AJIET

AJ INSTITUTE OF ENGINEERING & TECHNOLOGY

A Unit of Laxmi Memorial Education Trust®

(Approved by AICTE, New Delhi, Affiliated to Visvesvaraya Technological University, Belgavi)

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAUM

MATHEMATICS – I FOR CIVIL ENGINEERING STREAM

(Subject Code: BMATC101)

MASTER COPY STUDENT MANUAL

for

FIRST YEAR B.E (CIVIL)

SEMESTER - I

Mrs. Vilma D'Souza

Assistant Professor, Department of Mathematics

Vision

To produce top-quality engineers who are groomed for attaining excellence in their profession and competitive enough to help in the growth of nation and global society.

Mission

- To offer affordable high-quality graduate program in engineering with value education and make the students socially responsible.
- To support and enhance the institutional environment to attain research excellence in both faculty and students and to inspire them to push the boundaries of knowledge base.
- To identify the common areas of interest amongst the individuals for the effective industry- institute partnership in a sustainable way by systematically working together.
- To promote the entrepreneurial attitude and inculcate innovative ideas among the engineering professionals.

Program Outcome

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis : Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural science and engineering sciences.

PO3: Design/development of solutions : Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations.

PO4: Conduct investigations of complex problems : Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment sustainability: Understand the impact of the professional engineering solutions in the societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics : Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Lifelong learning: recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broader context of technological change.

Mathematics-I for Civil Engineering stream

(Effective from the academic year 2022 -2023)

SEMESTER – I

Course Code	**BMATC101**	CIE Marks	**50**
CREDITS	**04**	SEE Marks	**50**
Teaching Hours/Week (L:T:P:S)	2:2:2:0	Exam Hours	**3 hours**

Course Type : Integrated

Total Hours of Pedagogy 40 hours Theory + 10 to12 Lab

Course Objectives:

The goal of the course Mathematics-I for Civil Engineering stream (22MATC11) is to

- **Familiarize** the importance of calculus associated with one variable and two variables for Civil engineering.
- **Analyze** Civil engineering problems applying Ordinary Differential Equations.
- **Develop** the knowledge of Linear Algebra referring to matrices.

Module-1: Calculus (8 hours)

Introduction to polar coordinates and curvature relating to Civil Engineering. Polar coordinates, Polar curves, angle between the radius vector and the tangent, angle between two curves. Pedal equations. Curvature and Radius of curvature - Cartesian, Parametric, Polar and Pedal forms. Problems. Self-study: Center and circle of curvature, evolutes and involutes. Applications: Structural design and paths, Strength of materials, Elasticity.

Module-2: Series Expansion and Multivariable Calculus (8 hours)

Introduction of series expansion and partial differentiation in Civil Engineering applications. Taylor's and Maclaurin's series expansion for one variable (Statement only) – problems. Indeterminate forms - L'Hospital's rule-Problems. Partial differentiation, total derivative -

differentiation of composite functions. Jacobian and problems. Maxima and minima for a function of two variables. Problems.

Module-3: Ordinary Differential Equations (ODEs) of First Order (8 hours)

Introduction to first-order ordinary differential equations pertaining to the applications for Civil Engineering. Linear and Bernoulli's differential equations. Exact and reducible to exact differential equations - Integrating factors on 1 N ($-\partial N\ \partial x$) and 1 M ($-\partial M\ \partial y$). Orthogonal trajectories, L-R & C-R circuits. Problems. Non-linear differential equations: Introduction to general and singular solutions, Solvable for p only, Clairaut's equations, reducible to Clairaut's equations. Problems.

Module-4: Ordinary Differential Equations of Higher Order (8 hours)

Importance of higher-order ordinary differential equations in Civil engineering applications. Higher-order linear ODEs with constant coefficients - Inverse differential operator, method of variation of parameters, Cauchy's and Legendre's homogeneous differential equations - Problems.
Self-Study: Formulation and solution of Cantilever beam. Finding the solution by the method of undetermined coefficients.

Module-5: Linear Algebra (8 hours)

Introduction of linear algebra related to Civil Engineering applications. Elementary row transformation of a matrix, Rank of a matrix. Consistency and Solution of system of linear equations - Gauss-elimination method, Gauss-Jordan method and approximate solution by Gauss-Seidel method. Eigenvalues and Eigenvectors, Rayleigh's power method to find the dominant Eigenvalue and Eigenvector.

Course outcomes :

At the end of the course the student will be able to :

CO1: Apply the knowledge of calculus to solve problems related to polar curves
CO2: Learn the notion of partial differentiation to compute rate of change of multivariate functions
CO3: Analyze the solution of linear and nonlinear ordinary differential equations
CO4 : Make use of matrix theory for solving the system of linear equations and compute eigenvalues and eigenvectors
CO5 : Familiarize with modern mathematical tool - PYTHON

Weblinks and Video Lectures (e-Resources):

1. http://nptel.ac.in/courses.php?disciplineID=111
2. http://www.class-central.com/subject/math(MOOCs)
3. http://academicearth.org/
4. VTU e-Shikshana Program
5. VTU EDUSAT Program

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50). The minimum passing mark for the SEE is 35% of the maximum marks (18 marks out of 50). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures not less than 35% (18 Marks out of 50) in the semesterend examination (SEE), and a minimum of 40% (40 marks out of 100) in the total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation(CIE):

The CIE marks for the theory component of the IC shall be 30 marks and for the laboratory component 20 Marks. CIE for the theory component of the IC

CIE for the theory component of the IC :

Three Tests each of 20 Marks; after the completion of the syllabus of 35-40%, 65-70%, and 90- 100% respectively.

- Two Assignments/two quizzes/ seminars/one field survey and report presentation/one-course project totalling 20 marks. Total Marks scored (test + assignments) out of 80 shall be scaled down to **30 marks.**

CIE for the practical component of the IC

- On completion of every experiment/program in the laboratory, the students shall be evaluated and marks shall be awarded on the same day. The 15 marks are for conducting the experiment and preparation of the laboratory record, the other 05 marks shall be for the test conducted at the end of the semester.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to 15 marks.
- The laboratory test (**duration 03 hours**) at the end of the 15th week of the semester/after completion of all the experiments (whichever is early) shall be conducted for 50 marks and scaled down to **05 marks**.

Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IC/IPCC for **20 marks**.

- The minimum marks to be secured in CIE to appear for SEE shall be 12 (40% of maximum marks) in the theory component and 08 (40% of maximum marks) in the practical component. The laboratory component of the IC/IPCC shall be for CIE only. However, in SEE, the questions from the laboratory component shall be included. The maximum of 05 questions is to be set from the practical component of IC/IPCC, the total marks of all questions should not be more than 25 marks. The theory component of the IC shall be for both CIE and SEE.

Semester End Examination (SEE):

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the subject (**duration 03 hours**)

- The question paper shall be set for 100 marks. The medium of the question paper shall be English/Kannada). The duration of SEE is 03 hours.
- The question paper will have 10 questions. Two questions per module. Each question is set for 20 marks. The students have to answer 5 full questions, selecting one full question from each module. The student has to answer for 100 marks and **marks scored out of 100 shall be proportionally reduced to 50 marks.**

- There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics under that module**.

Suggested Learning Resources:

Books (Title of the Book/Name of the author/Name of the publisher/Edition and Year)

Text Books :

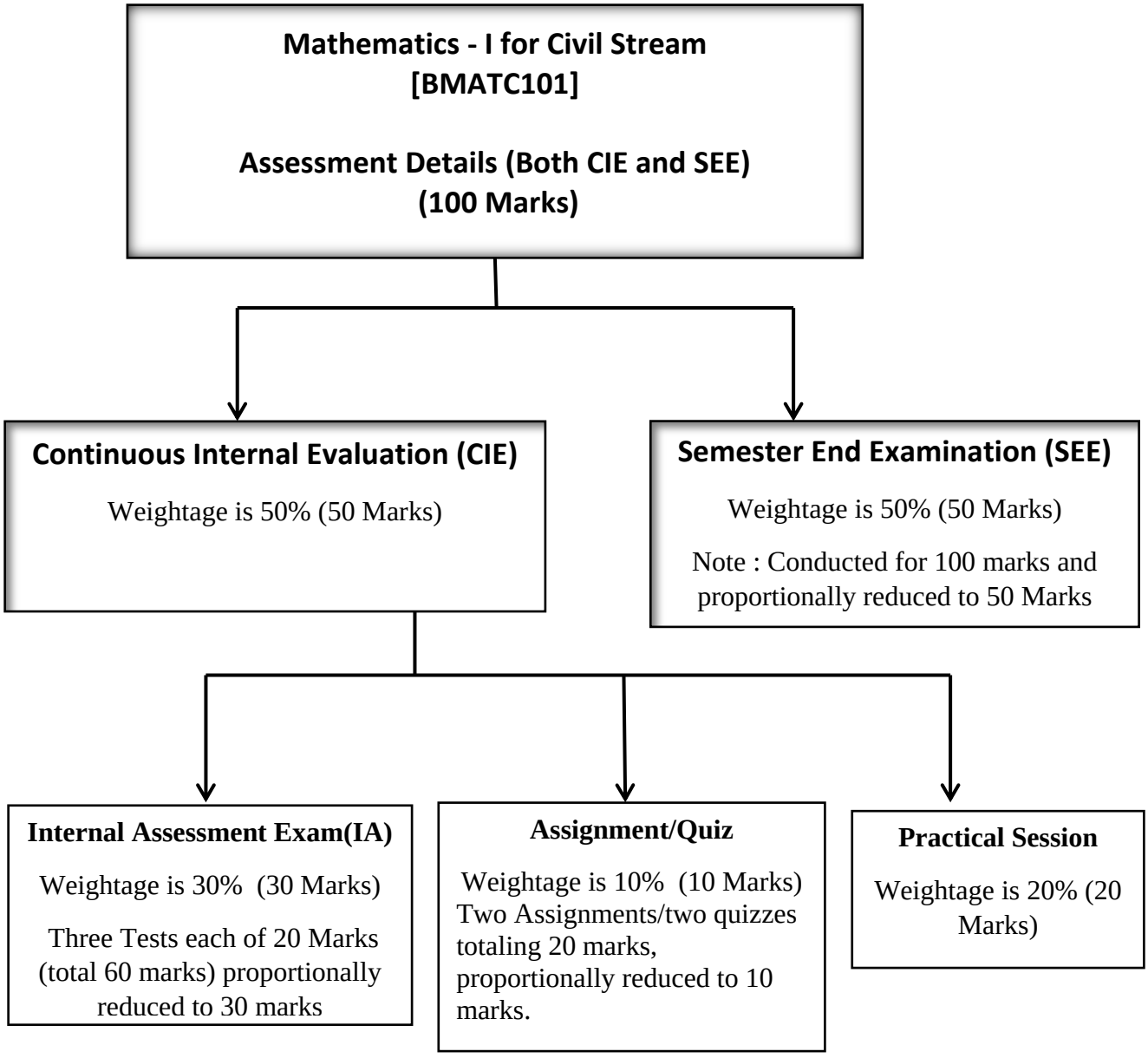
1. **B. S. Grewal**: "Higher Engineering Mathematics", Khanna Publishers, 44th Ed., 2021.
2. **E. Kreyszig**: "Advanced Engineering Mathematics", John Wiley & Sons, 10th Ed., 2018.
3. **David M Burton**: "Elementary Number Theory" Mc Graw Hill, 7th Ed.,2017.

Reference Books :

4. **V. Ramana**: "Higher Engineering Mathematics" McGraw-Hill Education, 11th Ed., 2017
5. **Srimanta Pal & Subodh C. Bhunia**: "Engineering Mathematics" Oxford University Press, 3 rd Ed., 2016.
6. **N.P Bali and Manish Goyal**: "A Textbook of Engineering Mathematics" Laxmi 26.10.2022 5 Publications, 10th Ed., 2022.
7. **C. Ray Wylie, Louis C. Barrett**: "Advanced Engineering Mathematics" McGraw – Hill Book Co., New York, 6th Ed., 2017.
8. **Gupta C.B, Sing S.R and Mukesh Kumar**: "Engineering Mathematic for Semester I and II", Mc-Graw Hill Education(India) Pvt. Ltd 2015.
9. **H. K. Dass and Er. Rajnish Verma**: "Higher Engineering Mathematics" S. Chand Publication, 3rd Ed., 2014.
10. **James Stewart**: "Calculus" Cengage Publications, 7th Ed., 2019.
11. **David C Lay**: "Linear Algebra and its Applications", Pearson Publishers, 4th Ed., 2018.
12. **Gareth Williams:** "Linear Algebra with Applications", Jones Bartlett Publishers Inc., 6th Ed., 2017.

13. **Gilbert Strang:** "Linear Algebra and its Applications", Cengage Publications, 4th Ed. 2022.
14. **William Stallings**: "Cryptography and Network Security" Pearson Prentice Hall, 6th Ed., 2013.
15. **Kenneth H Rosen**: "Discrete Mathematics and its Applications" McGraw-Hill, 8th Ed. 2019.
16. **Ajay Kumar Chaudhuri**: "Introduction to Number Theory" NCBA Publications, 2nd Ed., 2009.
17. **Thomas Koshy**: "Elementary Number Theory with Applications" Harcourt Academic Press, 2 nd Ed., 2008.

Assessment Process for Mathematics- I for Civil Engineering Stream Course



List of Laboratory experiments (2 hours/week per batch/ batch strength 15) 10 lab sessions + 1 repetition class + 1 Lab Assessment

1) 2D plots for Cartesian and polar curves
2) Finding angle between polar curves, curvature and radius of curvature of a given curve
3) Finding partial derivatives and Jacobian
4) Applications to Maxima and Minima of two variables
5) Solution of first-order ordinary differential equation and plotting the solution curves
6) Solutions of second order ordinary differential equations with initial/boundary conditions
7) Solution of differential equation of oscillations of a spring/ deflection of beam with different loads
8) Numerical solution of system of linear equations, test for consistency and graphical representation
9) Solution of system of linear equations using Gauss-Seidel iteration
10) Compute eigenvalues and eigenvectors and find the largest and smallest eigenvalue by Rayleigh power method.

Instructions and method of evaluation

1. In each Lab student have to show the record of previous Lab.
2. Each Lab will be evaluated for 15 marks and finally average will be taken for 15 marks.
3. Viva questions shall be asked in labs and attendance also can be considered for everyday Lab evaluation.
4. Tests shall be considered for 5 marks and final Lab assessment is for 20 marks.
5. Student has to score minimum 8 marks out of 20 to pass Lab component.

Introduction

Basics of Python

https://drive.google.com/file/d/1gVG2IJ8BIjhYDwDx6jWJns59h9dGOGVi/view?usp=share_link

Programming Structures

Conditional structure

What is conditioning in Python?

• Based on certain conditions, the flow of execution of the program is determined using proper syntax.

How to use if conditions?

- if statement — for implementing one-way branching
- if-else statements —for implementing two-way branching
- nested if statements —for implementing multiple branching
- if-elif ladder — for implementing multiple branching

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

Example: #Check if the given number is positive

```
a=int (input (" Enter an integer: "))
if a>0:
print (" Entered value is positive ")
```

```
# 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 .
```

LAB 1: 2D plots for Cartesian and polar curves

1.1 **Objectives**:

Use python
1. to plot Cartesian curves.
2. to plot polar curves.
3. to plot implicit functions.

Syntax for the commands used:
1. 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)

2. A scatter plot of y versus x with varying marker size and/or color.
scatter (x_axis_data , y_axis_data , s=None , c=None , marker =None , cmap=None , vmin=None , vmax =None ,alpha =None , linewidths =None ,edgecolors = None)

3. 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)

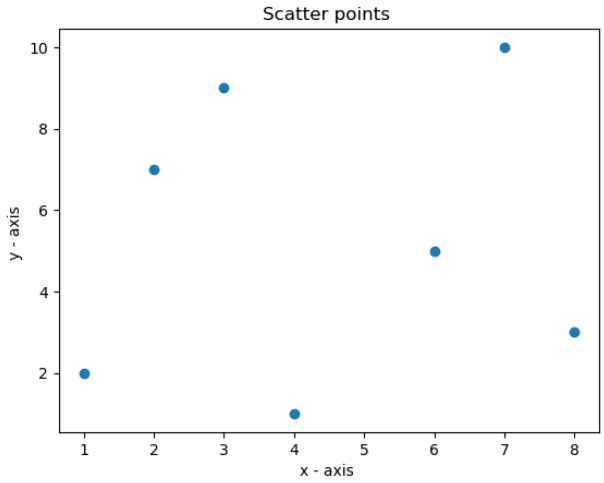
4. Return evenly spaced values within a given interval. arange can be called with a varying number of positional arguments.
numpy . arange ([start ,]stop , [step ,] dtype =None , *, like = None)
https://matplotlib.org/stable/api/pyplot_summary.html#modulematplotlib.pyplot

1.2 Example: Plotting points (Scattered plot)

```
#Syntax
# importing the required module
#x axis values
# corresponding y axis values
# plotting the points
# naming the x axis
# naming the y axis
# giving a title to my graph
# function to show the plot
```

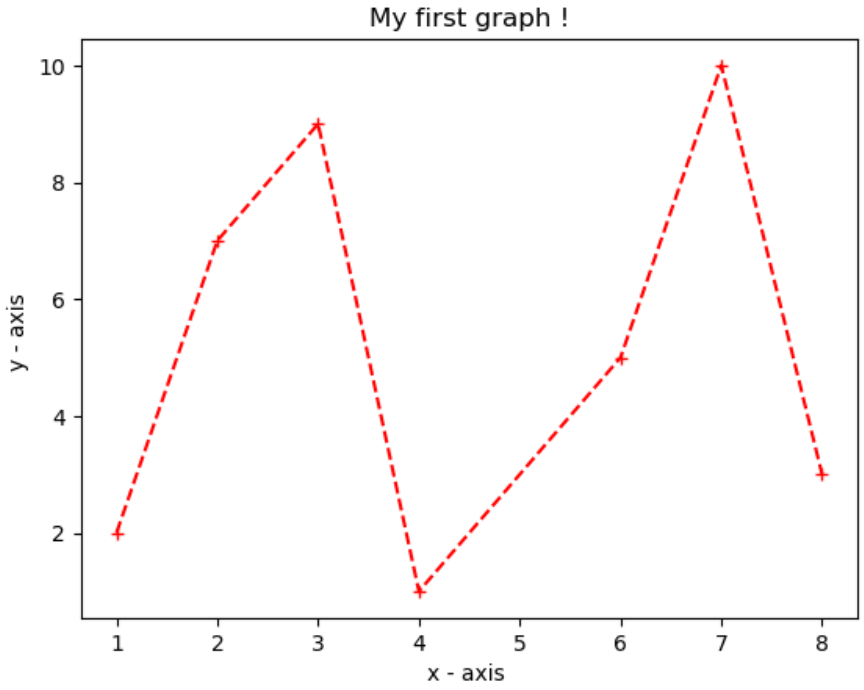
Example: Plotting points(Scattered plot)

```
# 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 () # function to show the plot
```



Example: Plotting a line(Line plot)

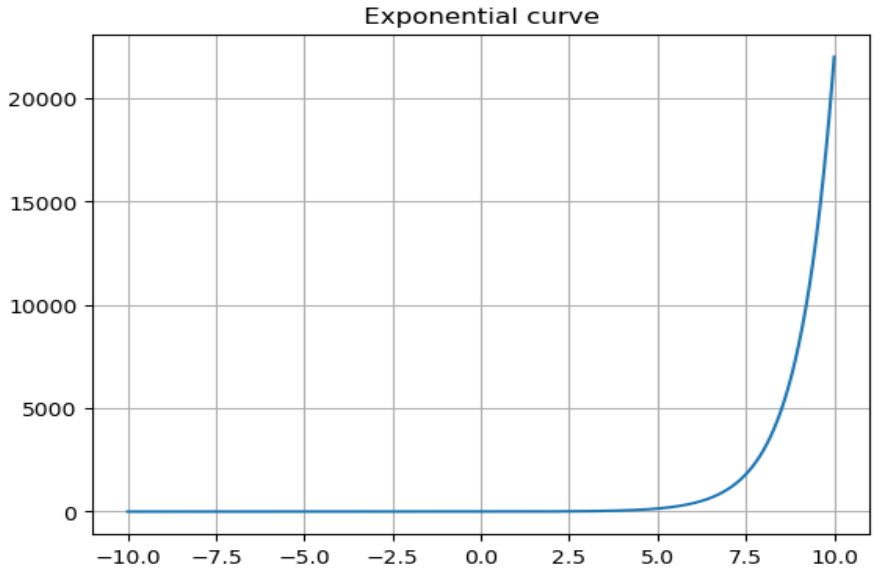
```
# importing the required module
import matplotlib . pyplot as plt
x = [1,2,3,4,6,7,8] # x axis values
y = [2,7,9,1,5,10 ,3] # corresponding y axis values
plt . plot (x, y, 'r+--') # plotting the points
plt . xlabel ('x - axis ') # naming the x axis
plt . ylabel ('y - axis ') # naming the y axis
plt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



Functions

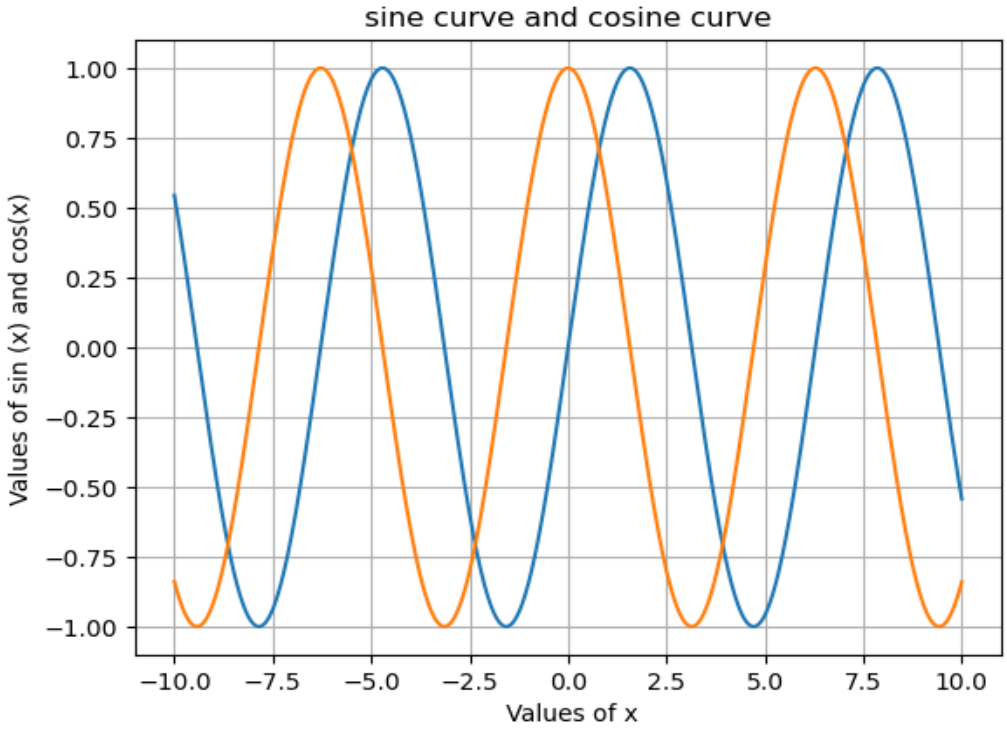
1. Exponential curve, $y = e^x$

```
# 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
```



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 ()
```



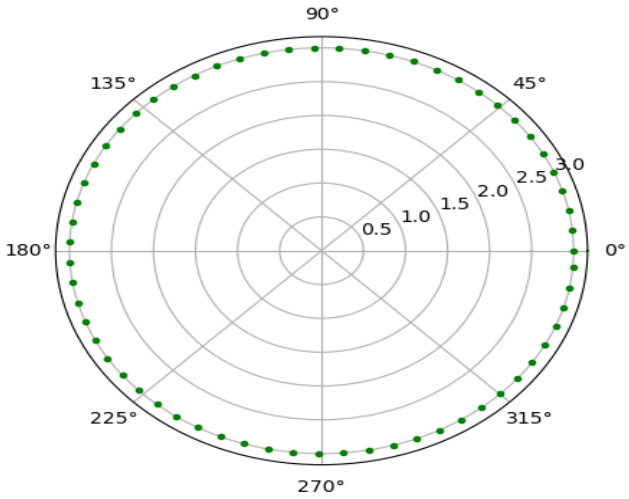
Polar Curves

The matplotlib.pyplot.polar() function in pyplot module of matplotlib python library is used to plot the curves in polar coordinates.

Syntax:
matplotlib . pyplot . polar (theta , r, ** kwargs)
• Theta: This is the angle at which we want to draw the curve.
• r: It is the distance.

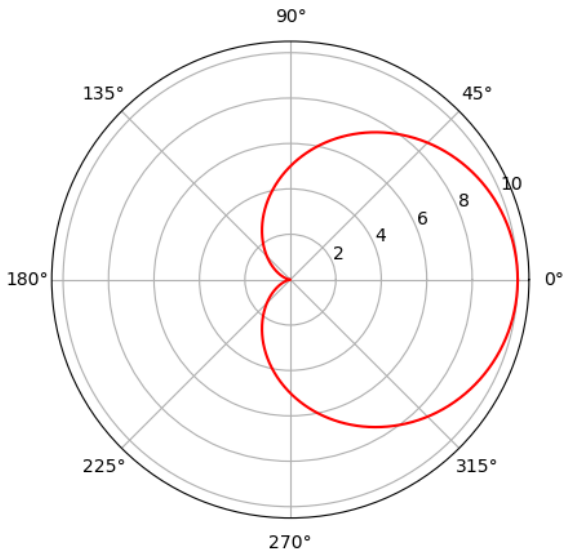
1)Plot a Circle: r = p, where p is the radius of the circle

```
import numpy as np
import matplotlib . pyplot as plt
plt . axes ( projection = 'polar ')
r = 3
rads = np. arange (0, (2 * np.pi), 0.1)
# plotting the circle
for i in rads :
 plt. polar (i, r, 'g.')
plt . show ()
```



3. Plot Cardioid: r = 5(1 + cosθ)

```
from pylab import *
theta = linspace (0,2*np.pi , 1000 )
r1=5+5*cos ( theta )
polar (theta ,r1 ,'r')
show ()
```



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

2.1 Objectives:

Use python
1. To find angle between two polar curves.
2. To find radius of curvature.

Syntax for the commands used:

1. diff()
diff (function , variable)
2. Derivative()
Derivative (expression , reference variable)
• expression – A SymPy expression whose unevaluated derivative is found.
• reference variable – Variable with respect to which derivative is found.
• Returns: Returns an unevaluated derivative of the given expression.
3. doit()
doit (x)
4. Return : evaluated object
5. simplify()
simplify (expression)
6. expression – It is the mathematical expression which needs to be simplified.
7. Returns: Returns a simplified mathematical expression corresponding to the input expression.
8. display()
display (expression)
9. expression – It is the mathematical expression which needs to be simplified.
10. Returns: Displays the expression.
11. syntax of Substitute : subs
()math_expression . subs (variable , substitute)
12. variable – It is the variable or expression which will be substituted.
13. substitute – It is the variable or expression or value which comes a substitute.
14. Returns: Returns the expression after the substitution.

2.2 1. Angle between two polar curves

Angle between radius vector and tangent is given by $\tan \phi = r \frac{d\theta}{dr}$

If $\tan \phi 1$ and $\tan \phi 2$ are angle between radius vector and tangent of two curves then $|\phi 1 - \phi 2|$ is the angle between two curves at the point of intersection.

```
#Syntax
# Define the variables required as symbols
# Input first polar curve
# Input first polar curve
# Find the derivative of first function
# Find the derivative of second function
# solve r1 ==r2, to find the point of intersection between curves substitute the value of "t" in t1
# Substitute the value of "t" in t2
# To find the inverse tan of w1
 # To find the inverse tan of w2
# Angle between two curves is abs (w1 -w2)
```

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
tanphi1=r1/dr1
tanphi2=r2/dr2
q= solve (r1-r2 ,t) # solve r1 ==r2 , to find the point of intersection
#between curves
w1=tanphi1. subs ({t: float (q[1])}) # substitute the value of "t" in t1
w2=tanphi2. subs ({t: float (q[1])}) # substitute the value of "t" in t2
phi1= atan (w1) # to find the inverse tan of w1
phi2= atan (w2) # to find the inverse tan of w2
w=abs(phi1-phi2) # angle between two curves is abs(w1 -w2)
print ('Angle between curves in radians is %0.3f '%(w))
```

Output: Angle between curves in radians is 1.571

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)
tanphi1=r1/dr1
tanphi2=r2/dr2
q= solve (r1-r2 ,t)
w1=tanphi1. subs ({t: float (q[0])})
w2=tanphi2. subs ({t: float (q[0])})
phi1= atan (w1)
phi2= atan (w2)
w=abs(phi1-phi2)
print ('Angle between curves in radians is %0.4f '% float (w))
```

Output: Angle between curves in radians is 1.5708

2.3 2. Radius of curvature

Formula to calculate Radius of curvature in polar form is $\rho = \frac{(r^2+r_1^2)^{\frac{3}{2}}}{r^2+2r_1^2-rr_2}$

```
#Syntax
# Define the variables required as symbols
# Input first polar curve
# Input first polar curve
# Find the first derivative of r w.r.t "t"
# Find the second derivative of r w.r.t "t"
# Substitute r1 and r2 in formula
# Substitute t in rho
```

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

```
from sympy import *
t= Symbol ('t') # define t as symbol
r= Symbol ('r')
r=4*(1+cos(t))
r1= diff (r,t) # find the first derivative of r w.r.t "t"
r2= diff (r1,t)# 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 )
```

Output: The radius of curvature is 3.7712 units

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= diff (r,t)
r2= diff (r1 ,t)
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 ))
```

Output: The radius of curvature is $\frac{(a^2)^{1.5}}{2a^2}$

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

3.1 Objectives:

Use python
1. to find partial derivatives of functions of several variables.
2. to find Jacobian of function of two and three variables.

Syntax for the commands used:

1. To create a matrix:
Matrix ([[row1],[row2],[row3] .... [rown]])

Ex: A 3 × 3 matrix can be defined as
Matrix ([[a11 ,a12 , a13],[a21 ,a22 ,a23],[a31 , a32 a33]])

2. Evaluate the determinant of a matrix M.
Determinant (M)
det (M)

3. To evaluates 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,

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

1) Prove that mixed partial derivatives, uxy = uyx 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)
ux = diff (u,x) # Differentate u w.r.t x
uy = diff (u,y) # Differentate u w.r.t. y
uxy = diff (ux ,y) # or duxy = diff (u,x,y)
uyx = diff (uy ,x) # or duyx = diff (u,y,x)
# Check the condition uxy=uyx
if uxy == uyx :
 print ('Mixed partial derivatives are equal ')
else :
 print ('Mixed partial derivatives are not equal ')
```

Output: Mixed partial derivatives are equal

2. Prove that if u = e^x(x cos(y) − y sin(y)) then uxx + uyy = 0.

```
from sympy import *
x,y = symbols ('x y')
u=exp(x)*(x*cos(y)-y*sin(y))
display (u)
ux = diff (u,x)
uy = diff (u,y)
uxx = diff (ux ,x) # or uxx= diff (u,x,x) second derivative of u w.r.t x
uyy = diff (uy ,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 :',w1)
```

Output: $(xcos(y) - ysin(y))e^x$

0

Jacobians

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

$$J = \frac{\partial(u,v)}{\partial(x,y)} = \begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix}$$

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
ux = diff (u,x)
uy = diff (u,y)
uz = diff (u,z)
vx = diff (v,x)
vy = diff (v,y)
vz = diff (v,z)
wx = diff (w,x)
wy = diff (w,y)
wz = diff (w,z)
# construct the Jacobian matrix
J= Matrix ([[ux ,uy ,uz],[vx ,vy ,vz],[wx ,wy ,wz]]);
print ("The Jacobian matrix is \n")
display (J)
# Find the determinat of Jacobian Matrix
Jac =det (J)
print ('\n\n J = ', Jac )
```

Output: The Jacobian matrix is

$$\begin{bmatrix} y/z & x/z & -xy/z^2 \\ -yz/z^2 & z/x & y/x \\ z/y & -xz/y^2 & x/y \end{bmatrix}$$

J = 4

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
ux = diff (u,x)
uy = diff (u,y)
uz = diff (u,z)
vx = diff (v,x)
vy = diff (v,y)
vz = diff (v,z)
wx = diff (w,x)
wy = diff (w,y)
wz = diff (w,z)
J= Matrix ([[ux ,uy ,uz],[vx ,vy ,vz],[wx ,wy ,wz]]);
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 = det(J1)
display ( Jac1 )
```

Output: The Jacobian matrix is

$$\begin{bmatrix} 1 & 6y & -3z^2 \\ 8xyz & 4x^2z & 4x^2y \\ -y & -x & 6z^2 \end{bmatrix}$$

J = $4x^3y - 24x^2y^3 + 12x^2yz^3 + 24x^2z^3 - 288xy^2z^3$

J at (1,-1,0):

LAB 4 : Applications to Maxima and Minima of two variables

4.1 Objectives:

Use python
to find find the maxima and minima of function of two variables.

Syntax for the commands used:

1. To solve
   sympy . solve (expression)
   Returns the solution to a mathematical expression/polynomial
2. To evaluate an expression
   sympy . evalf ()
   Returns the evaluated mathematical expression.
3. To construct an instant function
   sympy . lambdify (variable , expression , library)
   Converts a SymPy expression to an expression that can be numerically evaluated.
   lambdify acts like a lambda function, except it, converts the SymPy names to the names of the given numerical library, usually NumPy or math.

Maxima and minima problem

1. Find the Maxima and minima of $f(x,y) = x^3 + 3xy^2 - 15x^2 - 15y^2 + 72x$

```
import sympy
from sympy import *
x,y= symbols('x,y')
f=x**3+3*x*y**2-15*x**2-15*y**2+72*x
fx=diff(f, x)
fy=diff(f,y)
eq1,eq2 = Eq(fx,0), Eq(fy,0)
sol=solve([eq1, eq2], (x,y))
print ("The critical points are", sol)
fxx=diff(fx, x)
fxy= diff(fx,y)
fyy=diff(fy, y)
for i, j in sol:
    A= fxx.subs({x:i, y:j})
    B= fxy.subs({x:i, y:j})
    C= fyy.subs({x:i, y:j})
    delta= A*C -B*2
    if (delta>0 and A>0):
        print("The function has minimum at (",i,",",j,")")
        q1=f.subs({x:i, y:j})
        print("Minimum value:", q1)
    elif (delta>0 and A<0):
        print("The function has maximum at (",i,",",j,")")
        q2=f.subs({x:i, y:j})
        print("Minimum value:", q2)
    elif(delta<0):
        print("(",i,",",j,") is a saddle point")
    elif(delta==0):
        print("it needs further investigations")
```

Output

The critical points are [(4, 0), (5, -1), (5, 1), (6, 0)]

The function has maximum at (4 , 0)

Minimum value: 112

(5 , -1) is a saddle point

(5 , 1) is a saddle point

The function has minimum at (6 , 0)

Minimum value: 108

2. Find the Maxima and minima of $f(x, y) = x^3 + y^3 - 3x - 12y + 20$

```
import sympy
from sympy import *
x,y= symbols('x,y')
f=x**3+y**3-3*x-12*y+20
fx=diff(f, x)
fy=diff(f,y)
eq1,eq2 = Eq(fx,0), Eq(fy,0)
sol=solve([eq1, eq2], (x,y))
print ("The critical points are", sol)
fxx=diff(fx, x)
fxy= diff(fx,y)
fyy=diff(fy, y)
for i, j in sol:
    A= fxx.subs({x:i, y:j})
    B= fxy.subs({x:i, y:j})
    C= fyy.subs({x:i, y:j})
    delta= A*C -B*2
    if (delta>0 and A>0):
        print("The function has minimum at (",i,",",j,")")
        q1=f.subs({x:i, y:j})
        print("Minimum value:", q1)
    elif (delta>0 and A<0):
        print("The function has maximum at (",i,",",j,")")
        q2=f.subs({x:i, y:j})
        print("Minimum value:", q2)
    elif(delta<0):
        print("(",i,",",j,") is a saddle point")
    elif(delta==0):
        print("it needs further investigations")
```

Output

The critical points are [(-1, -2), (-1, 2), (1, -2), (1, 2)]

The function has maximum at (-1 , -2)

Minimum value: 38

(-1 , 2) is a saddle point

(1 , -2) is a saddle point

The function has minimum at (1 , 2)

Minimum value: 2

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.

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

```
# soln of diff. eqn \frac{dy}{dx}+ytanx=y^3 secx
from sympy import *
init_printing()
x,y=symbols('x,y')
y=Function('y')(x)
y1=diff(y,x)
print ("\n Differential Equation is ")
DE = Eq(y1 + y*tan(x) - y**3*sec(x), 0) # define the differential equation
display ( DE )
# General solution
print ("\n General Solution is ")
GS = dsolve ( DE ) # Solve the differential equation
display ( GS ) # Display the solution
```

Output

The differential equation is

$$-y^3(x)secx + y(x)tanx + \frac{d}{dx}y(x) = 0$$

General Solution is

$$y(x) = -\sqrt{\frac{1}{c_1 - 2sinx}}\ cosx,\ y(x) = \sqrt{\frac{1}{c_1 - 2sinx}}\ cosx$$

Solve: $xy\,(\frac{dy}{dx})^2 + x^2y^2\frac{dy}{dx} + xy = 0$

```
from sympy import *
init_printing()  # initialize the printing
x = Symbol('x')
x,y=symbols('x,y')
y=Function('y')(x)
y1=diff(y,x)
eq=Eq(x*y*y1**2-(x**2+y**2)*y1+x*y,0)
print ("\n Differential Equation is ")
display(eq)
sol=dsolve(eq)
print ("\n General Solution  is")
display(sol)
```

Output

The differential equation is

$$xy\,(\frac{dy}{dx})^2 + x^2y^2\frac{dy}{dx} + xy = 0$$

General Solution is

$$y(x) = -\sqrt{c_1 + x^2},\ y(x) = \sqrt{c_1 + x^2},\ y(x) = c_1x$$

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.

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

```
# Import all the functions available in the SymPy library .
from sympy import *
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\n General 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)
```

Output

Differential Equation :

$$6y(x) - \cos(4x) - 5\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

General solution :

$$y(x) = C_1 e^{2x} + C_2 e^{3x} - \frac{\sin(4x)}{25} - \frac{\cos(4x)}{50}$$

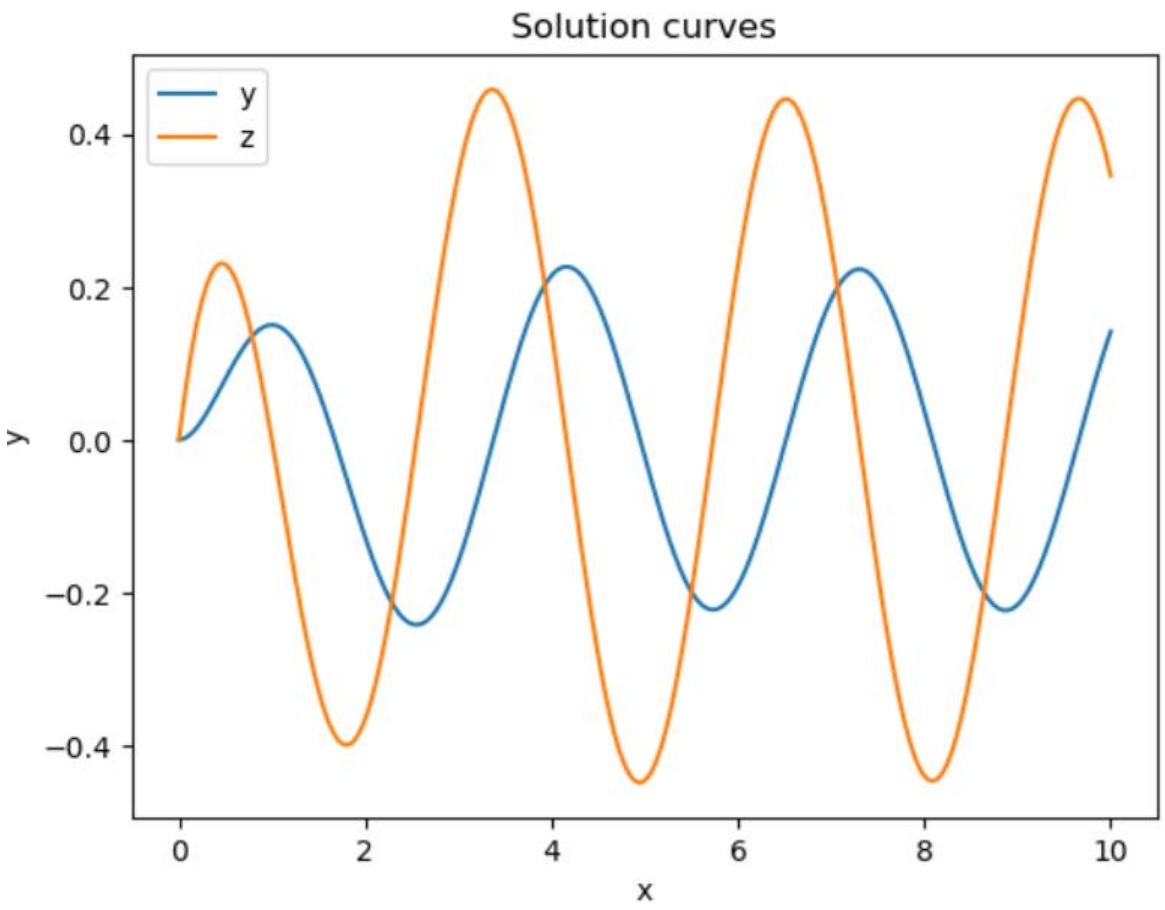
Particular Solution :

$$y(x) = 2e^{3x} + e^{2x} - \frac{\sin(4x)}{25} - \frac{\cos(4x)}{50}$$

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

```
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 ()
```

Output



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:

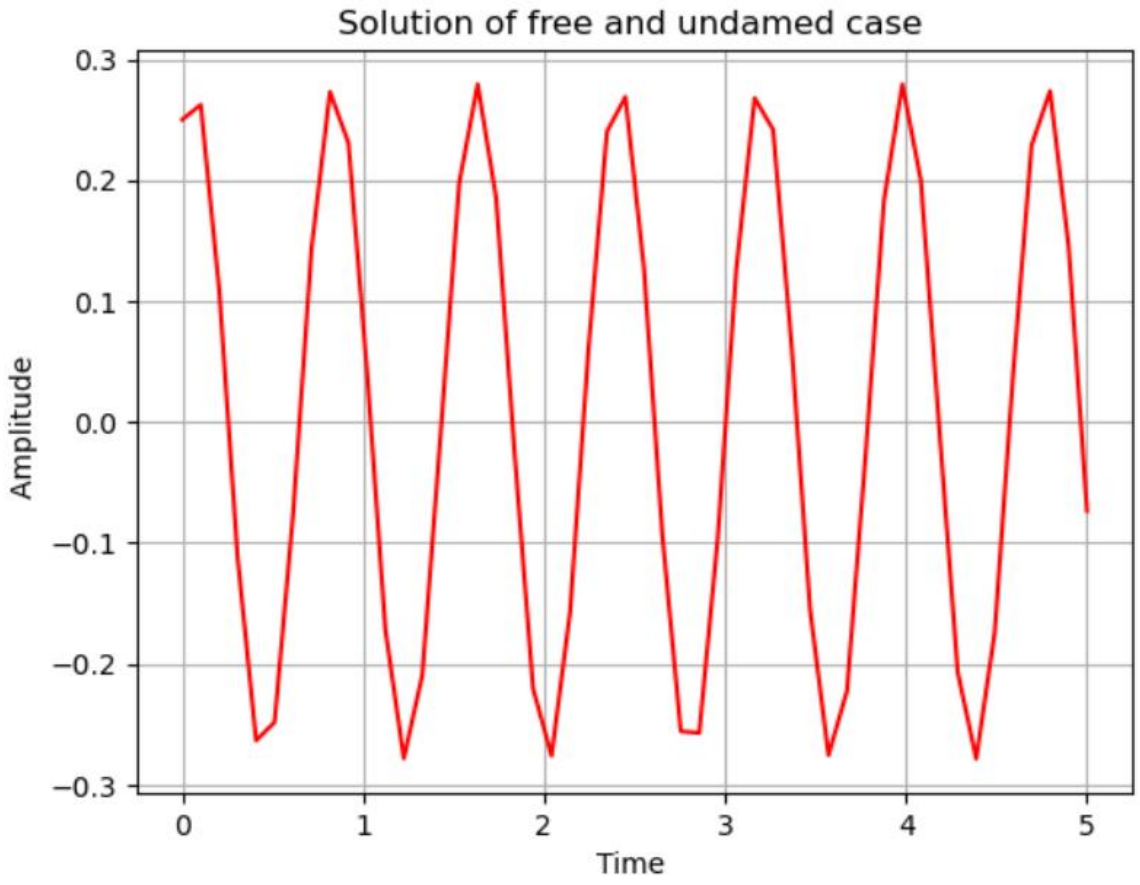
Differential Equation : $m\frac{d^2x}{dt^2} + a\ \frac{dx}{dt} + kx = f(t)$

1) Solve: $\frac{d^2x}{dt^2} + 64x = 0$, x(0)=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 ()
```

Output

```
[ 0.25        0.26230496  0.10930661 -0.11257621 -0.26351406 -0.24838663
 -0.07672717  0.14328539  0.27300019  0.23067161  0.0429749  -0.17180437
 -0.27831338 -0.20943065 -0.00856575  0.19769725  0.2793724   0.18498844
 -0.02597433 -0.22056822 -0.2761611  -0.15771861  0.06011739  0.2400677
  0.26872852  0.12803795 -0.09334152 -0.25589763 -0.2571883  -0.09640015
  0.1251389   0.26781603  0.24171681  0.06328883 -0.15502346 -0.27564072
 -0.22255055 -0.0292101   0.18253839  0.2792521   0.19998249 -0.0053151
 -0.20726315 -0.27859497 -0.1743576   0.03975907  0.22881978  0.27367938
  0.14606758 -0.0735953 ]
```

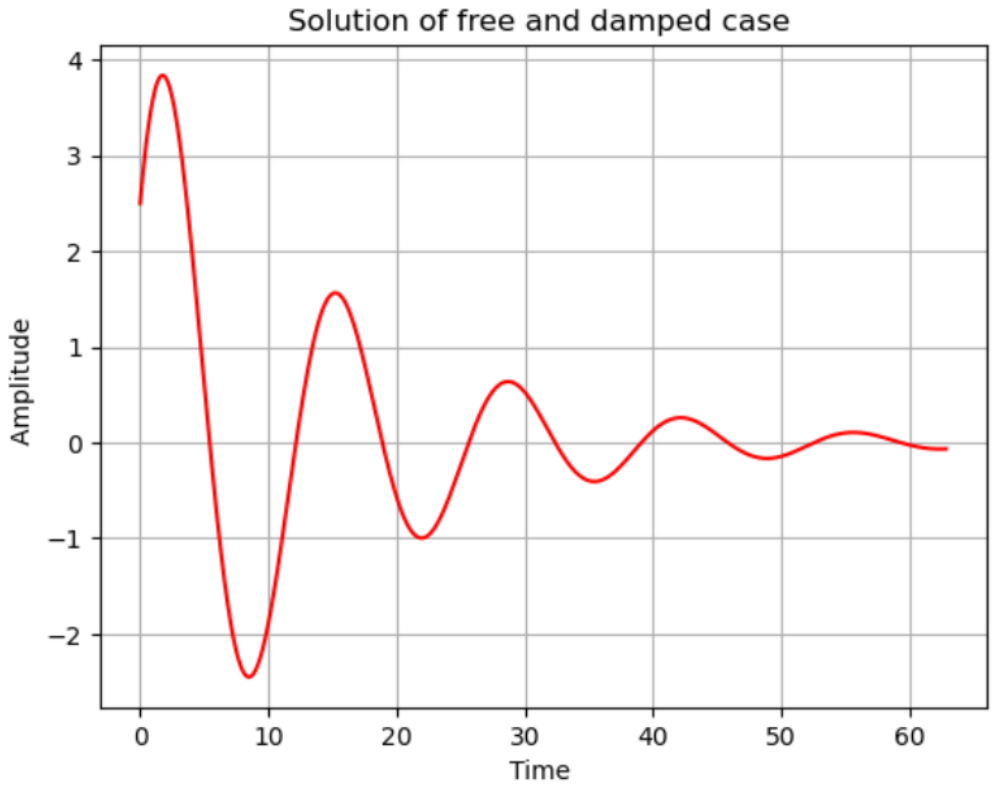


2) Solve: $9\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + 1.2\,x = 0$, x(0)=1.5, x'(0)=2.5 and plot the solution curve.

```
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 ()
```

Output

```
[[ 2.5         1.5       ]
 [ 2.546773    1.47613782]
 [ 2.59279255  1.45205195]
 ...
 [-0.06627678  0.00620788]
 [-0.06607481  0.00664319]
 [-0.0658592   0.00707522]]
```



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.

1) Check whether the following system of homogenous linear equation has non-trivial solution.

x1 + 2x2 − x3 = 0, 2x1 + x2 + 4x3 = 0, 3x1 + 3x2 + 4x3 = 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)")
```

Output

System has trivial solution

2) Check whether the following system of homogenous linear equation has non-trivial solution.

x1 + 2x2 − x3 = 0, 2x1 + x2 + 4x3 = 0, x1 − x2 + 5x3 = 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)")
```

Output

System has 1 non - trivial solution (s)

3) Examine the consistency of the following system of equations and solve if consistent.

x1 + 2x2 − x3 = 1, 2x1 + x2 + 4x3 = 2, 3x1 + 3x2 + 4x3 = 1.

```
import numpy as np
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 ")
```

Output

The system has unique solution

```
[[ 7.]
 [-4.]
 [-2.]]
```

4) Examine the consistency of the following system of equations and solve if consistent.

x1 + 2x2 − x3 = 1, 2x1 + x2 + 5x3 = 2, 3x1 + 3x2 + 4x3 = 1.

```
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

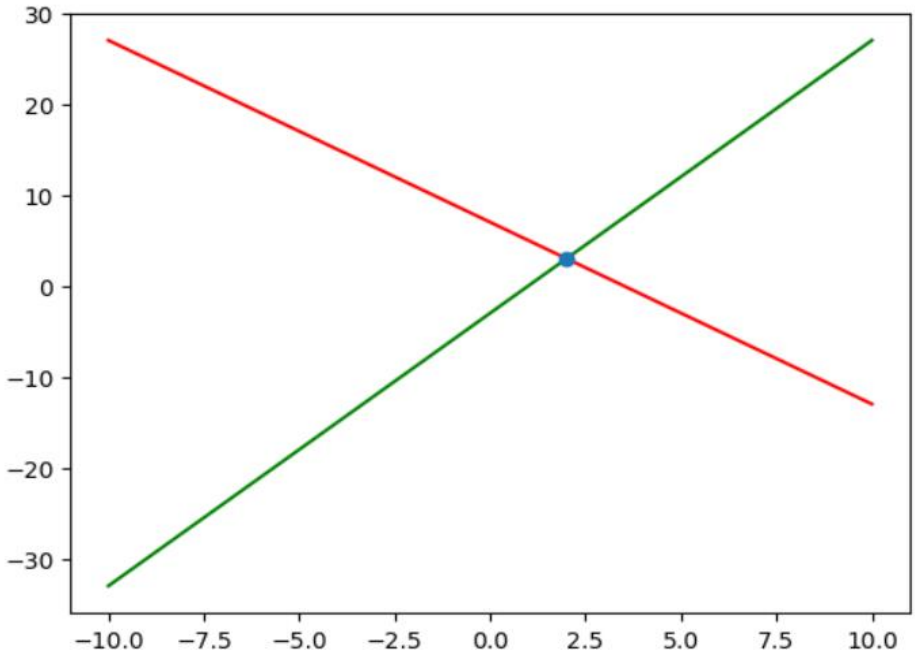
The system of equations is inconsistent

5) 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')
```

Output

point of intersection is A(2 , 3)



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.

Syntax for the commands used:

1. 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) = x2 −3x+1)

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

```
# Defining equations to be solved in diagonally dominant form
from sympy import *
x, y, z = symbols('x, y,z')
f1 = (17-y+2*z)/20
f2 = (-18-3*x+z)/20
f3 = (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 ('Count \tx\ty\tz\n')
condition = True
while condition :
    x1 = f1.subs({x:x0 ,y:y0 ,z:z0})
    y1 = f2.subs({x:x1 ,y:y0 ,z:z0})
    z1 = f3.subs({x:x1 ,y:y1 ,z: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 ('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
```

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

```
from sympy import *
x, y, z = symbols('x, y,z')
f1 = (1+y-2*z)/3
f2 = (3-x+z)/2
f3 = (2-2*x+2*y)/6
# 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 ('Count \tx\ty\tz\n')
condition = True
while condition :
    x1 = f1.subs({x:x0 ,y:y0 ,z:z0})
    y1 = f2.subs({x:x1 ,y:y0 ,z:z0})
    z1 = f3.subs({x:x1 ,y:y1 ,z: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 ('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       0.3333  1.3333  0.6667

Solution : x=0.333 , y=1.333 and z = 0.667

2       0.3333  1.6667  0.7778

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

LAB 10: Compute eigenvalues and eigenvectors and find the largest and smallest eigenvalue by the Rayleigh 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) = x2 −3x+1)

5. 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.

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

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

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 :  8.982056720677651

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

2) 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
X = np. array ([1, 1,1])
A = np. array ([[6,-2,2 ], [-2,3,-1],[2,-1,3]])
for i in range (15):
    X = np.dot(A, X)
    eigenvalue=max(abs(X))
    X = X/eigenvalue
    oldeigenvalue = eigenvalue
print("\n Given matrix is : \n", A)
print ("Numerically largest eigenvalue is %0.3f" %(eigenvalue))
print ("Corresponding Eigen vector is ", X)
```

Output

Given matrix is :

[[6 -2 2]

[-2 3 -1]

[2 -1 3]]

Numerically largest eigenvalue is 8.000

Corresponding Eigen vector is [1. -0.5 0.5]