NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY

VENKATAGIRI KOTE, DEVANAHALLI, BENGALURU-562164

Department of Mathematics

Python Manual-I SEM

Introduction

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently. There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open source, this means that source code is also available to the public. So you can download it, use it as well as share it.

Set up working environment

There are different ways to install Python and related packages, here we recommend to use Anaconda or Miniconda to install and manage your packages. Depending on the *operating systems* (OS) you are using, i.e Windows, Mac OS X, or Linux, you need to download a specific installer for your machine. Both Anaconda and Miniconda are aiming to provide easy ways to manage Python work environment in scientific computing and data sciences.

Here we will use Mac OS X as an example to show you the install processes. For windows users, please skip the rest of this section and read Appendix A for all the processes. The main differences between Anaconda and Miniconda are:

- **Anaconda** is a complete distribution framework that includes the Python interpreter, package manager as well as the commonly used packages in scientific computing.
- **Miniconda** is a light version of Anaconda that does not include the common packages, therefore, you need to install all the different packages by yourself. But it does have the Python interpreter and package manager.

Installing Anaconda on Windows

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 .

Types of operators in Python

Python language supports a wide range of operators. They are

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Identity Operators
7. Membership Operators

Operators:

Operator	Description
( )	Parentheses
**	Exponent
* / %	Multiplication/division/modulus
+ –	Addition/subtraction
<< >>	Bitwise shift left, Bitwise shift right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to
== !=	Relational is equal to/is not equal to
is, is not in, not in	Identity Membership operators
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR
Not	Logical NOT
And	Logical AND
Or	Logical OR
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment

Data types in Python.

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary

Conditional structure

- Based on certain conditions, the flow of execution of the program is determinedusing proper syntax.
- Often called decision-making statements in Python.

How to use if conditions?

if statement — for implementing one-way branching

```
#Syntax:
if condition:
statements
```

if-else statements —for implementing two-way branching

```
Synatx:
if condition:
    statements 1
 else:
    statements 2
    # If condition is True-  statements1 will be executed#
    otherwise - statements 2 will be executed
```

nested if statements —for implementing multiple branching

```
# Syntax:
 if condition 1:
     statements 1
 elif condition 2:
      statements 2
 elif condition 3:
      statements 3
else:
     statements 4
  # If condition 1 is True - Statements 1 will be executed.
  # else if condition 2 is True - Statements 2 will be executed and so on.
  # If any of the conditions is not True then statements in else block is
  executed.
```

Control flow (Loops)

Loop types:

1. While loop:

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

- Is used to execute a block of statements repeatedly until a given condition is satisfied.
- When the condition becomes false, the line immediately after the loop in the pro-gram is executed
    - Syntax:

```
while
    expression:
    statement(s)
```

Break statement

- It terminates the current loop and resumes execution at the next statement.
- The most common use for break is when some external condition is triggered re-quiring a hasty exit from a loop.
- The break statement can be used in both while and for loops.
- If you are using nested loops, the break statement stops the execution of the inner-most loop and start executing the next line of code after the block.

```
# Use of break ststement
i=1
while   i< 6 :
  print(i) if
  i==3:
    breaki+=1
    output
    1 2 3
```

Continue statement

- The continue statement rejects all the remaining statements in the current iterationof the loop and moves the control back to the top of the loop.
- The continue statement can be used in both while and for loops.

```
i=0
while i<6:
  i+=1
  if i==3:
    continue
  output
   1 2 4 5 6

  print(i)
```

For loop

- are used for sequential traversal
- it falls under the category of definite iteration
- also used to access elements from a container (for example list, string, tuple) usingbuilt-in function range()

Syntax:

```
for variable_name in
        sequence : statement_1
        statement_2
        ....
```

The range() function

Syntax:

range(a) : Generates a sequence of numbers from 0 to a, excluding a, incrementing by 1.

range(a,b): Generates a sequence of numbers from a to b excluding b, incrementing by 1.

range(a,b,c):Generates a sequence of numbers from a to b excluding b, incrementing byc.

```
#Print numbers from 101 to 130 with a step length 2 excluding 130.
for i in range(101,
    130,2): print(i)
output
101 103 105 107
    109 111 113
    115 117 119
    121 123 125
    127 129
```

Type conversion in Python:

Python defines type conversion functions to directly convert one data type to another which is useful in day-to-day and competitive programming. This article is aimed at providing information about certain conversion functions.

There are two types of Type Conversion in Python:

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion

In Implicit type conversion of data types in Python, the Python interpreter automatically converts one data type to another without any user involvement.

Explicit Type Conversion

In Explicit Type Conversion in Python, the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type. Various forms of explicit type conversion are explained below:

1. int(a, base): This function converts any data type to integer. 'Base' specifies the base in which string is if the data type is a string.
2. float(): This function is used to convert any data type to a floating-point number.
3. ord() : This function is used to convert a character to integer.
4. hex() : This function is to convert integer to hexadecimal string.
5. oct() : This function is to convert integer to octal string.
6. tuple() : This function is used to convert to a tuple.
7. set() : This function returns the type after converting to set.
8. list() : This function is used to convert any data type to a list type.
9. dict() : This function is used to convert a tuple of order (key,value) into a dictionary.
10. str() : Used to convert integer into a string.
11. complex(real,imag) : This function converts real numbers to complex(real,imag) number.
12. chr(number): This function converts number to its corresponding ASCII character.

LAB 1: 2D plots of Cartesian and polar curves.

Syntax for the commands used:

1. Importing the required module
   - import matplotlib.pyplot as plt
   - import numpy as np
   - from sympy import plot implicit,symbols,x,y=symbols('x y')
2. Xlabel('x -axis') # label the graph
3. Ylabel('y-axis') # label the graph
4. Title('plot the graph')

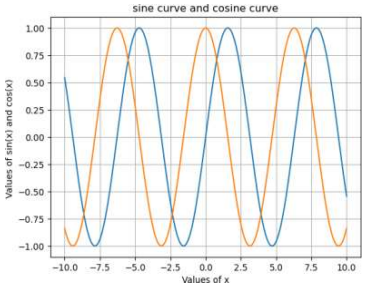
5. Show() # function to show the graph
6. Plot *y* versus *x* as lines and or markers using default line style, color and other customizations.
   - Plot(x,y,color='green',marker='o',linestyle='dashed',linewidth=2,markersize=12)
7. A scatter plot of *y* versus *x* with color.
   - Scatter(x axis data,y axis data,color="red")
8. Return num evenly spaced numbers over a specified interval [start, stop].The endpoint of the interval can optionally be excluded.
   - Numpy.linspace(start,stop,num=50,endpoint=True,retstep=False,dtype=None,axis=0)
9. Return evenly spaced values within a given interval. arange can be called with a varying number of positional arguments.
   - numpy.arrange([start,]stop,[step,]dtype=None, * , like=None)
10. Implicit Function
    - Plot_implicit(expr,x_var=None,y_var=None,adaptive=true,depth=0,points=300,line color='blue',show=True,**kwargs)

1. Plot *sine and cosine curves*.

Command:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arrange(-10,10,0.001)
y1 = np.sin(x)
y2 = np.cos(x)
Plt.plot(x, y1, x, y2)   # plotting sine and cosine function together with same values of x
plt.title("sine curve and cosine curve")
plt.xlabel("value of x")
plt.ylabel("values of sin(x) and cos(x)")
plt.grid()
plt.show()
```

Output:

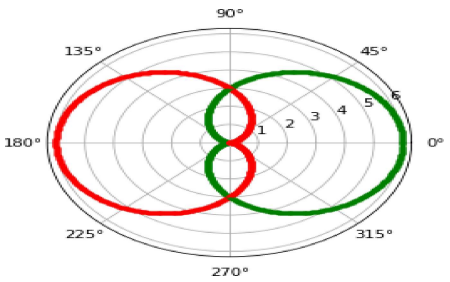


2. Plot cardioid curves $r = a + a\cos\theta$ *and* $r = a - a\cos\theta$.

Command:

```
import numpy as np
import matplotlib.pyplot as plt
import math
 plt.axes(projection='polar')
a=3
rad=np.arange(0,(2*np.pi),0.01)
for i in rad:
r = a + (a * np.cos(i))
Plt.polar( i,r,'g.')
r1 = a - (a * np.cos(i))
Plt.polar( i,r1,'r.')
 # display the polar plot
plt.show()
```

Output:



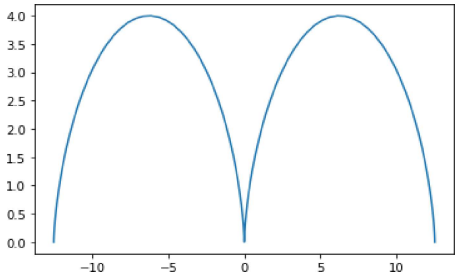
3. Plot Cycloid curve $x = a(\theta - sin\theta)$, $y = a(1 - cos\theta)$.

Command:

```
import numpy as np
import matplotlib.pyplot as plt
def cycloid(r):
x = []
y = []
# create the list of x coordinates and list of y coordinates
 for theta in np.linspace(-2*np.pi,2*np.pi,100):
```

```
# loop over a list of theta , which ranges from -2 pi to 2 pi
    x.append(r*(theta-np.sin(theta)))
        #add the corresponding expression of x to the x list
    y.append(r*(1-np.cos(theta)))
    plt.plot(x,y)      # plot using matplotlib . piplot
    plt.show()  # show the plot
    cycloid(2)  # call the function
```

Output:



LAB 2: Finding angle between two polar curves, curvature and radius of curvature.

Syntax for the commands used:

- diff (function, variable)
- Derivative (expression, reference variable)
- doit ()
- simplify(expression)
- display(expression)
- math expression. Subs(variable, substitute)

1 Find the angle between the curves $r = 4(1 + cost) and\ r = 5(1 - cost)$

Command:

```
from sympy import*
 r,t=symbols('r,t')
r1=4*(1+cos(t)); # Input first polar curve
r2=5*(1-cos(t)); # Input first polar curve
dr1=diff(r1,t)
dr2=diff(r2,t)
t1 =r1/dr1
t2 =r2/ dr2
q= solve(r1-r2,t)
```

```
w1=t1.subs({t:float(q[1])})
w2=t2.subs({t:float(q[1])})
y1=atan(w1) # to find the inverse tan of w1
y2=atan(w2) # to find the inverse tan of w2
w=abs(y1-y2)
print('angle between curves in radians is %0.3f'%( w ))
```

Output:

Angle between curves in radians is 1.571

2 Find the radius of curvature, $r = 4(1 + cost)$ at $t = \frac{\pi}{2}$.

Command:

```
from sympy import*
 t=symbol('t')     # define t as symbol
 r=symbol('r')
 r=4*(1+cos(t))
 r1=Derivative(r,t).doit()
 r2=Derivative(r1,t).doit()
 rho=(r**2+r1**2)**(1.5)/(r**2+2*r1**2-r*r2);
 rho1=rho.subs(t,pi/2)    # substitute t in rho
print('The radius of curvature is %3.4f units'%rho1)
```

Output:

```
The radius of curvature is 3.7712 units
```

3 Find the radius of curvature of $x = a\,cost, y = a\,sint$.

Command:

```
from sympy import*
from sympy.abc import rho, x,y,r,K,t,a,b,c,alpha
y=a*sin(t) # input the parametric equation
x=a*cos(t)
dydx=simplify(Derivative(y,t).doit())/simplify(Derivative(x,t).
doit())
rho=simplify((1+dydx**2)**1.5/(Derivative(dydx,t).doit
()/(Derivative(x,t).doit())))
 # substitute thederivative in radius of curvatureformula
print('Radius of curvature is')
display(ratsimp(rho))
t1=pi/2
r1=5
rho1=rho.subs(t,t1);
rho2=rho1.subs(a,r1);
```

```
print('\n\nRadius of curvature at r=5 and t= pi/2 is', simplify
(rho2)); curvature=1/rho2;
print('\n\n Curvature at (5,pi/2) is',float(curvature))
```

Output:

```
Radius of curvature is
```

$$-a\left(\frac{1}{sin^2(t)}\right)^{0.5}\sin(t)$$

```
\ nRadius of curvature at r=5 and t= pi/2 is -5
 Curvature at (5, pi/2) is  -0.2
```

LAB 3: Finding partial derivatives and Jacobian of functions of several variables.

Syntax for the commands used:

1. To create a matrix:
   - Matrix([[row1],[row2],[row3]----------[rown]])
2. To evaluate determinant of a matrix A
   - det(A)
3. To evaluate derivative of function w.r.t variable
   - diff(function, variable)
4. If function is of two or more than two independent variables then it differentiates the function partially w.r.t variable.

   If $u = u(x, y)$ then,
   - $\frac{\partial u}{\partial x} = diff(u, x)$
   - $\frac{\partial u}{\partial y} = diff(u, y)$
   - $\frac{\partial^2 u}{\partial x^2} = diff(u, x, x)$
   - $\frac{\partial^2 u}{\partial y^2} = diff(u, y, y)$
   - $\frac{\partial^2 u}{\partial x \partial y} = diff(u, x, y)$

1. Prove that if $u = e^x(xcosy - ysiny) then\ u_{xx} + u_{yy} = 0$.

Command:

```
from sympy import *
x,y=symbols('x,y')
u=exp(x)*(x*cos(y)-y*sin(y))
display(u)
dux=diff(u,x)
```

```
duy=diff(u,y)
uxx=diff(dux,x)   # or uxx= diff (u,x,x) second derivative of u w.r.t x
uyy=diff(duy,y)    # or uyy= diff (u,y,y) second derivative of u w.r.t y
w=uxx+uyy          # Add uxx and uyy
w1=simplify(w)      # Simply the w to get actual result
print('Ans:',float(w1))
```

Output:

$(x\cos(y)-y\sin(y))\ e^{x}$

```
Ans: 0.0
```

2. If $x = \rho\ cos\emptyset\ sin\theta\ , y = \ \rho\ cos\emptyset\ cos\theta, z = \ \rho \sin\emptyset\ then\ find\ \frac{\partial(x,y,z)}{\partial(\rho,\emptyset,\theta)}$.

Command:

```
from sympy import *
from sympy.abc import rho,phi,theta
X=rho*cos(phi)*sin(theta);
Y=rho*cos(phi)*cos(theta);
Z=rho*sin(phi);
dx=Derivative(X,rho).doit()
dy=Derivative(Y,rho).doit()
dz=Derivative(Z,rho).doit()
dx1=Derivative(X,phi).doit();
dy1=Derivative(Y,phi).doit();
dz1=Derivative(Z,phi).doit();
dx2=Derivative(X,theta).doit()
dy2=Derivative(Y,theta).doit();
dz2=Derivative(Z,theta).doit();
J=Matrix([[dx,dy,dz],[dx1,dy1,dz1],[dx2,dy2,dz2]]);
print('The Jacobian matrix is')
display(J)
print('\n\n J=\n')
display(simplify (Determinant(J).doit()))
```

Output:

```
The Jacobian matrix is
```

$$\begin{bmatrix} \sin(\theta)\cos(\phi) & \cos(\phi)\cos(\theta) & \sin(\phi) \\ -\rho\sin(\phi)\sin(\theta) & -\rho\sin(\phi)\cos(\theta) & \rho\cos(\phi) \\ \rho\cos(\phi)\cos(\theta) & -\rho\sin(\theta)\cos(\phi) & 0 \end{bmatrix}$$

J $=\rho^{2}\cos(\phi)$

LAB 4: Applications of Maxima and Minima of functions of two variables, Taylor series expansion and L'Hospital's Rule

Syntax for the commands used:

1. To solve mathematical expression or polynomial
   - sympy.solve(expression)
2. To evaluate mathematical expression
   - Sympy.evalf( )
3. To construct an instant function.
   - Sympy.lambdify(variable,expression,library)
4. To find the limit of a function
   - Limit(expression,variable,value)

1. Find the maxima and minima of $f(x,y) = x^2 + y^2 + 3x - 3y + 4$.

   Command:

```
import sympy
from sympy import Symbol,solve,Derivative,pprint
x=Symbol('x')
y=Symbol('y')
f=x**2+y**2+3*x-3*y+4
d1=Derivative(f,x).doit()
d2=Derivative(f,y).doit()
criticalpoints1=solve(d1)
criticalpoints2=solve(d2)
s1=Derivative(f,x,2).doit()
s2=Derivative(f,y,2).doit()
s3=Derivative(Derivative(f,y),x).doit()
print('function value is')
q1=s1.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
q2=s2.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
q3=s3.subs({y:criticalpoints1,x:criticalpoints2}).evalf()
delta=s1*s2-s3**2
print(delta,q1)
if(delta>0 and s1<0):
    print("f takes maximum")
elif(delta>0 and s1>0):
    print("f takes minimum")
if (delta<0):
    print("The point is a saddle point")
if (delta==0):
    print("further tests required")
```

Output:

```
function value is
4 2.00000000000000
f takes minimum
```

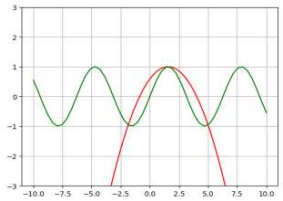
2. Expand $\sin x$ as Taylor series about $x = \frac{pi}{2}$ up to 3rd degree term. Also find $\sin(100^{\circ})$.

Command:

```
import numpy as np
from matplotlib import pyplot as plt
from sympy import *
x= Symbol ('x')
y=sin( 1*x )
format
x0= float (pi/2 )
dy= diff (y , x )
d2y = diff (y ,x , 2 )
d3y = diff (y ,x , 3 )
yat = lambdify (x , y )
dyat = lambdify (x , dy )
d2yat = lambdify (x , d2y )
d3yat = lambdify (x , d3y )
y=yat ( x0 )+(( x-x0 )/2 )* dyat ( x0 )+(( x-x0 ) ** 2/6 )* d2yat ( x0 )+(( x-x0 ) ** 3/24 )*d3yat ( x0 )
print ( simplify ( y ) )
yat = lambdify (x , y )
print ("%.3f" % yat (pi/2+10*(pi/180 ) ) )
def f(x):
return np.sin( 1*x )
x = np.linspace (-10 , 10 )
plt.plot(x,yat(x),color='red')
plt.plot(x,f(x),color ='green')
plt.ylim ([-3 , 3])
plt.grid ()
plt.show ()
```

Output:

```
-2.55134749822365e-18*x**3 - 0.166666666666667*x**2 + 0.523598775598299*x + 0.588766483287943
0.995
```



3. Find $\lim_{x \to 0} \frac{\sin x}{x}$

Command:

```
from sympy import Limit , Symbol ,exp ,sin
x= Symbol ('x')
L= Limit (( sin( x ) )/x ,x , 0 ) . doit ()
print ( L)
```

Output:

L = 1

LAB 5: Solution of First order differential equationand plotting the curves.

Syntax for the commands used:

1. To solve differential equation
   - dsolve( )
2. To find ordinary integration
   - odient( )

1 Solve: $\frac{dy}{dx} + y \tan x - y^3 \sec x = 0$

Command:

```
from sympy import Symbol,solve,Function,Derivative,dsolve
x,y=symbols('x,y')
y= Function("y")(x)
y1=Derivative(y,x)
z1=dsolve(Eq(y1+y*tan(x)-y**3*sec(x),0),y)
display(z1)
```

Output:

```
[Eq(y(x), -sqrt(1/(C1 - 2*sin(x)))*cos(x)),
 Eq(y(x), sqrt(1/(C1 - 2*sin(x)))*cos(x))]
```

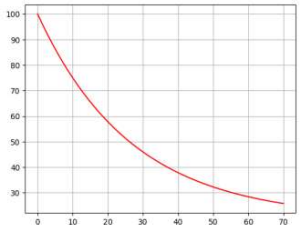
2 The temperature of a body drops from 100C to 75C in 10 minutes where the surrounding air is at the temperature 20 C. What will be the temperature of the body after half an hour? Plot the graph of cooling.

Command:

```
import numpy as np
from sympy import *
from matplotlib import pyplot as plt
t2=20   # surrounding temp
t1=100   # inital temp
# one reading t=1 minute temp is 75 degree
```

```
t=10
T=75
k1=(1/t)*log((t1-t2)/(T-t2))  # k calculation
print ('k=',k1 )
k= Symbol('k')
t= Symbol('t')
T= Function('T')(t)
T=t2+( t1-t2 )*exp(-k*t )  # solution
print ('T=',T)
# ploting the solution curve
T=T.subs (k , k1 )
T= lambdify (t , T )
t= np.linspace (0,70)
plt.plot (t,T(t),color='red')
plt.grid ()
plt.show ()
# When time t=30 minute T is
print('When time t=30 minute T is ,',T ( 30 ) ,'o C')
```

Output:




```
When time t=30 minute T is , 45.99609374999998 degree celicus
```

LAB 6: Programme to Double and Triple integration .

Syntax for the commands used:

To find integration

- integrate(function,(variable,minlimit,maxlimit))

1 Evaluate the integral $\int_0^1 \int_0^x (x^2 + y^2)\, dy\, dx$.

Command:

```
from sympy import *
x ,y , z= symbols ('x y z')
w1= integrate ( x ** 2+y ** 2 ,( y ,0 , x ) ,(x ,0 , 1 ) )
print ( w1 )
```

Output:

```
1/3
```

2 Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz)\, dz\, dy\, dx$.

Command:

```
from sympy import *
x= Symbol ('x')
y= Symbol ('y')
z= Symbol ('z')
w2= integrate (( x*y*z ) ,(z ,0 , 3-x-y ) ,(y ,0 , 3-x ) ,(x ,0 , 3 ) )
print ( w2 )
```

Output:

```
81/80
```

LAB 7: Evaluation of improper integrals, Beta and Gamma functions .

Syntax for the commands used:

1. To find Gamma
   - math.gamma(x)
2. To find Beta
   - math.beta(x,y)

1 Evaluate $\int_0^\infty e^{-x}\, dx$

Command:

```
from sympy import *
x= symbols ('x')
w1= integrate (exp (-x ) ,(x ,0 , float ('inf ') ) )
print ( simplify ( w1 ) )
```

Output:

```
1
```

2 Find Beta(3,5) ,Gamma(5).

Command:

```
from sympy import beta , gamma
m= input ('m :') ;
n= input ('n :') ;
m= float ( m ) ;
n= float ( n ) ;
s= beta (m , n ) ;
t= gamma ( n )
print ('gamma (',n ,') is %3.3f '%t )
print ('Beta (',m ,n ,') is %3.3f '%s )
```

Output:

```
m :3
n :5
gamma ( 5.0 ) is 24.000
Beta ( 3.0 5.0 ) is 0.010
```

3 Verify that Beta(m,n)=Gamma(m)Gamma(n) / Gamma(m+n) for m=5 and n=7.

Command:

```
from sympy import beta , gamma
m=5 ;
n=7 ;
m= float ( m ) ;
n= float ( n ) ;
s= beta (m , n ) ;
t=( gamma ( m )* gamma ( n ) )/ gamma ( m+n ) ;
print (s , t )
if (abs ( s-t )<=0.00001 ):
   print ('beta and gamma are related ')
else :
   print ('given values are wrong ')
```

Output:

```
0.000432900432900433 0.000432900432900433
beta and gamma are related
```

LAB 8: Numerical solution of system of equations, test for consistency.

Syntax for the commands used:

1. numpy.matrix(data, dtype = None)

   Returns a matrix from an array-like object, or from a string of data. A matrix is aspecialized 2-D array that retains its 2-D nature through operations.

2. numpy.linalg.matrix rank(A):

   Return rank of the array.

3. numpy.shape(A):

   Returns the shape of an array.

4. sympy.Matrix()

   Creates a matrix.

1. Check whether the following system homogenous linear equation has non –trivial solution $x_1 + 2x_2 - x_3 = 0, 2x_1 + x_2 + 4x_3 = 0, 3x_1 + 3x_2 + 4x_3 = 0$.

Commands:

```
import numpy as np
A=np.matrix([[1 ,2 ,- 1 ],[2 ,1 , 4 ],[3 ,3 , 4 ]])
B=np.matrix([[ 0 ],[ 0 ],[ 0 ]])
r=np.linalg.matrix_rank( A)
n=A.shape[ 1 ]
if (r==n):
   print("System has trivial solution")
   else :
      print("System has",n-r," non - trivial solution( s)")
```

Output:

System has trivial solution

2. Examine the consistency of the following system of equations and solve if consistent, $x_1 + 2x_2 - x_3 = 1, 2x_1 + x_2 + 5x_3 = 2, 3x_1 + 3x_2 + 4x_3 = 1$.

Commands:

```
import numpy as np
A=np.matrix([[1 ,2 ,- 1 ],[2 ,1 , 5 ],[3 ,3 , 4 ]])
B=np.matrix([[ 1 ],[ 2 ],[ 1 ]])
AB=np.concatenate (( A, B),axis= 1)
rA=np.linalg.matrix_rank(A)
rAB=np.linalg.matrix_rank(AB)
n=A. shape [ 1 ]
if (rA==rAB):
if (rA==n):
   print(" The system has unique solution ")
   print( np. linalg . solve (A, B))
   else :
   print(" The system has infinitely many solutions")
   else :
      print(" The system of equations is inconsistent")
```

Output:

The system of equations is inconsistent

LAB 9: Solution of system of linear equations by Gauss-Seidel method.

Objectives:

Use python

1. to check whether the given system is diagonally dominant or not.
2. to find the solution if the system is diagonally dominant.
3. Gauss Seidel method is an iterative method to solve system of linear equations.The method works if the system is diagonally dominant. That $is\ |aii| \geq \sum |aij|$ for all $i's$.

1 Solve system of equations by using Gauss seidel method
$20x + y - 2z = 17, 3x + 20y - z = -18, 2x - 3y + 20z = 25.$

Commands:

```
f1=lambda x,y,z:(17-y+2*z)/20
f2=lambda x,y,z:(-18-3*x+z)/20
f3=lambda x,y,z:(25-2*x+3*y)/20
# Initial setup
x0=0
y0=0
z0=0
count=1
# Reading tolerable error
e=float(input(' Enter tolerable error:'))
# Implementation of Gauss Seidel Iteration
print ('\ n Count\ tx\ ty\ tz\ n')
condition=True
while condition:
   x1=f1(x0, y0, z0)
   y1=f2(x1, y0, z0)
   z1=f3(x1, y1, z0)
print ('% d\ t% 0 .4f\ t% 0 .4f\ t% 0 .4f\ n' % (count, x1, y1, z1))
e1 =abs(x0 - x1 );
e2 =abs(y0 - y1 );
e3 =abs( z0 - z1 );
count += 1
x0=x1
y0=y1
z0=z1
condition=1>e and e2>e and e3 >e
print ('\ n Solution: x=% 0.3f, y=% 0.3f and z =% 0.3f\ n'% (x1, y1, z1))
```

Output:

Enter tolerable error:

0.001 Count	x	y	z
1	0.8500	-1.0275	1.0109
2	1.0025	-0.9998	0.9998
3	1.0000	-1.0000	1.0000

Solution: x=1.000, y=-1.000 and z = 1.000

LAB 10: Compute eigenvalues and corresponding eigen-vectorsFind dominant and corresponding eigen vector by Rayliegh power method.

Syntax for the commands used:

- np.linalg.eig(A)

  Compute the eigenvalues and right eigenvectors of a squarearray

- np.linalg.eigvals(A)

  Computes the eigenvalues of a non-symmetric array.

- np.array(parameter)

  Creates the array

- np.array([[1,2,3]])

  Is a one-dimensional array

- np.array([[1,2,3,6],[3,4,5,8],[2,5,6,1]]) is a multi-dimensional array
- lambda arguments:expression: Anonymous function or function without a name
  - This function can have any number of arguments but only one expression, which is evaluated and returned.
  - They are are syntactically restricted to a single expression.
  - Example: f=lambda $x : x * *2 - 3 * x + 1$ (Mathematically $f(x) = x^2 - 3x + 1$)
- np.dot(vector a, vector b): Returns the dot product of vectors a and b.

1. Obtain the eigen values and eigen vectors for the given matrix $\begin{bmatrix} 4 & 3 & 2 \\ 1 & 4 & 1 \\ 3 & 10 & 4 \end{bmatrix}$.

Commands:

```
import numpy as np
I=np.array([[4 ,3 , 2 ],[1 ,4 , 1 ],[3 , 10 , 4 ]])
print ("Given matrix:", I)
w,v=np.linalg.eig(I)        # x=np.linalg. eigvals(I)
print ("Eigen values:",w)
print ("Eigen vectors:",v)

# To  display one  eigen  value  and  corresponding eigen  vector

print (" Eigen value :", w[2])
print (" Corresponding Eigen vector :", v[:,0])
```

Output:

```
Given matrix:
[[ 4  3  2]
 [ 1  4  1]
 [ 3 10  4]]
Eigen values: [8.98205672 2.12891771 0.88902557]
Eigen vectors: [[-0.49247712 -0.82039552 -0.42973429]
 [-0.26523242 0.14250681 -0.14817858]
 [-0.82892584 0.55375355 0.89071407]]
 Eigen value: 0.8890255742950103
 Corresponding Eigen vector: [-0.49247712 -0.26523242 -0.82892584]
```

2. Compute the numerically largest eigen value of $p = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 3 & -1 \\ 2 & -1 & 3 \end{bmatrix}$ by power method.

Commands:

```
import numpy as np
def normalize( x):
    fac=abs(x).max()
    x_n =x/x.max()
    return fac,x_n
x=np.array([1,1,1])
a=np.array([[6 ,-2 , 2 ],[-2 ,3 ,- 1 ],[2 ,-1 , 3 ]])
for i in range (10):
    x = np.dot(a,   x)
    lambda_1,x=normalize ( x)
print(' Eigenvalue :',lambda_1)
print(' Eigenvector:',x)
```

Output:

```
Eigenvalue : 7.999988555930031
 Eigenvector: [ 1.   -0.49999785  0.50000072]
```