt.show()

out array [0. 0.18232156 0.33647224 0.47000363 0.58778666 0.69314718]



#base - should be mentioned after log, #if not given it is taken as e print(np.log10(100))

print(np.log(e)) print(np.log10(e))

2.0

1.0

0.4342944819032518

from sympy import\* from numpy import\*

print(log(100,10))

2

**Solving Algebraic Equations**

**Linear Equation**

**solve()**

solves the equation passed inside it solve returns a list

x=Symbol('x') solve(x+4)

[-4]

## Quadractic Equation

solve(x\*\*2+4)

[-2\*I, 2\*I]

solve(x\*\*2+2\*x+1)

[-1]

solve(x\*\*3-1)

[1, -1/2 - sqrt(3)\*I/2, -1/2 + sqrt(3)\*I/2]

print(solve(x\*\*3-1)[1])

-1/2 - sqrt(3)\*I/2

## Discriminant - D

if D>O : two diff real roots

if D<O :complex roots

if D=O :two equal real roots

x value = float(input('The x value: ')) y\_value = float(input('The y value: ')) z value= float(input('The z value: '))

discriminant= (y\_value\*\*2) - (4\*x\_value\*z\_value) if discriminant> 0:

print('Two real Solutions.') print('Discriminant value is:', discriminant)

elif discriminant== 0:

print('Two equal real Solutions.') print('Discriminant value is:", discriminant)

elif discriminant< 0:

print('Complex Solutions.') print('Discriminant value is:', discriminant)

**B** The x value: 1 The y value: 0 The z value: -1

Two real Solutions. Discriminant value is: **4.0**





from numpy import\* from sympy import\*

x=Symbol('x')

**Calculus-Limits**

**LIMIT OF A FUNCTION**

exprl =x\*\*2-4 ansl=limit(exprl,x,2) print(ansl)

0

expr2 =(x\*\*3-4\*x)/(2\*x\*\*2+3\*x) ans2=limit(expr2,x,0) print(ans2)

-4/3

expr3 =(x\*\*3)/((x+1)\*\*2) ans3=limit(expr3,x,1) print(ans3)

1/4

expr4 =(x-2)/(x\*\*2-3\*x+2) ans4=limit(expr4,x,2) print(ans4)

1

expr5=(3\*x+2\*x\*\*-1)/(x+4\*x\*\*-1) ans5=limit(expr5,x,0) print(ans5)

1/2

expr6=(x\*\*2-3\*x+2)/(x\*\*2-2\*x) ans6=limit(expr6,x,2) print(ans6)

1/2

expr7 =(x\*\*3+3\*x\*\*2+2\*x)/(x\*\*2-x-6) ans7=limit(expr7,x,2)

print(ans7)

-6

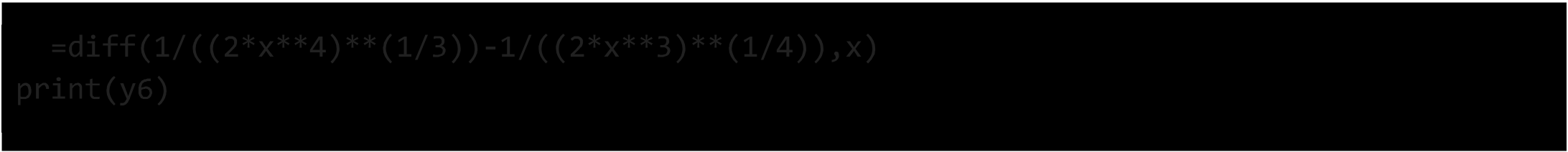
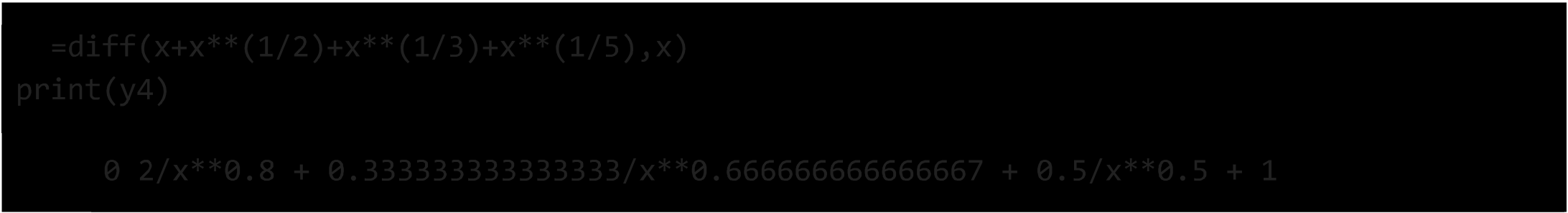
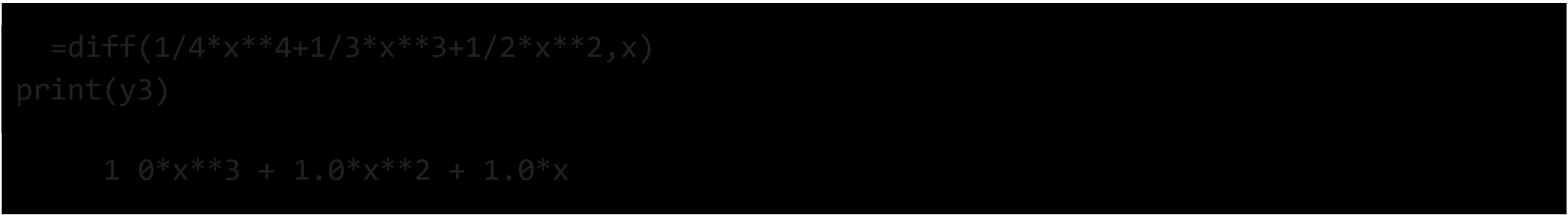
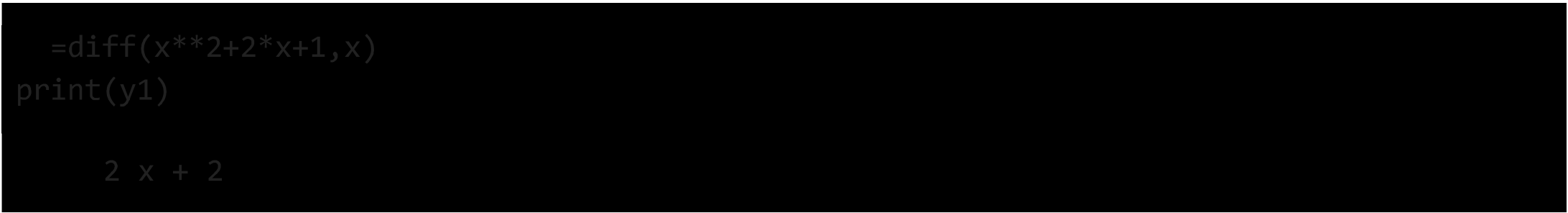
expr8 =(x\*\*2-2\*x-1)/(x\*\*3-x) ans8=limit(expr8,x,1) print(ans8)

-00

expr9 =(x\*\*2+7\*x-44)/(x\*\*2-6\*x+8) ans9=limit(expr9,x,4) print(ans9)



/2



0.630672311440286/(x\*(x\*\*3)\*\*0.25) - 1.0582673679788/(x\*(x\*\*4)\*\*0.333333333333333)

y7=diff(sin(x)+cos(x)+tan(x),x,1) print(y7)

-sin(x) + cos(x) + tan(x)\*\*2 + 1

y8=diff(log(x,10)-ln(x)+log(x,5),x) print(y8)

-1/x + 1/(x\*log(10)) + 1/(x\*log(S))

y9=diff(3\*asin(x)-2\*atan(x),x) print(y9)

S -2/(x\*\*2 + 1) + 3/sqrt(l - x\*\*2)





## Calculus-limits

from sympy import\*

x=Symbol('x') a=limit(sin(x)/x,x,0) b=limit(l/x,x,0) #default dir='+'

c=limit(l/x,x,0,dir='-') #dir='-' ----> **LHL** d=limit(l/x,x,0,dir='+-') e=limit(l/x,x,oo)

print(a,b,c,d,e)

1 00 -00 zoo 0 type(zoo)

sympy.core.numbers.Complexlnfinity

## Series Expansion POWER SERIES

#Macluarian's series



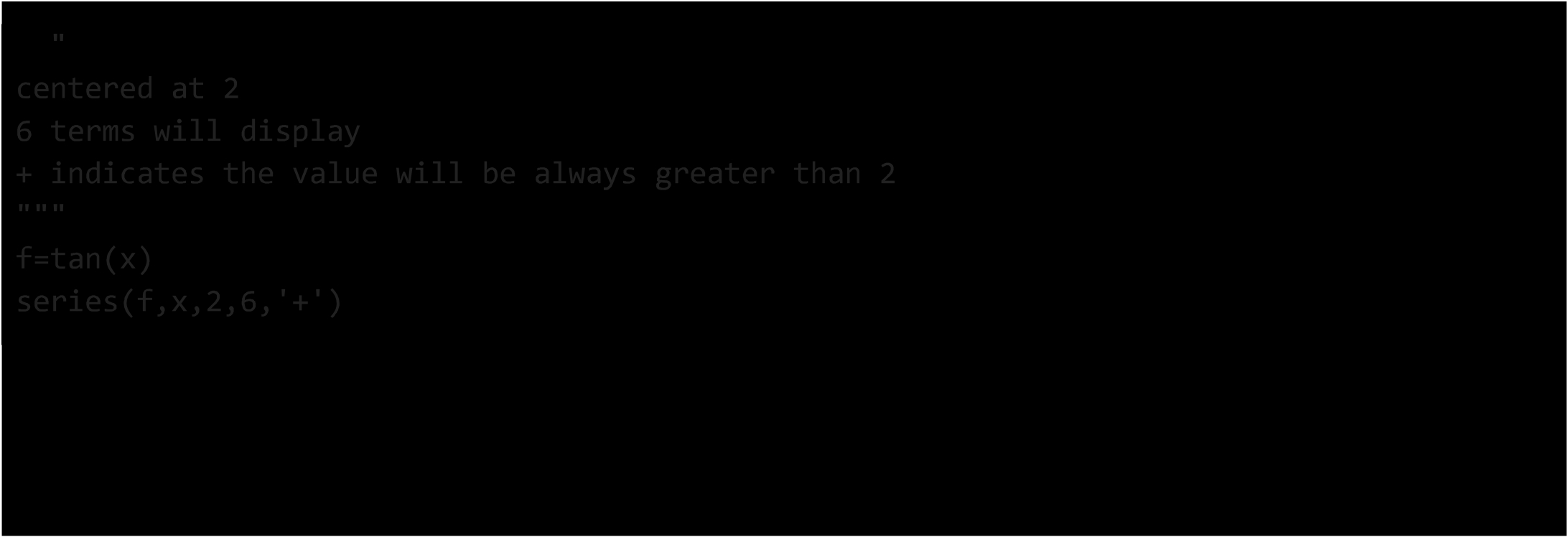
#centered at 0 series(cos(x),x)

x2 *x4 6*

**1-** - + - + 0 *(x )*

2 24

## YLORSERIES



tan (2) + (1 + tan2 (2)) *(x* - 2) + *(x* - 2)2 (tan3 (2) + tan (2)) + *(x* - 2)3 •

3

2

**(1** +4tan

(2) + tan 4( 2))

+( *x* - 2)4 (

tan 5(2) +5tan (2) + 2tan(2)) +(

*x* - 2)5 •

3 3 3 3

2 17 2 2 4 6 6

( 1 5 + t ( ) + 2 tan (2) + tan (2)) + 0 ( *(x* - 2) ; *x* ---+ 2)

centered at 2

6 indicates the degree of the polynomial and number of terms it is gonna display

- indicates the value will be always lesser than 2

f=tan(x) series(f,x,2,6,'-')

tan (2) + (2 - *x)* (-tan2 (2) - **1)** + (2 - x)2 (tan3 (2) + tan (2)) +

(2 - *x*)3[

- tan 4(2) -

4tan (2) - **1**- **)** +(2 - *x*)4( tan (2) +5tan (2) + 2tan(2)) +

2 3

5

3 3 3 3

2

(2-x)5

-tan6 (2)-2tan4 (2)- 17 tan (2)

-- 2)

+o(

(x-2)6 ;x---+2)

15 15

series(f,x,5,10,'-')

tan (5) + (5 - *x)* (-tan2 (5) - **1)** + (5 - x)2 (tan3 (5) + tan (5)) +

(5 - *x*)3[

- tan (5) - 4tan (5) - **1**- **)** +(5 - *x*)4( tan (5) +5tan (5) + 2tan(5)) +

2 3

4

5

3 3 3 3

(5 - *x)*5

- tan 6 (5) - 2 tan 4 (5) -

17 tan2 (5)

- -2) +

15 15

5 3

( 5 - *x* )

+ + + +

6 [ tan 7 ( 5) 7 tan (5) 77 tan (5) 17 tan (5))

3 45 45

(5 \_ *x)*7 \_ tans(5) \_ 8 tan6(5) \_ 12 tan4 (5) \_ 248 tan (5) \_ \_!I\_)+

3 5 315 315

8 ( 9() 7() 16tan5(5) 88tan3(5) 62tan(5))

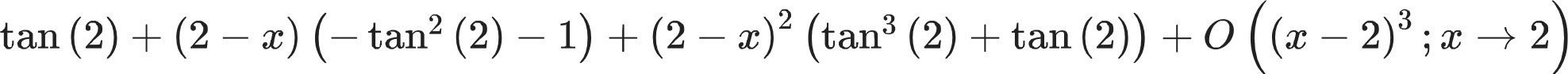
( 5 - *x* ) tan 5 + 3 tan 5 + 5 + 63 + 315 +

(5 \_*x* )9 (- tan10(5) \_ 10 tan8(5) \_ 37 tan6 (5) \_ 424 tan4 (5) \_ 1382 tan2(5) \_ )

3 9 189 2835 2835

0 *((x* - 5)10 *;x---+* 5)

series(f,x,2,3,'-')



#series expansion should have integral terms to be displayed #series(f,x,2,oo,'+') is an error bcz infinity is not an integer

al=O(x+x\*\*2) a2=O(x+x\*\*2,(x,0))

a3=O(x+x\*\*2,(x,0)) print(al,a2,a3)

O(x) O(x) O(x)

WAP to find the Taylor polynomial expansion of exponential function of x

f(x)=f(a)+ f'(a)(x-a) + f"(x)(x-a)"2/2! + + f"n(a)(x-a)\*\*n/n!

from sympy import\* x=Symbol('x')



func=exp(x) #input the function here

n=int(input("Enter the number of times differentiating")) #no.of times differenti a=float(input("Enter the center of series")) #center of series result=func.subs(x,a) #substituting x with a to find derivative of f(a)

#generation of series for i in range(l,n):

result+= diff(func,x,i).subs(x,a)\*(x-a)\*\*i/factorial(i) pretty\_print(result)

e the number of times differentiating 10 Enter the center of series 2

9

7.38905609893065·x + 0.0104254759773272·(0.5·x - 1) + 0.0469146418979724·(0.5

8 7 6

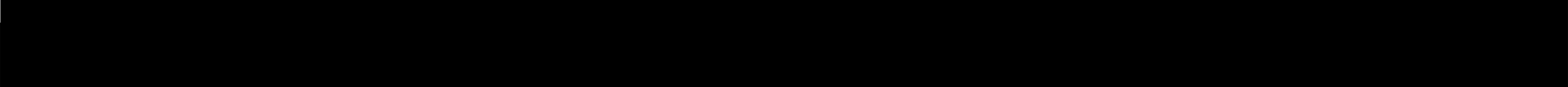
·X - 1) + 0.18765856759189·(0.5·X - 1) + 0.656804986571613·(0.5·X - 1) + 1.

5 4

97041495971484·(0.5·X - 1) + 4.9260373992871·(0.5·x - 1) + 9.8520747985742·(

3 2

0.5·X - 1) + 14.7781121978613·(0.5·X - 1) - 7.38905609893065



In [1]: **from** sympy **import\***

x = Symbol(**'x')**

d = diff(x\*\*2, x,1)

d

Out[1]: $\displaystyle 2 x$

In [6]: d = diff(sin(x),x,1)

d

Out[6]: $\displaystyle \cos{\left(x \right)}$

In [4]: d = diff(exp(x),x,1) d

Out[4]: $\displaystyle e/\{x}$

In [9]: y = Symbol('y')

d = diff(2\*x\*\*2 + 2\*y\*\*2, x,1) d

Out[9]: $\displaystyle 4 x$

In [10]: d = diff(sqrt(x\*\*3)+4\*x\*\*2, x, 1)

d

Out[10]: $\displaystyle 8 x + \frac{3 \sqrt{x/\{3}}}{2 x}$

In [11]: d = diff((x\*\*2)/ 8 - log(x), x,1)

d

Out[ll]: $\displaystyle \frac{x}{4} - \frac{l}{x}$

In **[2]: import** numpy **as** np

**from** scipy.integrate **import** odeint

**import** matplotlib.pyplot as plt

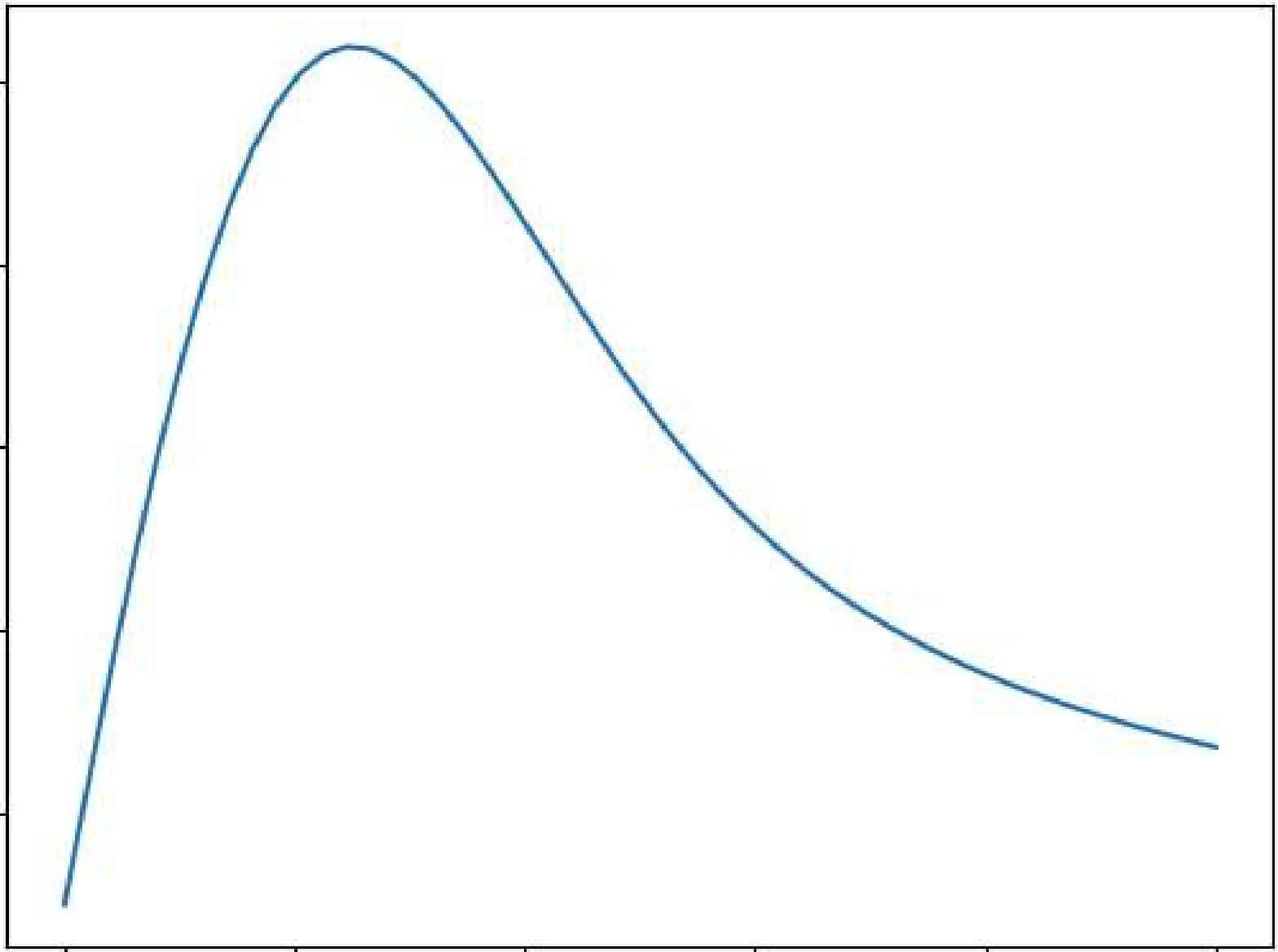
**y=l**

t = np.linspace(0,5)

**def** returns\_dydt(y, t): dydt = -y \* t + 13 **return** dydt

**y** = odeint(returns\_dydt, y, t) plt.plot(t,y)

pH.show()

10

8

6

4

2

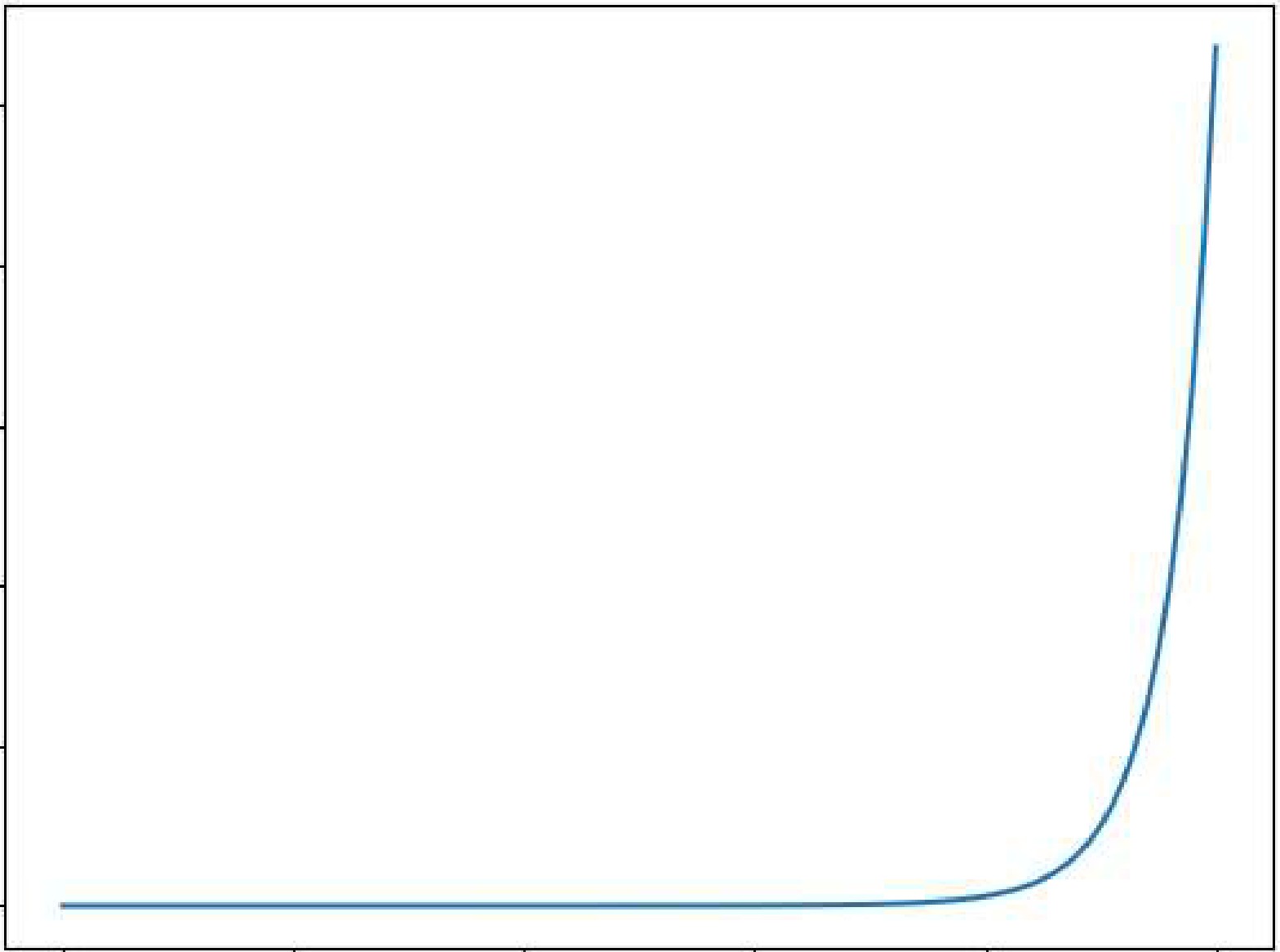
'1 2 3 4 5,

In [3]: def dy\_dx(y,x):

return x\*y

y = 1.0

x = np.linspace(0,5,100) y = odeint(dy\_dx, y,x) plt.plot(x,y)

pH.show()

2.50000

200000

150000

100000

50000

Q

Q 1 2 3 4 5



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

**from** scipy.integrate **import** odeint

**import** matplotlib.pyplot **as** plt

**y0=1**

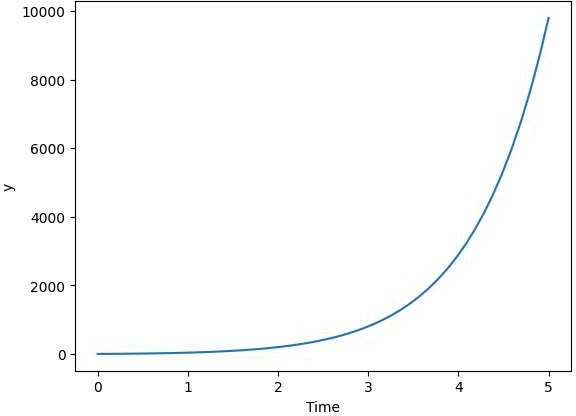
*#values of time*

t=np.linspace(0,5)

**def** returns\_dydt(y,t): dydt=13\*np.exp(t)+y **return** dydt

y=odeint(returns\_dydt,y0,t)

plt.plot(t,y) plt.xlabel("Time") plt.ylabel("y") plt.show()



**y0=1**

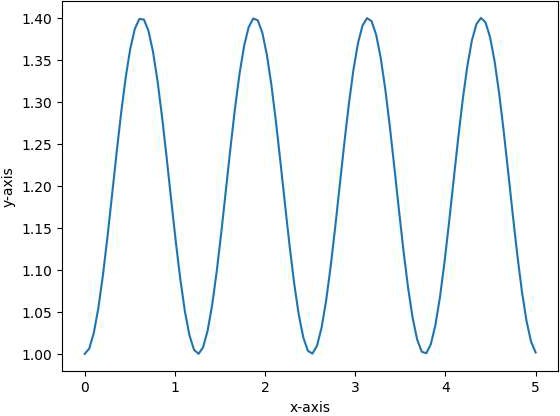
x=np.linspace(0,5,100)

**def** dy\_dx(y,x):

**return** np.sin(S\*x)

y=odeint(dy\_dx,y0,x) *#odeint -inbuilt function #plot results*

plt.plot(x,y) plt.xlabel("x-axis") plt.ylabel("y-axis") pH.show()





**from** scipy.integrate **import** odeint

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

**import** matplotlib.pyplot as plt

beta= .2

gamma= .1

N = 1000

I0 = 1

R0 = 0

S0 = N - I0 - R0

t = np.linspace(0,100,100)

**def** df(y,t,n, beta,gamma):

S,I,R = y

dsdt = -beta\*S\*I/N

didt = beta\*S\*I/N - gamma\*! dRdt = gamma\*!

**return** dsdt,didt,dRdt

y0 = S0,I0,R0

ret = odeint(df,y0,t,args=(N,beta,gamma)) S = ret[:,0]

I = ret[:,1]

R = ret[:,2]

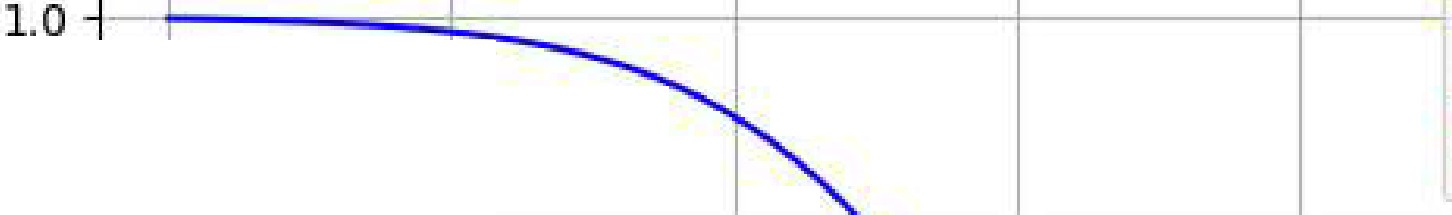
plt.plot(t,S/N,color='blue',label='S') plt.plot(t,I/N,color='red',label='I') plt.plot(t,R/N,color='green',label='R')

plt.legend()

pH.grid()

plt.show()

i



**0.8**,

+

+

+

1

**s** -

I

**R**

**0 20 40 60 100**

In [ ] :