SciPy Tutorial

SciPy is a Python library which is used to solve scientific and mathematical problems

single intergration

In [ ]:
```
from scipy.integrate import quad  # allows us to integrate from limit a to
```

In [ ]:
```
# defining function for integration
def integratefunc(x):
    return x
```

In [ ]:
```
quad(integratefunc,0,1)
```

here 5.55.. is the numerical error in computation

example 2

In [ ]:
```
def integratefunc2(x,a,b):
    return x*a+b

a=3
b=2
```

In [ ]:
```
quad(integratefunc2,0,1,args=(a,b))
```

example 3

In [ ]:
```
from scipy import integrate
func = lambda x: x**2 + x + 1
ans = integrate.quad(func, 1, 4)
print(ans)
print(ans[0])
```

double integration

In [ ]:
```
import scipy.integrate as integrate
```

In [ ]:
```
def f(x,y):
    return x+y
integrate.dblquad(f,0,1,lambda x:0, lambda x:2)
# range of x is 0,1 and y is 0,2
```

optimization

- Optimization is a mathematical procedure to select the best decision variables from a set of alternatives to get the optimum objective value.
- used to improve performance of system mathematically by fine tuning process paameters

In [ ]:
```
import numpy as np
from scipy import optimize
```

finding minimum value

In [ ]:
```
def f(x):
    return x**2+5*np.sin(x)
```

In [ ]:
```
minimal_value = optimize.minimize(f,x0=0,method='bfgs',options={'disp':True
```

In [ ]:
```
minimalValue=optimize.minimize(f,x0=0,method='bfgs')
minimalValue
```

finding roots

In [ ]:
```
from scipy.optimize import root
```

In [ ]:
```
def rootfunc(x):
    return x + 3.5 * np.cos(x)
```

In [ ]:
```
root=root(rootfunc,0.3)  #0.3 is intial guess for root
root
```

here x:[-1.21..] is the root

matrix

In [ ]:
```
from scipy import linalg
```

In [ ]:
```
matrix=np.array([[1,2],[3,8]])
matrix
```

inversing matrix

In [ ]:
```
linalg.inv(matrix)
```

determinant of matrix

In [ ]:
```
linalg.det(matrix)
```

solving linear functions

linear equations

- $2x + 3y + z = 21$
- $-x + 5y + 4z = 9$
- $3x + 2y + 9z = 6$

In [ ]:
```
numarr=np.array([[2,3,1],[-1,5,4],[3,2,9]])
num_val=np.array([21,9,6])
```

In [ ]:
```
linalg.solve(numarr,num_val)
```

another method

In [ ]:
```
from sympy import symbols, Eq, solve
x,y,z=symbols('x,y,z')
eq1=Eq((2*x+3*y+z),21)
eq2=Eq((-1*x+5*y+4*z),9)
eq3=Eq((3*x+2*y+9*z),6)
solve((eq1,eq2,eq3),(x,y,z))
```

single value decomposition

- Singular Value Decomposition is a factorization of a matrix into three matrices

In [ ]:
```
matrix=np.array([[1,2],[3,8]])
matrix.shape
```

In [ ]:
```
linalg.svd(matrix)
```

U: Contains the left singular vectors

Σ: Diagonal matrix with singular values

VH: Contains the right singular vectors

Eigenvalues and Eigenvectors

In [ ]:
```
# test_data matrix
test_data = np.array([[5,8],[7,9]])
eigenValues, eigenVector = linalg.eig(test_data)
```

In [ ]:
```
# eignen values
eigenValues
```

In [ ]:
```
#eigenvectors
eigenVector
```

statistics

In [ ]:
```
from scipy.stats import norm #normal distribution
```

generating 10 random numbers sampled from a normal distribution with mean 0 and standard deviation 1

In [ ]:
```
norm.rvs(loc=0,scale=1,size=10)  #loc - mean, scale- standard deviation
```

cumulative distribution function

probability that a random variable from this normal distribution is less than or equal to 5

In [ ]:
```
norm.cdf(5,loc=1,scale=2)
```

probability density function

value of the probability density function of the normal distribution at point 9.

In [ ]:
```
norm.pdf(9,loc=0,scale=1)
```

skewness and kurtosis

- skewness- measure of symetry
- kurtosis- A measure of whether the data is heavy or lightly tailed

In [ ]:
```
from scipy.stats import skew, kurtosis
v = np.random.normal(size=100)
print(skew(v))
print(kurtosis(v))
```

statistical description

In [ ]:
```
from scipy.stats import describe
v2 = np.random.normal(size=100)
res = describe(v2)
print(res)
```

examples

Test has 30 questions worth 150 marks

Two types of questions will be there:

- True,False worth 4 marks each
- Multiple choice worth 9 points each

Let x be the no of true/false questions

Let y be the MCQ questions

x + y = 30

4x + 9y = 150

In [ ]:
```
num=np.array([[1,1],[4,9]])
sol=np.array([30,150])
linalg.solve(num,sol)
```

Value of x = 24, Value of y = 6

Means, True/False questions will be 24 & MCQ questions will be 6 in number

In [ ]:
```

```