UNIT 2 - ALGEBRA

Use of SymPy and NumPy package

---

sympy

It is a Python library for symbolic mathematics

```
x=1
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+1)
a=a+1
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
```

$$2xy$$

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

```
2*x*y
```

```
p=x*(x+x)
p
```

$$2x^2$$

```
p=(x+2)*(x+3)
p
```

$$(x+2)(x+3)$$

```
from sympy import factor
```

factor

It finds the factor of the given expression

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

$$\left(x - y\right) \left(x + y\right)$$

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)
```

$$x^{2} - y^{2}$$

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

$$\left(x + y\right)^{3}$$

```
#viceversa case
f=expand(factors)
f
```

$$x^{3} + 3 x^{2} y + 3 x y^{2} + y^{3}$$

```
expand(f)
```

$$x^{3} + 3 x^{2} y + 3 x y^{2} + y^{3}$$

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

$$x y + x + y$$

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
```

$$y^2 + 2xy + x^2$$

```
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)
```

$$\left(1,\ \left[\left(-y + x,\ 1\right),\ \left(y + x,\ 1\right)\right]\right)$$

```
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
```

$$-3 + x\left(1 + y\right) + x^2 \cdot \left(2 - z\right) + x^3$$

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))
```

$$\frac{1 + x}{x}$$

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 b is: "+str(b))
```

```
<class 'sympy.core.numbers.Rational'>
The value of a is: 5/8
The value of b is: 3
```

```
p=pi**3
print("The value of p is: "+str(p))
```

```
The value of p is: 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**1
r=exp(1).evalf()
print("The value of r is: "+str(r))
```

```
The value of r is: 2.71828182845905
```

```
s=(pi+exp(1)).evalf()
print("The value of s is: "+str(s))
```

```
The value of s is: 5.85987448204884
```

oo standes for infinity

```
rslt=oo+1000
print("The value of rslt is: "+str(rslt))
```

```
The value of rslt is: oo
```

```
if oo>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

```
ans1=diff(sin(x)*exp(x),x)
print("Derivative of sin(x)*e^x : ",ans1)
```

```
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(x^2)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: x^27=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(1)) #gives sin inverse
print(acos(0)) #gives cos inverse
print(atan(1)) #gives tan inverse
```

```
pi/2
pi/2
pi/4
```

```
x=1.0
y=1.0
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))
```

```
(1+1j)
The arc sine is :  0.666239432492515 + 1.06127506190504*I
The arc cosine is :  0.904556894302381 - 1.06127506190504*I
The arc tangent is :  1.01722196789785 + 0.402359478108525*I
The hyperbolic sine is :  0.634963914784736 + 1.29845758141598*I
The hyperbolic cosine is :  0.833730025131149 + 0.988897705762865*I
The hyperbolic tangent is :  1.08392332733869 + 0.271752585319512*I
The inverse hyperbolic sine is :  1.06127506190504 + 0.666239432492515*I
The inverse hyperbolic cosine is :  1.06127506190504 + 0.904556894302381*I
The inverse hyperbolic tangent is :  0.402359478108525 + 1.01722196789785*I
```

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(1))
```

```
1.0000066849139877
0.6931471805599453
0.0
```

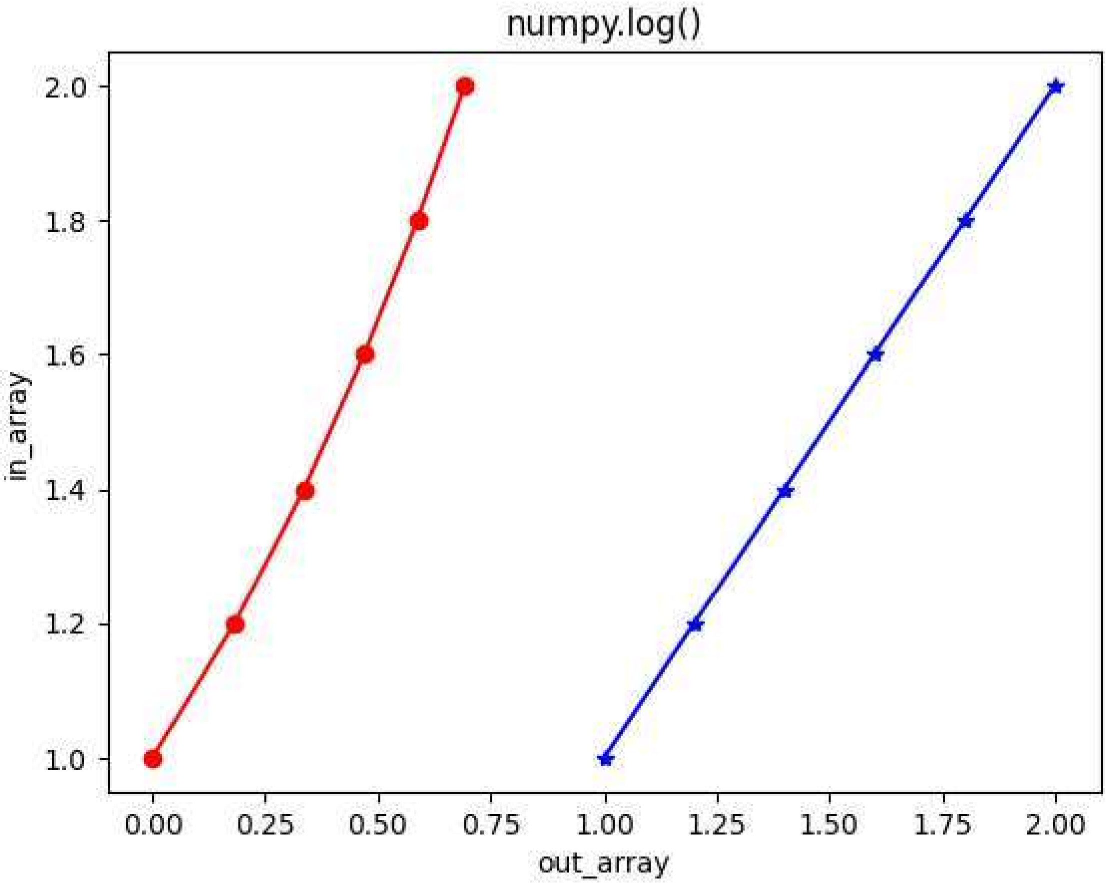
```
import numpy as np
import matplotlib.pyplot as plt
```

```
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>0 : two diff real roots

if D<0 :complex roots

if D=0 :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)
```

```
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

```
expr1 =x**2-4
ans1=limit(expr1,x,2)
print(ans1)
```

```
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)
```

```
-oo
```

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

```
15/2
```

DERIVATIVES

```
y1=diff(x**2+2*x+1,x)
print(y1)
```

```
2*x + 2
```

```
y2=diff(4*x**3-3*x**2+2*x-1,x)
print(y2)
```

```
12*x**2 - 6*x + 2
```

```
y3=diff(1/4*x**4+1/3*x**3+1/2*x**2,x)
print(y3)
```

```
1.0*x**3 + 1.0*x**2 + 1.0*x
```

```
y4=diff(x+x**(1/2)+x**(1/3)+x**(1/5),x)
print(y4)
```

```
0.2/x**0.8 + 0.333333333333333/x**0.666666666666667 + 0.5/x**0.5 + 1
```

```
y5=diff(x**(8/3)-x**(7/4)+x**(6/5),x)
print(y5)
```

```
1.2*x**0.2 - 1.75*x**0.75 + 2.66666666666667*x**1.66666666666667
```

```
y6=diff(1/((2*x**4)**(1/3))-1/((2*x**3)**(1/4)),x)
print(y6)
```

```
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(5))
```

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

```
-2/(x**2 + 1) + 3/sqrt(1 - x**2)
```

Calculus-limits

```
from sympy import *
```

```
x=Symbol('x')
a=limit(sin(x)/x,x,0)
b=limit(1/x,x,0) #default dir='+'
c=limit(1/x,x,0,dir='-') #dir='-' ----> LHL
d=limit(1/x,x,0,dir='+-')
e=limit(1/x,x,oo)
print(a,b,c,d,e)
```

```
1 oo -oo zoo 0
```

```
type(zoo)
```

```
sympy.core.numbers.ComplexInfinity
```

Series Expansion

POWER SERIES

```
#Macluarian's series
#centered at 0
series(cos(x),x)
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} + O\left(x^6\right)$$

TAYLOR SERIES

```
"""
centered at 2
6 terms will display
+ indicates the value will be always greater than 2
"""
f=tan(x)
series(f,x,2,6,'+')
```

$$\tan(2) + \left(1 + \tan^2(2)\right)(x-2) + (x-2)^2\left(\tan^3(2) + \tan(2)\right) + (x-2)^3 \cdot \left(\frac{1}{3} + \frac{4\tan^2(2)}{3} + \tan^4(2)\right) + (x-2)^4\left(\tan^5(2) + \frac{5\tan^3(2)}{3} + \frac{2\tan(2)}{3}\right) + (x-2)^5 \cdot \left(\frac{2}{15} + \frac{17\tan^2(2)}{15} + 2\tan^4(2) + \tan^6(2)\right) + O\left((x-2)^6; x \to 2\right)$$

```
"""
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,'-')
```

$$\tan(2) + (2-x)\left(-\tan^2(2) - 1\right) + (2-x)^2\left(\tan^3(2) + \tan(2)\right) + (2-x)^3\left(-\tan^4(2) - \frac{4\tan^2(2)}{3} - \frac{1}{3}\right) + (2-x)^4\left(\tan^5(2) + \frac{5\tan^3(2)}{3} + \frac{2\tan(2)}{3}\right) + (2-x)^5\left(-\tan^6(2) - 2\tan^4(2) - \frac{17\tan^2(2)}{15} - \frac{2}{15}\right) + O\left((x-2)^6; x \to 2\right)$$

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

$$\tan(5) + (5-x)\left(-\tan^2(5) - 1\right) + (5-x)^2\left(\tan^3(5) + \tan(5)\right) + (5-x)^3\left(-\tan^4(5) - \frac{4\tan^2(5)}{3} - \frac{1}{3}\right) + (5-x)^4\left(\tan^5(5) + \frac{5\tan^3(5)}{3} + \frac{2\tan(5)}{3}\right) + (5-x)^5\left(-\tan^6(5) - 2\tan^4(5) - \frac{17\tan^2(5)}{15} - \frac{2}{15}\right) + (5-x)^6\left(\tan^7(5) + \frac{7\tan^5(5)}{3} + \frac{77\tan^3(5)}{45} + \frac{17\tan(5)}{45}\right) + (5-x)^7\left(-\tan^8(5) - \frac{8\tan^6(5)}{3} - \frac{12\tan^4(5)}{5} - \frac{248\tan^2(5)}{315} - \frac{17}{315}\right) + (5-x)^8\left(\tan^9(5) + 3\tan^7(5) + \frac{16\tan^5(5)}{5} + \frac{88\tan^3(5)}{63} + \frac{62\tan(5)}{315}\right) + (5-x)^9\left(-\tan^{10}(5) - \frac{10\tan^8(5)}{3} - \frac{37\tan^6(5)}{9} - \frac{424\tan^4(5)}{189} - \frac{1382\tan^2(5)}{2835} - \frac{62}{2835}\right) O\left((x-5)^{10}; x \to 5\right)$$

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

$$\tan(2) + (2-x)\left(-\tan^2(2) - 1\right) + (2-x)^2\left(\tan^3(2) + \tan(2)\right) + O\left((x-2)^3; x \to 2\right)$$

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

```
a1=O(x+x**2)
a2=O(x+x**2,(x,0))
a3=O(x+x**2,(x,0))
print(a1,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 differentia
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(1,n):
  result+= diff(func,x,i).subs(x,a)*(x-a)**i/factorial(i)

pretty_print(result)
```

```
Enter 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
```