1st sem python (AG mam)

calculation of mean, variance, std dev

In [1]:
```
import math
import numpy as np

x = [2,3,0,1,-2,5]
y = [i**2 for i in x]
n = len(x)
mean = np.sum(x)/n
var = np.sum(y)/n-mean**2
stdev=math.sqrt(var)
print(f'mean, var, stdev = {mean}, {var}, {stdev}')
```

```
mean, var, stdev = 1.5, 4.916666666666667, 2.217355782608345
```

gcd and lcd of 2 numbers

```
m1, n1 = eval(input('enter 2 numbers: m, n = '))
mnprod = m1*n1

def gcd(m, n):
    r = m%n
    while r>0:
        m, n = n, r
        r = m%n
    return n

print(f'GCD = {gcd(m1,n1)} and LCD = {mnprod/gcd(m1,n1)}')
```

Pythagoras number generation

```
def gcd(m, n):
    r = m%n
    while r>0:
        m, n = n, r
        r = m%n
    return n

n1 = eval(input('enter a number: '))
for m in range(1,n1):
    if gcd(m, n1)==1:
        x = n**2 - m**2
        y = 2*n*m
        z = n**2 + m**2
        print(f'x = {x}, y = {y}, z = {z}')
```

In [ ]:
```

```

sorting ascending order

In [2]:
```
L = [10,-2,0,1,100,15,9-12]

def sort_ascending(L):
    n = len(L)
    for i in range(n):
        key = L[i]
        j = i-1
        while j>=0 and L[j]>key:
            L[j+1] = L[j]
            j = j-1
        L[j+1] = key
    return(L)

print(f'''Original form: {L}
sort by defined function: {sort_ascending(L)}
sort by available function (sorted): {sorted(L)}''')

L.sort()
print(f'sort: {L}')
L.sort(reverse=True)
print(f'sort(reverse): {L}')
```

```
Original form: [10, -2, 0, 1, 100, 15, -3]
sort by defined function: [-3, -2, 0, 1, 10, 15, 100]
sort by available function (sorted): [-3, -2, 0, 1, 10, 15, 100]
sort: [-3, -2, 0, 1, 10, 15, 100]
sort(reverse): [100, 15, 10, 1, 0, -2, -3]
```

Binary and Decimal conversions

decimal to binary conversion

```
from math import modf

n = eval(input('enter the decimal number: '))
ndec, nint = modf(n)
nint = int(nint)
print(f'integer part = {nint}, decimal part = {ndec}')

def bin_int(n):
    b = str()
    while n>0:
        res = n%2
        b += str(res)
        n = n//2
    return b[::-1]

def bin_frac(n):
    b = str()
    i = 1
    while n>0:
        n = n*2
        frac, intg = modf(n)
        b += str(int(intg))
        n = frac
```

```
        i += 1
        if i>6:  # input
            break
    return b

print(f'the binary number is {bin_int(nint)}.{bin_frac(ndec)}')
```

binary to decimal conversion

```
m = eval(input('give the binary number: '))
s, i = 0, 0
while m>0:
    res = m%10
    m = m//10
    s += (2**i)*res
    i += 1
print(f'Decimal number for the given binary number is {s}.')
```

Numerical methods for finding roots

roots of a quadratic equation

```
import math as m
a, b, c = eval(input('a, b, c = '))
D = b**2 -4*a*c
D1 = m.sqrt(abs(D))
a2 = 2*a

if D>0:
    print(f'distinct real roots = {(-b+D1)/a2}, {(-b-D1)/a2}')
elif D==0:
    print(f'equal real roots = {-b/a2}, {-b/a2}')
elif D<0:
    print(f'complex roots: {complex(-b,D1)/a2}, {complex(-b,-D1)/a2}')
```

In [3]:
```
a, b, c = 4, 8, 7   # input
D1 = (b**2-4*a*c)**0.5
print(f'D = {D1**2} \nroots = {(-b+D1)/(2*a)}, {(-b-D1)/(2*a)}')
```

```
D = (-47.99999999999999+5.878304635907294e-15j)
roots = (-1+0.8660254037844386j), (-1-0.8660254037844386j)
```

bisection method

```
def f(x):
    return x**3 -2*x -5

a, b, tol = eval(input('lower input, upper limit, tol = '))

while f(a)*f(b)>0:
    print('no root exists in this interval')
    break
while abs(b-a)>=tol:
```

```
    xm = (a+b)/2
    if f(xm)==0:
        print(f'root = {xm}')
        break
    if f(a)*f(xm)<0:
        b = xm
    else:
        a = xm
print(f'root = {(a+b)/2}')
```

Newton - Raphson method

```
def f(x):
    return x**3 -2*x -5
def h(x):
    return 3*x**2 -2

x1, tol = eval(input('initial point, tol = '))
while abs(f(x1))>=tol:
    x1 = x1 - f(x1)/h(x1)
    print(x1)
print(f'root = {x1}')
```

secant method

```
def f(x):
    return x**3 -2*x -5
x0, x1, tol = eval(input('x0, x1, tol = '))
while abs(f(x1))>=tol:
    x2 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0))
    x0, x1 = x1, x2
    print('x1={}, f(x1)={}'.format(x1, f(x1)))
print(f'root = {x1}')
```

Interpolation

direct forward difference method

In [4]:
```python
x = list(range(10))
y = [0, 0.368, 0.541, 0.448, 0.39, 0.34, 0.38, 0.31, 0.2, 0.14]

xx = 4.7  # input value
n = len(x)
h = x[1]-x[0]
p = (xx-x[0])/h
cf = p
k = 1
yy = y[0]
d = []
for i in range(n, 1, -1):
    for j in range(i-1):
        diff = y[j+1]-y[j]
        d.append(diff)
    yy += cf*d[0]
    cf *= (p-k)/(k+1)
    k += 1
    y = d
    d = []
print(f'for x = {xx}, y = {yy}')
```

```
for x = 4.7, y = 0.3468285114337938
```

Matrices

matrix addition

In [5]:
```python
A = [[1,2,3],[4,5,6]]
B = [[15,6,4],[3,4,2]]
C = [[0,0,0],[0,0,0]] # null matrix with the dimension of the result

row = len(A)
col = len(A[0])

for i in range(row):
    for j in range(col):
        C[i][j] = A[i][j] + B[i][j]

print(f'using defined function: {C}')
for rows in C:
    print(rows)

import numpy as np
An = np.array(A)
Bn = (B)
Cn = An + Bn
print(f'using numpy array: \n{Cn}')
```

```
using defined function: [[16, 8, 7], [7, 9, 8]]
[16, 8, 7]
[7, 9, 8]
using numpy array:
[[16  8  7]
 [ 7  9  8]]
```

matrix product

In [6]:
```
A = [[1,2,3],[4,5,6]]
B = [[1,2],[3,4],[5,6]]
C = [[0,0],[0,0]] # null matrix with the dimension of the result

for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            C[i][j] = A[i][k]*B[k][j]

print(f'multiplication by defined function: {C}')
for rows in C:
    print(rows)
```

```
multiplication by defined function: [[15, 18], [30, 36]]
[15, 18]
[30, 36]
```

In [ ]:
```

```

2nd sem python

Integration

integration by trapizoidal rule

In [7]:
```
import numpy as np

def f(x):
    return x**2

a,b,n=0,1,400
h=(b-a)/n
sum=f(a)+f(b)
for i in range(1,n-1):
    sum=sum+2*f(a+i*h)
int=sum*0.5*h
print('value=',int)
```

```
value= 0.3308468593750001
```

integration by Simpson's rule

In [8]:
```
import numpy as np

def f(x):
    return np.sin(x)

a=np.pi/3
b=np.pi
n=400
h=(b-a)/n
sum=f(a)+f(b)
sum_odd=0
sum_even=0
for i in range(1,n-1,2):
    sum_odd+=4*f(a+i*h)
    sum_even+=2*f(a+(i+1)*h)
int=h/3*(sum+sum_odd+sum_even)
print('value=',int)
```

```
value= 1.499963446082914
```