1.4 Functions

1. Exponential curve, $y = e^x$

```
# importing the required modules
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.001) # x takes the values between -10 and 10
                                    with a step length of 0.001
y = np.exp(x) # Exponential function
plt.plot(x,y)  # plotting the points
plt.title("Exponential curve ") # giving a title to the graph
plt.grid() # displaying the grid
plt.show() # shows the plot
```

1.3 Example: Plotting a line(Line plot)

```
# importing the required module
import matplotlib.pyplot as plt
x = [1,2,3,4,6,7,8] # x axis values
y = [2,7,9,1,5,10,3] # corresponding y axis values
plt.plot(x, y, 'r+--') # plotting the points
plt.xlabel('x - axis')  # naming the x axis
plt.ylabel('y - axis') # naming the y axis
plt.title('My first graph!') # giving a title to my graph
plt.show()  # function to show the plot
```

```
# importing the required module
import matplotlib.pyplot as plt

x = [1,2,3,4,6,7,8] # x axis values
y = [2,7,9,1,5,10,3] # corresponding y axis values
plt.scatter(x, y) # plotting the points
plt.xlabel('x - axis')  # naming the x axis
plt.ylabel('y - axis') # naming the y axis
plt.title('Scatter points') # giving a title to my graph
plt.show()
```

2. Sine and Cosine curves

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

x = np.arange(-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("Values of x")
plt.ylabel("Values of sin(x) and cos(x) ")
plt.grid()
plt.show()
```

```python
# A simple graph
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')  # Plot of y=x a linear curve
plt.plot(x, x**2, label='quadratic')  # Plot of y=x^2 a quadric curve
plt.plot(x, x**3, label='cubic')  # Plot of y=x^3 a cubic curve

plt.xlabel('x label')  # Add an x-label to the axes.
plt.ylabel('y label')  # Add a y-label to the axes.

plt.title("Simple Plot")  # Add a title to the axes.
plt.legend()  # Add a legend
plt.show() # to show the complete graph
```

1.5.1 Plot the following

1. Circle: $x^2 + y^2 = 5$

```python
# importing Sympy package with plot_implicit, symbols and Eq functions only
# symbols: used to declare variable as symbolic expression
# Eq: sets up an equation. Ex: 2x-y=0 is written as Eq(2*x-y,0)

from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p1 = plot_implicit(Eq(x**2 + y**2, 4),(x,-4,4),(y,-4,4),
                   title= 'Circle: $x^2+y^2=4$') # r= 2
```


1.5.1 Plot the following

1. Circle: $x^2 + y^2 = 5$

```python
# importing Sympy package with plot_implicit, symbols and Eq functions only
# symbols: used to declare variable as symbolic expression
# Eq: sets up an equation. Ex: 2x-y=0 is written as Eq(2*x-y,0)

from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p1 = plot_implicit(Eq(x**2 + y**2, 4),(x,-4,4),(y,-4,4),
                   title= 'Circle: $x^2+y^2=4$') # r= 2
```

1.6 Polar Curves

1. Circle: $r = p$,Where p is the radius of the circle

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

plt.axes(projection = 'polar')
r = 3
rads = np.arange(0, (2 * np.pi), 0.01)
# plotting the circle
for i in rads:
    plt.polar(i, r, 'g.')
plt.show()
```

3. Cardioid: $r = 5(1 + cos\theta)$

```
#Plot cardioid r=5(1+cos theta)
from pylab import *
theta=linspace(0,2*np.pi,1000)
r1=5+5*cos(theta)

polar(theta,r1,'r')
show()
```

4. Four leaved Rose: $r = 2|cos2x|$

```
#Plot Four Leaved Rose r=2 |cos2x|
from pylab import *
theta=linspace(0,2*pi,1000)
r=2*abs(cos(2*theta))
polar(theta,r,'r')
show()
```

5. Cardioids: $r = a + acos(\theta)$ and $r = a - acos(\theta)$

```
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)
# plotting the cardioid
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()
```

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

2.1

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

```
from sympy import *

r,t =symbols('r,t') # Define the variables required as symbols

r1=4*(1+cos(t)); #Input first polar curve
r2=5*(1-cos(t)); #Input first polar curve
dr1=diff(r1,t)  # find the derivative of first function
dr2=diff(r2,t)  # find the derivative of secodn function
t1=r1/dr1
t2=r2/dr2
q=solve(r1-r2,t)  # solve r1==r2, to find the point of intersection
                     between curves
w1=t1.subs({t:float(q[1])}) # substitute the value of "t" in t1
w2=t2.subs({t:float(q[1])})  # substitute the value of "t" in t2
y1=atan(w1)   # to find the inverse tan of w1
y2=atan(w2)  # to find the inverse tan of w2
w=abs(y1-y2)  # angle between two curves is abs(w1-w2)
print('Angle between curves in radians is %0.3f'%(w))
```

2. Find the angle between the curves $r = 4\cos t$ and $r = 5\sin t$.

```
from sympy import *
r,t =symbols('r,t')

r1=4*(cos(t));
r2=5*(sin(t));


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[0])})
w2=t2.subs({t:float(q[0])})

y1=atan(w1)
y2=atan(w2)
w=abs(y1-y2)
print('Angle between curves in radians is %0.4f'%float(w))
```

2.3 2. Radius of curvature

1. Find the radius of curvature, $r = 4(1 + \cos t)$ at t=$\pi/2$.

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

2. Find the radius of curvature for $r = asin(nt)$ at $t = pi/2$ and $n = 1$.

```
from sympy import *
t,r,a,n=symbols('t r a n')
r=a*sin(n*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)
rho1=rho1.subs(n,1)
print("The radius of curvature is")
display(simplify(rho1))
```

```
from sympy import *
from sympy.abc import rho, x,y,r,K,t,a,b,c,alpha  # define all symbols
                                    required
y=(sqrt(x)-4)**2
y=a*sin(t) #input the parametric equation
x=a*cos(t)
dydx=simplify(Derivative(y,t).doit())/simplify(Derivative(x,t).doit())
                                    # find the derivative of parametric
                                     equation
rho=simplify((1+dydx**2)**1.5/(Derivative(dydx,t).doit()/(Derivative(x,
                                    t).doit()))) #substitute the
                                    derivative in radius of curvature
                                    formula
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))
```

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

3.1 Objectives:

Use python

3.2 I. Partial derivatives

The partial derivative of $f(x, y)$ with respect to x at the point (x_0, y_0) is

1. Prove that mixed partial derivatives , $u_{xy} = u_{yx}$ for $u = exp(x)(xcos(y) - ysin(y))$.

```
from sympy import *
x,y =symbols('x y')

u=exp(x)*(x*cos(y)-y*sin(y)) # input mutivariable function u=u(x,y)
dux=diff(u,x) # Differentate u w.r.t x
duy=diff(u,y) # Differentate u w.r.t. y
duxy=diff(dux,y)  # or duxy=diff(u,x,y)
duyx=diff(duy,x)  # or duyx=diff(u,y,x)
# Check the condtion uxy=uyx
if duxy==duyx:
    print('Mixed partial derivatives are equal')
else:
    print('Mixed partial derivatives are not equal')
```

2. Prove that if $u = e^x(x\cos(y) - y\sin(y))$ then $u_{xx} + u_{yy} = 0$.

```
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))
```

3.3 II Jacobians

Let $x = g(u, v)$ and $y = h(u, v)$ be a transformation of the plane. Then the Jacobian of this transformation is

1. If $u = xy/z, v = yz/x, w = zx/y$ then prove that $J = 4$.

```
from  sympy import *

x,y,z=symbols('x,y,z')

u=x*y/z
v=y*z/x
w=z*x/y
# find the all first order partial derivates
dux=diff(u,x)
duy=diff(u,y)
duz=diff(u,z)

dvx=diff(v,x)
dvy=diff(v,y)
dvz=diff(v,z)

dwx=diff(w,x)
dwy=diff(w,y)
dwz=diff(w,z)

# construct the Jacobian matrix
J=Matrix([[dux,duy,duz],[dvx,dvy,dvz],[dwx,dwy,dwz]]);

print("The Jacobian matrix is \n")
display(J)

# Find the determinat of Jacobian Matrix
Jac=det(J).doit()
print('\n\n J = ', Jac)
```

2. If $u = x + 3y^2 - z^3, v = 4x^2yz, w = 2z^2 - xy$ then prove that at $(1, -1, 0)$, $J = 20$.

```
from  sympy import *

x,y,z=symbols('x,y,z')

u=x+3*y**2-z**3
v=4*x**2*y*z
w=2*z*z**2-x*y
dux=diff(u,x)
duy=diff(u,y)
duz=diff(u,z)

dvx=diff(v,x)
dvy=diff(v,y)
dvz=diff(v,z)

dwx=diff(w,x)
dwy=diff(w,y)
dwz=diff(w,z)

J=Matrix([[dux,duy,duz],[dvx,dvy,dvz],[dwx,dwy,dwz]]);

print("The Jacobian matrix is ")
display(J)

Jac=Determinant(J).doit()
print('\n\n J = \n')
display(Jac)

J1=J.subs([(x, 1), (y, -1), (z, 0)])

print('\n\n J at (1,-1,0):\n')
```

2. If $u = x + 3y^2 - z^3$, $v = 4x^2yz$, $w = 2z^2 - xy$ then prove that at $(1, -1, 0)$, $J = 20$.

```
from  sympy import *

x,y,z=symbols('x,y,z')

u=x+3*y**2-z**3
v=4*x**2*y*z
w=2*z*z**2-x*y
dux=diff(u,x)
duy=diff(u,y)
duz=diff(u,z)

dvx=diff(v,x)
dvy=diff(v,y)
dvz=diff(v,z)

dwx=diff(w,x)
dwy=diff(w,y)
dwz=diff(w,z)

J=Matrix([[dux,duy,duz],[dvx,dvy,dvz],[dwx,dwy,dwz]]);

print("The Jacobian matrix is ")
display(J)

Jac=Determinant(J).doit()
print('\n\n J = \n')
display(Jac)

J1=J.subs([(x, 1), (y, -1), (z, 0)])

print('\n\n J at (1,-1,0):\n')
Jac1=Determinant(J1).doit()
display(Jac1)
```

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

4.1 Objectives:

Use python

4.2 Maxima and minima problem

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

```
import sympy
from sympy import Symbol, solve, Derivative, pprint
x=Symbol('x')
y=Symbol('y')
f=x**2+x*y+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")
```

4.3 Taylor series expansion

1. Expand $\sin(x)$ **as Taylor series about** $x = pi/2$ **upto 3rd degree term. Also find** $\sin(100^0)$

```
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()
```

4.4 Maclaurin Series

2. Find the Maclaurin series expansion of $\sin(x) + \cos(x)$ upto 3rd degree term. Calculate $\sin(10) + \cos(10)$.

```
import numpy as np
from matplotlib import pyplot as plt
from sympy import *
x=Symbol('x')

y=sin(x)+cos(x)
format
x0=float(0)
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

def f
   re

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()
```

3. Prove that $\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x$=e

```
from sympy import *
from math import inf
x=Symbol('x')
l=Limit((1+1/x)**x,x,inf).doit()
display(l)
```

4.5 L'Hospital' rule

We can evaluate inderminate forms easily in python using Limit command

1. $\lim_{x\to 0}\frac{\sin(x)}{x}$

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

2. Evaluate $\lim_{x\to 1}\frac{((5x^4-4x^2-1)}{(10-x-9x^3)}$

```
from sympy import *
x=Symbol('x')
l=Limit((5*x**4-4*x**2-1)/(10-x-9*x**3),x,1).doit()
print(l)
```

3. Prove that $\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x$**=e**

```
from sympy import *
from math import inf
x=Symbol('x')
l=Limit((1+1/x)**x,x,inf).doit()
display(l)
```

4.6 Exercise:

Plot the following:

LAB 5: Solution of First order differential equation and ploting the solution curves

5.1 Objectives:

Use python

1. To find the solution of first order differential equations.
2. To represent the solution graphically.

Syntax for the commands used:

1. `dsolve()`

```
sympy.solvers.ode.dsolve(eq, func=None, hint='default', simplify=
                         True, ics=None, xi=None, eta=
                         None, x0=0, n=6, **kwargs)
```

Parameters

- `eq`: eq can be any supported ordinary differential equation (see the ode docstring for supported methods). This can either be an Equality, or an expression, which is assumed to be equal to 0.
- `func`: f(x) is a function of one variable whose derivatives in that variable make up the ordinary differential equation eq. In many cases it is not necessary to provide this; it will be autodetected (and an error raised if it could not be detected).
- `hint`: hint is the solving method that you want dsolve to use. Use `classify_ode(eq, f(x))` to get all of the possible hints for an ODE. The default hint, default, will use whatever hint is returned first by `classify_ode()`. See Hints below for more options that you can use for hint.
- `simplify`: simplify enables simplification by odesimp(). See its docstring for more information. Turn this off, for example, to disable solving of solutions for func or simplification of arbitrary constants. It will still integrate with this hint. Note that the solution may contain more arbitrary constants than the order of the ODE with this option enabled.
- `xi and eta`: are the infinitesimal functions of an ordinary differential equation. They are the infinitesimals of the Lie group of point transformations for which the differential equation is invariant. The user can specify values for the infinitesimals. If nothing is specified, xi and eta are calculated using infinitesimals() with the help of various heuristics.
- `ics`: is the set of initial/boundary conditions for the differential equation.It should be given in the form of {`f(x0): x1, f(x).diff(x).subs(x, x2): x3`} and so on. For power series solutions, if no initial conditions are specified f(0) is assumed to be C0 and the power series solution is calculated about 0.

- `x0`: is the point about which the power series solution of a differential equation is to be evaluated.
- `n`: gives the exponent of the dependent variable up to which the power series solution of a differential equation is to be evaluated. also be much faster than all, because integrate() is an expensive routine.
- **Usage:**
  - Solves any kind of ordinary differential equation and system of ordinary differential equations.
  - Usage `dsolve(eq, f(x), hint)` −> Solve ordinary differential equation eq for function f(x), using method hint.

2. `odeint()`: The odeint (ordinary differential equation integration) library is a collection of advanced numerical algorithms to solve initial-value problems.

```
y = odeint(model, y0, t)
```

Parameters:

- `model`: Function name that returns derivative values at requested y and t values as `dydt = model(y,t)`
- `y0`: Initial conditions of the differential states
- `t`: Time points at which the solution should be reported.

3. `linspace()`:

```
linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=
                     None, axix=0)
```

Prameters

- `start`: It represents the starting value of the sequence.
- `stop`: It represents the ending value of the sequence.
- `num`: It generates a number of samples. The default value of num is 50 and it must be a non-negative number. It is of int type and can be optional.
- `endpoint`: By default its value is True. If we take it as False then the value can be excluded from the sequence. It is of bool type and can be optional.
- `retstep`: If its True then it returns samples and step value where the step is the spacing between the samples.
- `dtype(data type)`: It represents the type of the output array. It can also be optional.
- `axis`: The axis is the result to store the samples. It is of int type and can be optional.

1. Solve : $\frac{dP(t)}{dt} = r$.

```
from sympy import *
init_printing()

t,r = symbols('t,r')  # Define the symbols
P = Function('P')(t)  # define function
C1 = Symbol('C1')

print("\nDifferential Equation")
DE1=Derivative(P, t, 1)-r   # define the differeentail equation
display(DE1)

# General solution
print("\nGeneral Solution")

GS1=dsolve(DE1)  # Solve the differentail equation
display(GS1)  # Display the solution

print("\nParticular Solution")
PS1=GS1.subs({C1:2})  # substitute the value of the conastant
display(PS1)
```


2: Solve: $\frac{dy}{dx} + tanx - y^3secx = 0$.

```
from sympy import *

x,y=symbols('x,y')
y=Function("y")(x)

y1=Derivative(y,x)
z1=dsolve(Eq(y1+y*tan(x)-y**3*sec(x)),y)

display(z1)
```

3: Solve: $x^3\frac{dy}{dx} - x^2y + y^4cosx = 0$.

```
from sympy import *

x,y=symbols('x,y')
y=Function("y")(x)
y1=Derivative(y,x)
z1=dsolve(Eq(x**3*y1-x**2*y+y**4*cos(x),0),y)
display(z1)
```

5.2 Solution curves

Solving IVP using odeint:

1. Solve $\frac{dy}{dt} = -ky$ with parameter $k = 0.3$ and $y(0) = 5$.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Function returns dy/dt

def model(y,t):
    k=0.3
   # dydt=-k*y
    return -k*y

# initial condition

y0=5

# values for time
t=np.linspace(0,20)

# solve ODE
y= odeint(model,y0,t)

plt.plot(t,y)
plt.title('Solution of dy/dt=-ky; k=0.3, y(0)=5')
plt.xlabel('time')
plt.ylabel('y(t)')
plt.show()
```

2. Simulate $\tau \frac{dy}{dt} = -y + K_p u$; $K_p = 3.0$, $\tau = 2.0$.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

Kp=3
taup=2

# Differential Equation:

def model(y,t):
    u = 1
    return (-y + Kp * u)/taup

t3 = np.linspace(0,14,100)

# ODE integrator
y3 = odeint(model,0,t3)

plt.plot(t3,y3,'r-',linewidth=1,label='ODE Integrator')
plt.xlabel('Time')
plt.ylabel('Response (y)')
plt.legend(loc='best')
plt.show()
```

3. Application problem

A culture initially has P_0 number of bacteria. At $t = 1$ hour the number of bacteria is measured to be $\frac{3}{2}P_0$. If the rate of growth is proportional to the number of bacteria $P(t)$ present at time t, determine the time necessary for the number of bacteria to triple.

The differential equation is : $\frac{dp}{dt} = kp; P(1) = \frac{3}{2}p_0$.

The solution is : $y = P_0e^{0.405465108108164t}$, $y_0 = 20$.

```
from pylab import *
t=arange(0,10,0.5)  # Define the range where we want solution
P0=20
y=20*exp(0.405465108108164*t)
plot(t,y)
xlabel('Time')
ylabel('no of bacteria')
title('Law of Natural Growth')
show()
```

4. Newton's Law of cooling

Solving Newton's law of cooling by solution. The solution of mathematical representation of Newton's Law of cooling is,$T = t_2 + (t_1 - t_2)e^{-kt}$, where, T=temperature at any time t, t_1 = Initial temperature, t_2 = surrounding temperature, k = thermal conductivity of the material.

1. The temperature of a body drops from 100 C to 75 C 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.


```
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')
```

5.3 Exercise:

Plot the following:

1. Solve $ysinxdx - (1 + y^2 + cos^2x)dy = 0$.
   Ans: $(1/2)ycos2x + (3/2)y + y^3/3 = 0$
2. Solve $\frac{dy}{dx} = x + y$ subject to condtion $y(0) = 2$.
   Ans: $y = 3e^x - x - 1$
3. Solve $\frac{dy}{dx} = x^2$ subject to condtion $y(0) = 5$.
   Ans:$y = x^3/3 + 5$
4. Solve $x^2y' = ylog(y) - y'$.
   Ans:$y(x) = e^{C_1 tan^{-1}(x)}$
5. Solve $y' - y - xe^x = 0$.
   Ans:$y(x) = \left(C_1 + \frac{x^2}{2}\right) e^x$

LAB 8: Numerical solution of system of equations, test for consistency and graphical representation of the solution.

8.1 Objectives:

Use python

1. to find solution of system of equations numerically.
2. to test for consistency and represent the solution graphically.

Syntax for the commands used:

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

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

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

2. `numpy.linalg.matrix_rank(A)`:

```
numpy.linalg.matrix_rank(A)
```

Return rank of the array.

3. `numpy.shape(A)`:

```
numpy.shape(A)
```

Returns the shape of an array.

4. `sympy.Matrix()`

```
sympy.Matrix()
```

Creates a matrix.

8.2 Solution of system of equations

System of homogenous linear equations:

The linear system of equations of the form $AX = 0$ is called system of homogenous linear equations. ¡br¿ The n-tuple $(0, 0, \ldots, 0)$ is a trivial solution of the system. ¡br¿ The homogeneous system of m equations $AX = 0$ in n unknowns has a non trivial solution if and only if the rank of the matrix A is less than n. Further if $\rho(A) = r < n$, then the system possesses $(n - r)$ linearly independent solutions.

Example 1:

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

```
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)")
```

```
System has trivial solution
```

Example 2:

Check whether the following system of homogenous linear equation has non-trivial solution. $x_1 + 2x_2 - x_3 = 0, \qquad 2x_1 + x_2 + 4x_3 = 0, \qquad x_1 - x_2 + 5x_3 = 0.$

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[1,-1,5]])
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)")
```



```
System has 1 non-trivial solution(s)
```

8.3 System of Non-homogenous Linear Equations

The linear system of equations of the form $AX = B$ is called system of non-homogenous linear equations if not all elements in B are zeros.

The non homogeneous system of m equations $AX = B$ in n unknowns is

- consistent (has a solution) if and only if, $\rho(A) = \rho([A|B])$
- has unique solution, $\rho(A) = n$
- has infintely many solutions, $\rho(A) < n$
- system is inconsistent $\rho(A) \neq \rho([A|B])$.

Example 3:

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

```
A=np.matrix([[1,2,-1],[2,1,4],[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")
```

```
The system has unique solution
[[ 7.]
 [-4.]
 [-2.]]
```

Example 4:

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

```
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")
```

```
The system of equations is inconsistent
```

Alternate method for the above problem using sympy package

```
import sympy as sp
x, y, z=sp.symbols('x y z')
```

```
A=sp.Matrix([[1,2,-1],[2,1,5],[3,3,4]])
B=sp.Matrix([[1],[2],[1]])
AB=A.col_insert(A.shape[1],B)
rA=A.rank()
rAB=AB.rank()
n=A.shape[1]
print("The coefficient matrix is")
sp.pprint(A)
print(f"The rank of the coefficient matrix is {rA}")
print("The augmented matrix is")
sp.pprint(AB)
print(f"The rank of the augmented matrix is {rAB}")
print(f"The number of unkowns are {n}")
if (rA==rAB):
    if (rA==n):
        print("The system has unique solution")
    else:
        print("The system has infinitely many solutions")
    print(sp.solve_linear_system(AB,x,y,z))
else:
    print("The system of equations is inconsistent")
```

8.4 Graphical representation of solution

Example 5:

Obain the solution of $3x + 5y = 1; x + y = 1$ graphically.

```
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

x,y=symbols('x,y')
sol=solve([3*x+5*y-1,x+y-1],[x,y])
p=sol[x]
q=sol[y]

print('Point of intersection is A (', p ,',', q, ')\n')
x = np.arange(-10, 10, 0.001)

y1 = (1-3*x)/5
y2=1-x

plt.plot(x,y1,x,y2)
plt.plot(p,q,marker = 'o')

plt.annotate('A', xy=(p,q), xytext=(p+0.5, q))
plt.xlim(-5,7)
plt.ylim(-7,7)
plt.axhline(y=0)
plt.axvline(x=0)
plt.title("$3x+5y=1; x+y=1$")
plt.xlabel("Values of x")
plt.ylabel("Values of y ")
```

```
plt.legend(['$3x+5y=1$', '$x+y=1$'])
plt.grid()
plt.show()
```

```
Point of intersection is A ( 2 , -1 )
```

Example 6:

Obtain the solution of $2x + y = 7; 3x - y = 3$ graphically.

```
from sympy import *
import numpy as np
import matplotlib.pyplot as plt

x,y=symbols('x,y')
sol=solve([2*x+y-7,3*x-y-3],[x,y])
p=sol[x]
q=sol[y]

print('Point of intersection is A (', p ,',', q, ')\n' )
x = np.arange(-10, 10, 0.001)

y1 = 7-2*x
y2=3*x-3

plt.plot(x,y1,'r')
plt.plot(x,y2,'g')

plt.plot(p,q,marker = 'o')

plt.annotate('A', xy=(p,q), xytext=(p+0.5, q))
plt.xlim(-5,7)
plt.ylim(-7,7)
plt.axhline(y=0)
plt.axvline(x=0)
plt.title("$2x+y=7; 3x-y=3$")
plt.xlabel("Values of x")
plt.ylabel("Values of y ")

plt.legend(['$2x+y=7$', '$3x-y-3$'])

plt.grid()
plt.show()
```

```
Point of intersection is A ( 2 , 3 )
```

8.5 Exercise:

1. Find the solution of the system homogeneous equations $x+y+z = 0$, $2x+y-3z = 0$ and $4x - 2y - z = 0$.
   Ans: The system has trivial solution.

2. Find the solution of the system non-homogeneous equations $25x + y + z = 27$, $2x + 10y - 3z = 9$ and $4x - 2y - 12z = -10$.
   Ans: [1, 1, 1]

3. Find the solution of the system non-homogeneous equations $x + y + z = 2$, $2x + 2y - 2z = 4$ and $x - 2y - z = 5$.
   Ans:[3,-1,0]

4. Check whether the following system of equations are consistent.
   a. $x + y + z = 2$, $2x + 2y - 2z = 6$ and $x - 2y - z = 5$.
   b. $2x + y + z = 4$, $4x + 2y - 2z = 8$ and $4x + 22y + 2z = 5$.
   Ans: a. Consistent, b. Inconsistent

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

9.1 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.

Gauss Seidel method is an iterative method to solve system of linear equations. The method works if the system is diagonally dominant. That is $|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|$ for all $i's$.

Example 1:

Solve the system of equations using Gauss-Seidel method: $20x+y-2z = 17; 3x+20y-z = -18; 2x-3y+20z = 25$.

```
# Gauss Seidel Iteration
# Defining equations to be solved
# in diagonally dominant form
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('\nCount\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 = e1>e and e2>e and e3>e

print('\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n'% (x1,y1,z1))
```

```
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
```

Example 2:

Solve $x + 2y - z = 3; 3x - y + 2z = 1; 2x - 2y + 6z = 2$ by Gauss-Seidel Iteration method.

```
# Defining equations to be solved
# in diagonally dominant form
f1 = lambda x,y,z: (1+y-2*z)/3
f2 = lambda x,y,z: (3-x+z)/2
f3 = lambda x,y,z: (2-2*x+2*y)/6

# Initial setup
x0,y0,z0 = 0,0,0

# Reading tolerable error
e = float(input('Enter tolerable error: '))
# Implementation of Gauss Seidel Iteration
print('\t Iteration\t x\t y\t z\n')
for i in range(0,25):
    x1 = f1(x0,y0,z0)
    y1 = f2(x1,y0,z0)
    z1 = f3(x1,y1,z0)
    #Printing the values of x, y, z in ith iteration
    print('%d\t%0.4f\t%0.4f\t%0.4f\n' %(i, x1,y1,z1))
    e1 = abs(x0-x1);
    e2 = abs(y0-y1);
    e3 = abs(z0-z1);

    x0 = x1
    y0 = y1
    z0 = z1

    if e1>e and e2>e and e3>e:
        continue
    else:
```

```
        break

print('\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n'% (x1,y1,z1))
```

```
Enter tolerable error: 0.001
         Iteration          x          y          z

0        0.3333        1.3333        0.6667


1        0.3333        1.6667        0.7778



Solution: x=0.333, y=1.667 and z = 0.778
```

Example 3:

Apply Gauss-Siedel method to solve the system of equations: $20x + y - 2z = 17; 3x + 20y - z = -18; 2x - 3y + 20z = 25$.

```
from numpy import *
def seidel(a, x ,b):
    #Finding length of a(3)
    n = len(a)
    # for loop for 3 times as to calculate x, y , z
    for j in range(0, n):
        # temp variable d to store b[j]
        d = b[j]

        # to calculate respective xi, yi, zi
        for i in range(0, n):
            if(j != i):
                d=d-a[j][i] * x[i]
        # updating the value of our solution
        x[j] = d / a[j][j]
    # returning our updated solution
    return x
a=array([[20.0,1.0,-2.0],[ 3.0,20.0,-1.0],[2.0,-3.0,20.0]])
x=array([[0.0],[0.0],[0.0]])
b=array([[17.0],[-18.0],[25.0]])
for i in range(0, 25):
    x = seidel(a, x, b)
print(x)
```

```
[[ 1.]
 [-1.]
 [ 1.]]
```

Note: In the next example we will check whether the given system is diagonally dominant or not.

Example 4:

Solve the system of equations $10x + y + z = 12; x + 10y + z = 12; x + y + 10z = 12$ by Gauss-Seidel method.

```
from numpy import *
import sys
#This programme will check whether the given system is diagonally dominant or not

def seidel(a, x ,b):
    #Finding length of a(3)
    n = len(a)
    # for loop for 3 times as to calculate x, y , z
    for j in range(0, n):
        # temp variable d to store b[j]
        d = b[j]

        # to calculate respective xi, yi, zi
        for i in range(0, n):
            if(j != i):
                d=d-a[j][i] * x[i]
        # updating the value of our solution
        x[j] = d/a[j][j]
    # returning our updated solution
    return x
a=array([[10.0,1.0,1.0],[ 1.0,10.0,1.0],[1.0,1.0,10.0]])
x=array([[1.0],[0.0],[0.0]])
b=array([[12.0],[12.0],[12.0]])

# We shall check for diagonally dominant
for i in range(0,len(a)):
  asum=0
  for j in range(0,len(a)):
    if (i!=j):
      asum=asum+abs(a[i][j])

  if(asum<=a[i][i]):
    continue
  else:

    sys.exit("The system is not diagonally dominant")

for i in range(0, 25):
    x = seidel(a, x, b)
print(x)
# Note here that the inputs if float gives the output in float.
```



```
[[1.]
 [1.]
 [1.]]
```

Note: In the next example, the Upper triangular matrix is calculated by the numpy function for finding lower triangular matrix. this upper triangular matrix is multiplied by the chosen basis function and subtracted by the rhs B column matrix. the new x found is the product of inverse(lower triangular matrix) and the B-UX. This program is available on github

Example 5:

Apply Gauss-Siedel method to solve the system of equations: $5x-y-z=-3; x-5y+z=-9; 2x+y-4z=-15$.

```
import numpy as np
from scipy.linalg import solve

def gauss(A, b, x, n):

    L = np.tril(A)
    U = A - L
    for i in range(n):
        xnew = np.dot(np.linalg.inv(L), b - np.dot(U, x))
        x=xnew
    print(x)
#          print(x)
    return x

'''___MAIN___'''

A = np.array([[5.0, -1.0, -1.0], [1.0, -5.0, 1.0], [2.0, 1.0, -4.0]])
b = [-3.0,-9.0,-15.0]
x = [1, 0, 1]

n = 20

gauss(A, b, x, n)
solve(A, b)
```

```
[1. 3. 5.]
array([1., 3., 5.])
```


9.2 Exercise:

1. Check whether the following system are diagonally dominant or not
   a. $25x+y+z=27$, $2x+10y-3z=9$ and $4x-2x-12z=-10$.
   b. $x+y+z=7$, $2x+y-3z=3$ and $4x-2x-z=-1$.

   Ans: a. Yes   b. No

2. Solve the following system of equations using Gauss-Seidel Method.
   a. $4x+y+z=6$, $2x+5y-2z=5$ and $x-2x-7z=-8$.
   b. $27x+6y-z=85$, $6x+15y+2z=72$ and $x+y+54z=110$

   Ans: a. [1,1,1]   b. [2.42, 3.57, 1.92]

LAB 10: Compute eigenvalues and corresponding eigenvectors. Find dominant and corresponding eigenvector by Rayliegh power method.

10.1 Objectives:

Use python

1. to find eigenvalues and corresponding eigenvectors.
2. to find dominant and corresponding eigenvector by Rayleigh power method.

Syntax for the commands used:

1. `np.linalg.eig(A)`: Compute the eigenvalues and right eigenvectors of a square array

   ```
   np.linalg.eig(A)
   ```

   Returns the following:

   - w(. . . , M) array
     The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When a is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs.
   - v(. . . , M, M) array
     The normalized (unit "length") eigenvectors, such that the column v[:,i] is the eigenvector corresponding to the eigenvalue w[i].

2. `np.linalg.eigvals(A)`: Computes th eigenvalues of a non-symmetric array.

3. `np.array(parameter)`: Creates ndarray

   - `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

4. `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 $f(x) = x^2 - 3x + 1$)

5. `np.dot(vector_a, vector_b)`: Returns the dot product of vectors a and b.

10.2 Eigenvalues and Eigenvectors

Eigenvector of a matrix A is a vector represented by a matrix X such that when X is multiplied with matrix A, then the direction of the resultant matrix remains same as vector X.

Example 1:

Obtain the eigen values and eigen vectors for the given matrix.

$$\begin{bmatrix} 4 & 3 & 2 \\ 1 & 4 & 1 \\ 3 & 10 & 4 \end{bmatrix}.$$

```
import numpy as np
I=np.array([[4,3,2],[1,4,1],[3,10,4]])
print("\n Given matrix: \n", I)

#x=np.linalg.eigvals(I)
w,v = np.linalg.eig(I)

print("\n Eigen values: \n", w)

print("\n Eigen vectors: \n", v)


## To display one eigen value and correspondingeigen vector

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

```
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:
  8.982056720677654

 Corresponding Eigen vector : [-0.49247712 -0.26523242 -0.82892584]
```

Example 2:

Obtain the eigen values and eigen vectors for the given matrix.

$$A = \begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix}.$$

```
import numpy as np
I=np.array([[1,-3,3],[3,-5,3],[6,-6,4]])

print("\n Given matrix: \n", I)

w,v = np.linalg.eig(I)

print("\n Eigen values: \n", w)

print("\n Eigen vectors: \n", v)
```

```
Given matrix:
 [[ 1 -3  3]
 [ 3 -5  3]
 [ 6 -6  4]]

 Eigen values:
 [ 4.+0.00000000e+00j -2.+1.10465796e-15j -2.-1.10465796e-15j]

 Eigen vectors:
 [[-0.40824829+0.j          0.24400118-0.40702229j  0.24400118+0.40702229j]
 [-0.40824829+0.j         -0.41621909-0.40702229j -0.41621909+0.40702229j]
```


10.3 Largest eigenvalue and corresponding eigenvector by Rayleigh method

For a given Matrix A and a given initial eigenvector X_0, the power method goes as follows: Consider AX_0 and take the largest number say λ_1 from the column vector and write $AX_0 = \lambda_1 X_1$. At this stage , λ_1 is the approximate eigenvalue and X_1 will be the corresponding eigenvector. Now multiply the Matrix A with X_1 and continue the iteration. This method is going to give the dominant eigenvalue of the Matrix.

Example 4:

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

```
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)
```

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

Example 5:

Compute the numerically largest eigenvalue of $P = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 1 \end{bmatrix}$ by power method.

```
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([[1,1,3 ],
             [1,5,1],[3,1,1]])

for i in range(10):
    x = np.dot(a, x)
    lambda_1, x = normalize(x)

print('Eigenvalue:', lambda_1)
print('Eigenvector:', x)
```



```
Eigenvalue: 6.001465559355154
Eigenvector: [0.5003663 1.        0.5003663]
```

10.4 Exercise:

1. Find the eigenvalues and eigenvectors of the following matrices

   a. $P = \begin{bmatrix} 25 & 1 \\ 1 & 3 \end{bmatrix}$

   Ans. Eigenvalues are 25.04536102 and 2.95463898; and corresponding eigenvectors are [0.99897277 $\;-0.04531442$] and [0.04531442 0.99897277].

   b. $P = \begin{bmatrix} 25 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & -4 \end{bmatrix}$

   Ans. Eigenvalues are 25.18215138, -4.13794129 and 2.95578991; and corresponding eigenvectors are [0.9966522 0.06880398 0.04416339], [0.04493037 −0.00963919 −0.99894362] and [0.0683056 −0.99758363 0.01269831].

c. $P = \begin{bmatrix} 11 & 1 & 2 \\ 0 & 10 & 0 \\ 0 & 0 & 12 \end{bmatrix}$

Ans. Eigenvalues are 11., 10. and 12.; and corresponding eigenvectors are [1. −0.70710678 0.89442719], [0. 0.70710678 0.], and [0. 0. 0.4472136].

d. $P = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 12 \end{bmatrix}$

Ans. Eigenvalues are 12.22971565, 3.39910684 and 1.37117751; and eigenvectors are [−0.11865169 −0.85311963 0.50804396], [−0.10808583 −0.49752078 −0.86069189] and [−0.98703558 0.1570349 0.03317846].

2. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 25 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & -4 \end{bmatrix}$ by power method.

   Take $X_0 = (1, 0, 1)^T$.
   Ans. 25.182151221680012

3. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 6 & 1 & 2 \\ 1 & 10 & -1 \\ 2 & 1 & -4 \end{bmatrix}$ by power method.

   Take $X_0 = (1, 1, 1)^T$.
   Ans. 10.107545112667367

4. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 5 & 1 & 1 \\ 1 & 3 & -1 \\ 2 & -1 & -4 \end{bmatrix}$ by power method.

   Take $X_0 = (1, 0, 0)^T$.
   Ans. 5.544020973078026

Computer Science and Engineering Stream

LAB 6: Finding GCD using Euclid's algorithm.

6.1 Objectives:

Use python

1. to find the GCD of two given integers by Euclid's algorithm

2. to check whether given two integers are relatively prime or not.

Euclidean algorithm

is useful to find GCD of two numbers. The algorithm is as follows:

The two numbers a and b can be assumed positive such that $a < b$. Let r_1 be the remainder when b is divided by a. Then $0 \leq r_1 < a$. That is $b = ak_1 + r_1$.

Now let r_2 be the remainder when a is divided by r_1. That is $a = r_1k_2 + r_2$. Where $0 \leq r_2 < r_1$. Continue this process of dividing each divisor by the next remainder. At some stage we obtain remainder 0. The **last non-zero remainder is the GCD** of a and b. This is known as Euclid's algorithm.

Algorithm analysis:

1. Recursive process - operations are repeated till **stopping criterion** is reached

2. The **output** of one step is used as the **input of the next step**.

Example 1:

Find the GCD of (614,124).


```
"""
The function is named "gcd1", which takes as inputs two numbers:
1. 'a', and
2. 'b'
where, a < b.
In case the first number is larger than the second number, the function
will interchange the numerals. The answer however remains unchanged.
"""

def gcd1(a,b):
    c = 1 # Assume non-zero remainder
    if b < a: # Preprocessing of input
        t = b # Temporary variable 't' used to swap values of 'a' and '
                                            b'
        b = a
        a = t
    while (c > 0): # Condition checked: Is the remainder non-zero?
        c = b%a
        print(a,c) # Display divisor and remainder
        b = a
```

```
        a = c
        continue # This command gets activated whenever 'while' is TRUE
    """
    At this stage, 'while' loop no longer works because 'c > 0' is
                                    FALSE.
    Remainders can't be negative, so the
    """
    print('GCD =',b)
gcd1(614,124)
```

```
124 118
118 6
6 4
4 2
2 0
GCD = 2
```

Relatively prime

Two numbers a and b are called **relatively prime** or **co-prime** if their GCD (also known as HCF) is equal to 1.

For example: 2 and 19 are relatively prime, because 1 is the largest natural number that divides **both** 2 and 19.

Example 2:

Prove that 163 and 512 are relatively prime.

```
def gcd1(a,b):
    c=1;
    if b <a:
        t=b;
        b=a;
        a=t;
    while (c>0):
        c=b%a;
        print(a,c);
        b=a;
        a=c;
        continue
    print('GCD= ',b);
gcd1(163,512)
```

```
163 23
23 2
2 1
1 0
GCD=  1
```

Divides

If GCD of a and b is a, then a divides b.

Note that when GCD(a, b) $= a$ is equivalent to the statement a is that the largest natural number that divides both a and b.

For example: The GCD of 4 and 8 is 4, as 4 is the largest number that divides both 4 and 8. Since 4 is one of the given numbers, 4 divides 8.

Example 4:

Prove that 8 divides 128.

```
def gcd1(a,b):
    c=1;
    if b <a:
        t=b;
        b=a;
        a=t;
    while (c>0):
        c=b%a;
        print(a,c);
        b=a;
        a=c;
        continue
    print('GCD = ',b);
gcd1(8,128)
```

```
8 0
GCD=  8
```

Example 5:

Calculate GCD of (a,b) and express it as linear combination of a and b. Calculate GCD=d of 76 and 13 , express th GCD as $76x + 13y = d$

```
from sympy import *
a=int(input('enter the first number :'))
b=int(input('enter the second number :'))
s1=1;
s2=0;
t1=0;
t2=1;
r1=a;
r2=b;
r3=(r1%r2);
q = (r1-r3)/r2;
s3=s1-s2*(q);
t3=t1-t2*q;

while (r3!=0):
    r1=r2;
    r2=r3;
    s1=s2;
```

```
    s2=s3;
    t1=t2;
    t2=t3;
    r3=(r1%r2);
    q = (r1-r3)/r2;
    s3=s1-s2*(q);
    t3=t1-t2*q;


print('the GCD of ',a,' and',b,'is',r2);
print('%d x %d + %d x %d = %d\n'%(a,s2,b, t2,r2));
```

```
enter the first number :76
enter the second number :13
the GCD of  76  and 13 is 1
76 x 6 + 13 x -35 = 1
```

Note:

`SymPy` is a Python library for symbolic mathematics and has an inbuilt command for GCD.

The functions `gcd` and `igcd` can be imported to compute the GCD of numbers.

```
from sympy import gcd
gcd(1235,2315)
```

```
5
```

```
from sympy import igcd
igcd(3228,93)
```

```
3
```


6.2 Exercise:

1. Find the GCD of 234 and 672 using Euclidean algorithm.
   Ans: 6

2. What is the largest number that divides both 1024 and 1536?
   Ans: 512

3. Find the greatest common divisor of 6096 and 5060?
   Ans: 4

4. Prove that 1235 and 2311 are relatively prime.
   Ans: Sketch of proof: if largest common divisor is one, then numbers are relatively prime (or coprime); and vice versa.

5. Are 9797 and 7979 coprime?
   Ans: No, their gcd is 101

6. Write a function in Python to compute the greatest common divisor of 15625 and 69375.
   Alternate tip: `SymPy` is a library (module) providing `gcd` function
   Advanced tip: `from sympy.abc import x` allows to find GCD of algebraic expressions.

7. Using a Python module, find the GCD of 4096 and 6144.
   Ans: A sample program is as below:

```
from sympy import *
#from sympy import gcd
answer7 = gcd(4096, 6144)
answer7a = gcd(6144, 4096)
print ('GCD =', answer7, '(1st method),', answer7a (2nd method)')
# Desired outcome: GCD = 2048
```

LAB 7: Solving linear congruence of the form $ax \equiv b(\mod m)$.

7.1 Objectives:

Use python

1. to find solution of linear congruence.
2. to find multiplicative inverse of a mod p.

Example 1:

Show that the linear congruence $6x \equiv 5(\mod 15)$ has no solution.

```
from sympy import *
from math import*5

a=int(input('enter integer a ')); #7
b=int(input('enter integer b ')); #9
m=int(input('enter integer m ')); #15
d=gcd(a,m)
if (b%d!=0):#Reminder calculation
    print('the congruence has no integer solution');
else:
    for i in range(1,m-1):
        x=(m/a)*i+(b/a)
        if(x//1==x):#check whether x is an integer
            print('the solution of the congruence is ', x)
            break
```

```
enter integer a 6
enter integer b 5
enter integer m 15
the congruence has no integer solution
```


Example 2:

Find the solution of the congruence $5x \equiv 3(\text{mod } 13)$.

```
from sympy import *
#Linear congruence
#Consider ax=b(mod m),x is called the solution of the congruce

a=int(input('enter integer a ')); #7
b=int(input('enter integer b ')); #9
m=int(input('enter integer m ')); #15
d=gcd(a,m)
if (b%d!=0):
    print('the congruence has no integer solution');
else:
    for i in range(1,m-1):
        x=(m/a)*i+(b/a)
```

```
        if(x//1==x):#check whether x is an integer
            print('the solution of the congruence is ', x)
            break
```

```
enter integer a 5
enter integer b 3
enter integer m 13
the solution of the congruence is  11.0
```

Note:

The solution of the congruence $ax \equiv 1(\text{mod } p)$ is called multiplicative inverse of a mod p.

Example 4:

Find the inverse of 5 mod 13.

```
from sympy import gcd
#Linear congruence
#Consider ax=b(mod m),x is called the solution of the congrunce

a=int(input('enter integer a ')); #7
b=int(input('enter integer b ')); #9
m=int(input('enter integer m ')); #15
d=gcd(a,m)
if (b%d!=0):
    print('the congruence has no integer solution');
else:
    for i in range(1,m-1):
        x=(m/a)*i+(b/a)
        if(x//1==x):#check whether x is an integer
            print('the solution of the congruence is ', x)
            break
```



```
enter integer a 5
enter integer b 1
enter integer m 13
the solution of the congruence is  8.0
```

7.2 Exercise:

1. Find the solution of the congruence $12x \equiv 6(\text{ mod } 23)$.
   Ans: 12

2. Find the multiplicative inverse of 3 mod 31.
   Ans: 21

3. Prove that $12x \equiv 7(\text{ mod } 14)$ has no solution. Give reason for the answer.
   Ans: Because GCD(12,14)=2 and 2 doesnot divide 7.

Electrical & Electronics Engineering Stream

LAB 6: Programme to compute area, volume and center of gravity

6.1 Objectives:

Use python

1. to evaluate double integration.
2. to compute area and volume.
3. to calculate center of gravity of 2D object.

Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

6.2 Double and triple integration

Example 1:

Evaluate the integral $\int_0^1 \int_0^x (x^2+y^2)dydx$

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

```
1/3
```

Example 2:

Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz)dzdydx$

```
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)
```

```
81/80
```

Example 3:

Prove that $\int\int(x^2+y^2)dydx = \int\int(x^2+y^2)dxdy$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w3=integrate(x**2+y**2,y,x)
pprint(w3)
w4=integrate(x**2+y**2,x,y)
pprint(w4)
```

6.3 Area and Volume

Area of the region R in the cartesian form is $\int_R\int dxdy$

Example 4:

Find the area of an ellipse by double integration. A=4 $\int_0^a \int_0^{(b/a)\sqrt{a^2-x^2}} dydx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
#a=Symbol('a')
#b=Symbol('b')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)
```

```
24.0*pi
```


Area of the region R in the polar form is $\int_R\int rdrd\theta$

Example 5:

Find the area of the cardioid $r = a(1+cos\theta)$ by double integration

```
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
#a=4

w3=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
pprint(w3)
```

6.4 Volume of a solid is given by $\int\limits_V \int \int dxdydz$

Example 6:

Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
w2=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(w2)
```

```
a*b*c/6
```

6.5 Center of Gravity

Find the center of gravity of cardioid . Plot the graph of cardioid and mark the center of gravity.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection = 'polar')
a=5


rad = np.arange(0, (2 * np.pi), 0.01)

# plotting the cardioid
for i in rad:
    r = a + (a*np.cos(i))
    plt.polar(i,r,'g.')

plt.polar(0,I,'r.')
plt.show()
```

6.6 Exercise:

1. Evaluate $\int_0^1 \int_0^x (x+y)dydx$
   Ans: 0.5

2. Find the $\int_0^{log(2)} \int_0^x \int_0^{x+log(y)} (e^{x+y+z})dzdydx$
   Ans: -0.2627

3. Find the area of positive quadrant of the circle $x^2 + y^2 = 16$
   Ans: 4π

4. Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{2} + \frac{y}{3} + \frac{z}{4} = 1$
   Ans: 4

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

7.1 Objectives:

Use python

1. to find partial derivatives of functions of several variables.
2. to find Jacobian of fuction of two and three variables.

Syntax for the commands used:

1. gamma

   ```
   math.gamma(x)
   ```

   Parameters :

   x : The number whose gamma value needs to be computed.

2. beta

   ```
   math.beta(x,y)
   ```

   Parameters :

   x ,y: The numbers whose beta value needs to be computed.

3. **Note:** We can evaluate improper integral involving infinity by using `inf`.

Example 1:

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

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

```
1
```

Gamma function is $x(n) = \int_0^\infty e^{-x}x^{n-1}dx$

Example 2:

Evaluate $\Gamma(5)$ by using definition

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(w1))
```

```
24
```

Example 3:

Evaluate $\int_0^\infty e^{-st}\cos(4t)dt$. That is Laplace transform of $\cos(4t)$

```
from sympy import *
t,s=symbols('t,s')
# for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(simplify(w1))
```

Example 4:

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

```
#beta and gamma functions
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)
```

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


Example 5:

Calculate Beta(5/2,7/2) and Gamma(5/2).

```
#beta and gamma functions
# If the number is a fraction give it in decimals. Eg 5/2=2.5
from sympy import beta, gamma
m=float(input('m : '));
n=float(input('n :'));

s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is %3.3f '%s)
```

```
m : 2.5
n :3.5
gamma ( 3.5 ) is 3.323
Beta ( 2.5 3.5 ) is 0.037
```

Example 6:

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

```
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')
```

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

7.2 Exercise:

1. Evaluate $\int_0^\infty e^{-t}cos(2t)dt$
   Ans: 1/5

2. Find the value of Beta(5/2,9/2)
   Ans: 0.0214

3. Find the value of Gamma(13)
   Ans: 479001600

4. Verify that $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$ for m=7/2 and n=11/2
   Ans: True

Mechanical & Civil Engineering Stream

LAB 6: Solution of second order ordinary differential equation and plotting the solution curve

6.1 Objectives:

Use python

1. to solve second order differential equations.

2. to plot the solution curve of differential equations.

A second order differential equation is defined as
$\frac{d^2y}{dx^2} + P(x)\frac{dy}{dx} + Q(x)y = f(x)$, where $P(x)$, $Q(x)$ and $f(x)$ are functions of x.
When $f(x) = 0$, the equation is called **homogenous** second order differential equation. Otherwise, the second order differential equation is **non-homogenous**.

Example 1:

Solve: $y'' - 5y' + 6y = cos(4x)$.

```
# Import all the functions available in the SymPy library.
from sympy import *


#For the ease of representing the
x=Symbol('x')
y=Function("y")(x)
C1,C2=symbols('C1,C2')

y1=Derivative(y,x)
y2=Derivative(y1,x)

print("Differential Equation :\n")
diff1=Eq(y2-5*y1+6*y-cos(4*x),0)

display(diff1)

print("\n\nGeneral solution: \n")
z=dsolve(diff1)

display(z)

# Let c1=1, c2=2
PS=z.subs({C1:1,C2:2})
print("\n\n Particular Solution:\n")
display(PS)
```

Example 2:

Plot the solution curve (particular solution) of the above differential equation.

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

x1=np.linspace(0,2,1000)
y1=2*np.exp(3*x1+np.exp(2*x1)-np.sin(4*x1)/25-np.cos(4*x1)/50)

plt.plot(x1,y1)
plt.title("Solution curve")
plt.show()
```

Example 3:

Plot the solution curves of $y'' + 2y' + 2y = cos(2x), y(0) = 0, y'(0) = 0$

We can turn this into two first-order equations by defining a new depedent variable. For example,

$$z = y' \Rightarrow z' + 2z + 2y = cos(2x), z(0) = y(0) = 0.$$

$$y' = z; y(0) = 0$$

$$z' = cos(2x) - 2z - 2y; z(0) = 0.$$

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def dU_dx(U, x):
    # Here U is a vector such that y=U[0] and z=U[1]. This function
                                    should return [y', z']
    return [U[1], -2*U[1] - 2*U[0] + np.cos(2*x)]

U0 = [0, 0]
xs = np.linspace(0, 10, 200)
Us = odeint(dU_dx, U0, xs)

ys = Us[:,0]#  all the rows of the first column
ys1=Us[:,1]# all the rows of the second column

plt.xlabel("x")
plt.ylabel("y")
plt.title("Solution curves")
plt.plot(xs,ys,label='y');
plt.plot(xs,ys1,label='z');
plt.legend()
plt.show()
```

Example 4:

Solve: $3\frac{d^2x}{dt^2} + 2\frac{dx}{dt} - 2x = cos(2x)$ with $x(0) = 0; x'(0) = 0$ and plot the solution curve.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def f(u,x):
  return(u[1],-2*u[1]+2*u[0]+np.cos(2*x))

y0=[0,0]
xs=np.linspace(1,10,200)

us=odeint(f,y0,xs)
ys=us[:,0]

plt.plot(xs,ys,'r-')

plt.xlabel('t values')
plt.ylabel('x values')

plt.title('Solution curve')
plt.show()
```

6.2 Exercise:

1. An object weighs 2 kg stretches a spring 6 m. The spring is then released from the equilibrium position with an upward velocity of 16 m/sec. The motion of the object is denoted by $x'' + (8^2)x = 0$ where $\omega = 8$ is the angular frequency. Find $x(t)$ using initial conditions $x(0) = 0$ and $x'(0) = -16$ and plot the solution.

   Ans: $x(t) = -2\sin(8t)$

   Sketch of all solutions in this exercise: Note that $x(t) = c_1 \cos(8t) + c_2 \sin(8t)$, where $c_1 = x(0) = 0$ and $c_2 = x'(0) = -16$.

   Hint: Use `from scipy.integrate import odeint` and check the first column of the simulation result.

2. The mass of 16 kg stretches a spring by $\frac{8}{9}$ such that there is no damping and no external forces acting on the system. The spring is initially displaced 6 inches upwards from its equilibrium position and given an initial velocity of 1 ft/sec downward. Find the displacement at any time t, $u(t)$ denoted by the second order differential equation $\frac{1}{2}\frac{d^2}{dt^2}u(t) + 18u(t) = 0$ with initial conditions $u(0) = -\frac{1}{2}$ and $u'(0) = 1$ and plot the solution curve.

   Ans: $u(t) = -\frac{1}{2}\cos(6t) + \frac{1}{6}\sin(6t)$

   `https://tutorial.math.lamar.edu/classes/de/Vibrations.aspx`

3. The instantaneous position of the base of a stamping machine is given by the solutions of the second order differential equation $y'' + 100y' = \sin(10t)$. If the initial conditions are denoted by $y(0) = 0.005$ and $y'(0) = 0$, then find the position of the machine base and draw a plot for the solution.

Ans: $\frac{1}{200}\cos(10t) + \frac{1}{200}\sin(10t) + \frac{1}{20}\cos(10t)$

https://www.sjsu.edu/me/docs/hsu-Chapter%208%20Second%20order%20DEs_04-25-19.pdf

LAB 7: Solution of differential equation of oscillations of a spring with various load

7.1 Objectives:

Use python

1. to solve the differential equation of oscillation of a spring.
2. to plot the solution curves.

The motion of the spring mass system is given by the differential equation $m\frac{d^2x}{dt^2} + a\frac{dx}{dt} + kx = f(t)$ where, m is the mass of a spring coil,x is the displacement of the mass from its equillibrium position, a is damping constant, k is spring constant.

Case 1: Free and undamped motion - $a = 0, f(t) = 0$

Differential Equation : $m\frac{d^2x}{dt^2} + kx = 0$

Case 2: Free and damped motion: $f(t) = 0$

Differential Equation : $m\frac{d^2x}{dt^2} + a\frac{dx}{dt} + kx = 0$

Case 3: Forced and damped motion: Differential Equation : $m\frac{d^2x}{dt^2} + a\frac{dx}{dt} + kx = f(t)$

Example 1:

Solve $\frac{d^2x}{dt^2} + 64x = 0, x(0) = \frac{1}{4}, x'(0) = 1$ and plot the solution curve.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def f(u,x):
  return(u[1],-64*u[0])

y0=[1/4,1]
xs=np.linspace(0,5,50)

us=odeint(f,y0,xs)
ys=us[:,0]
print(ys)
plt.plot(xs,ys,'r-')

plt.xlabel('Time')
plt.ylabel('Amplitude')

plt.title('Solution of free and undamed case')
plt.grid(True)
plt.show()
```

Example 2:

Solve $9\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + 1.2x = 0, x(0) = 1.5, x'(0) = 2.5$ and plot the solution curve.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def f(u,x):
  return(u[1],-(1/9)*(1.2*u[1]+2*u[0]))

y0=[2.5,1.5]
xs=np.linspace(0,20*np.pi,2000)

us=odeint(f,y0,xs)
print(us)
ys=us[:,0]

plt.plot(xs,ys,'r-')

plt.xlabel('Time')
plt.ylabel('Amplitude')

plt.title('Solution of free and damped case')
plt.grid(True)
plt.show()
```


7.2 Exercise:

1. An object weighs 2 kg stretches a spring 6 m. The spring is then released from the equilibrium position with an upward velocity of 16 m/sec. The motion of the object is denoted by $x'' + (8^2)x = 0$ where $\omega = 8$ is the angular frequency. Find $x(t)$ using initial conditions $x(0) = 0$ and $x'(0) = -16$ and plot the solution.

   Ans: $x(t) = -2\sin(8t)$

   Sketch of all solutions in this exercise: Note that $x(t) = c_1\cos(8t) + c_2\sin(8t)$, where $c_1 = x(0) = 0$ and $c_2 = x'(0) = -16$.

   Hint: Use `from scipy.integrate import odeint` and check the first column of the simulation result.

2. The mass of 16 kg stretches a spring by $\frac{8}{9}$ such that there is no damping and no external forces acting on the system. The spring is initially displaced 6 inches upwards from its equilibrium position and given an initial velocity of 1 ft/sec downward. Find the displacement at any time t, $u(t)$ denoted by the second order differential equation $\dfrac{1}{2}\dfrac{d^2}{dt^2}u(t) + 18u(t) = 0$ with initial conditions $u(0) = -\dfrac{1}{2}$ and $u'(0) = 1$ and plot the solution curve.

   Ans: $u(t) = -\frac{1}{2}\cos(6t) + \frac{1}{6}\sin(6t)$

   `https://tutorial.math.lamar.edu/classes/de/Vibrations.aspx`