CPL104

S16036

1. Vectors

I.)

a.)

Vectors	Scalers
• Has both magnitude and direction	• Has only magnitude
• Two or three dimentional	• One dimentional
e.g. : displacement, velocity	e.g. : distance, speed

b.) Taking the square root of the sum of the squares of the vectors of the individual components

c.) If the two vectors are a and b,

$$dot\ product = \vec{a}.\vec{b} = |a|.|b|.\cos\theta$$

d.) By using a matrix and solving it
If the two vectors are a and b,

$$cross\ product = |a| \times |b|.\sin\theta$$

e.) If the two vectors are a and b,

$$\theta = \cos^{-1}(\frac{\vec{a}.\vec{b}}{|a|.|b|})$$

f.) A unit vector means the vector which is the direction same as the original vector and has a magnitude of 1. We can find the unit vector by dividing the vector by its magnitude.
If the vector is a,

$$Unit\ vector = \frac{\vec{a}}{|a|}$$

II.)

a.)

By using analytical method:

$$E = \frac{1}{4\pi\varepsilon}\frac{Q}{r^2}$$

x direction $= \frac{1}{4\pi\varepsilon}\left[\frac{12}{5\sqrt{5}} - \frac{10}{5}\right] = -8.3401 \times 10^9$

y direction $= \frac{1}{4\pi\varepsilon}\left[\frac{12}{\sqrt{5}} - \frac{4}{4}\right] = 32.2990 \times 10^9$

Total electric field $= 3.3358 \times 10^{10}$

```python
#question1
#part II
#(a)
import numpy as np
from numpy.linalg import norm
import math

charges = np.array([2e-6, -4e-6, 6e-6])  # charges
dielec = 9e9  # 1/4(pi)epsilon

def magnitude_to_vector(magnitude, angle_degrees):
    # Convert angle from degrees to radians
    angle_radians = math.radians(angle_degrees)

    # Calculate the vector components
    component_x = magnitude * np.cos(angle_radians)
    component_y = magnitude * np.sin(angle_radians)

    # Return the vector as a tuple (x, y)
    return (component_x, component_y)

# Defining the vectors
vector1 = np.array([1, 0])
vector2 = np.array([0, 2])
vector3 = np.array([-2, -1])
```

```
    # Calculating the magnitudes of distances
    magnitude1 = np.linalg.norm(vector1)
    magnitude2 = np.linalg.norm(vector2)
    magnitude3 = np.linalg.norm(vector3)

    # Calculating electric fields by each charge on the origin
    EF1 = dielec * (2e-6) / (magnitude1)**2
    EF2 = dielec * (-4e-6) / (magnitude2)**2
    EF3 = dielec * (6e-6) / (magnitude3)**2

    print("Electric field on charge 1:", EF1)
    print("Electric field on charge 2:", EF2)
    print("Electric field on charge 3:", EF3)

    # Electric fields in the positive x direction
    EF_positiveX = (EF1 * -1) + (EF3 * (2 / np.sqrt(5)))

    # Electric fields in the positive y direction
    EF_positiveY = EF2 + (EF3 * (1 / np.sqrt(5)))

    # Total electric field as a magnitude
    totalEfield = np.sqrt(np.square(EF_positiveX) + np.square(EF_positiveY))

    print("Magnitude of total electric field:", totalEfield)

    angle_radians = np.arctan2(EF_positiveY, EF_positiveX)  # obtaining the tan-1 angle in radians
    angle_degrees = np.degrees(angle_radians)  # obtaining the angle in degrees

angle_radians = np.arctan2(EF_positiveY, EF_positiveX)  # obtaining the tan-1 angle in radians
angle_degrees = np.degrees(angle_radians)  # obtaining the angle in degrees

print("Direction of the total electric field:", angle_degrees, "degrees")  # displaying the angle in degrees

vector = magnitude_to_vector(totalEfield, angle_degrees)
print("The electric field as a vector:",vector)

unitVector = vector/totalEfield
print("Unit vector:",unitVector)
```

```
Electric field on charge 1: 18000.0
Electric field on charge 2: -9000.0
Electric field on charge 3: 10799.999999999998
Magnitude of total electric field: 9324.61179749811
Direction of the total electric field: -153.434948822922 degrees
The electric field as a vector: (-8340.18633720091, -4170.093168600457)
Unit vector: [-0.89442719 -0.4472136 ]
```

b.)

```
#question1
#part II
#(b)

import numpy as np
from numpy.linalg import norm

# Creating matrices
inertia = np.array([[3, 0, 0], [0, 5, 0], [0, 0, 4]])
angVelocity = np.array([[0], [0], [5]])

# Calculate the angular momentum for each pair of vectors
angMoment = np.dot(inertia, angVelocity)
magAngMoment= np.linalg.norm(angMoment)
print("Value of angular momentum:",magAngMoment)
```

```
Value of angular momentum: 20.0
```

```
#question1
#part II
#(c)
charge = 2e-6
velocity = np.array([3,4,0]) #representing the values as vectors
mField = np.array([0,0.2,0.3])

mForce = charge * np.cross(velocity, mField)     #magnetic force as a vector
magnitudeMForce = np.linalg.norm(mForce)         #magnitude of magnetic force

print("Magnetic force:",mForce)
print("Magnitude of the magnetic force:",magnitudeMForce)
```

```
Magnetic force: [ 2.4e-06 -1.8e-06  1.2e-06]
Magnitude of the magnetic force: 3.231098884280702e-06
```

2. Matrices

I.

a.)

Matrix addition properties:

- Commutative property: A+B=B+A
- Associative property: (A+B)+C=A+(B+C)
- Closure property: Sum of two matrices is always a matrix of the same size as the originals
- Additive identity property: When a zero matrix is added to any matrix, the result is the initial matrix
- Additive inverse property: When a matrix is added to the initial matrix, it results in the zero matrix

Matrix multiplication properties:

- Noncommutative property: $AB \neq BA$
- Associative property: (AB)C=A(BC)
- Multiplicative identity property: $[A]_{n\times n} \ln[I] = A$
- Dimension property: Number of columns in the first matrix is equal to the number of rows in the second matrix
- Distributive property: A(B+C)=AB+AC

b.) Transpose means flipping a matrix over its diagonal. (swapping rows and columns)
If the order of the original matrix is $n \times m$, then the order of the transporse matrix will be $m \times n$.

c.) Inverse of square matrix means a square matrix that when multiplied by the original matrix, results in the identity matrix. It is denoted by A^{-1}.
Matrix must be a square matrix.
Determinant of the matrix must not be zero

d.) A square matrix is a matrix which has an equal number of rows and columns. In a square matrix, the number of rows and columns are equal but in a rectangular matrix number of rows and columns are not equal.

e.) Diagonal matrix is a square matrix in which every element except the principal diagonal element is zero.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

f.) Identity matrix is a matrix of order $n \times n$ such that each main diagonal element is equal to 1, and the remaining elements in the matrix are equal to 0.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Properties of identity matrices:

- It is always a square matrix.
- When an identity matrix is multiplied by any matrix, it gives the matrix itself.
- Identity matrix resulted when two inverse matrices are multiplied.

Uses of identity matrices:

- To solve linear systems of equations
- To find the inverse of a matrix
- To calculate eigenvectors and eigenvalues
- To represent linear transformations

g.) The rank of a matrix means the maximum number of linearly independent rows or columns in the matrix. Matrix of zero matrix is zero.

- If the number of rows is lesser than the number of columns in the matrix, then the vectors of the matrix are always linearly dependent.
- If the number of rows in the matrix is greater than or equal to the number of columns in the matrix, then the vectors of the matrix are linearly independent only if elementary row operations on matrix can convert it not a matrix containing only mutually orthogonal identity matrices.

h.)

- Elementary row operations: When row operations such as row interchanges, multiplying a row by a non-zero constant, then the rank doesn't change.
- Matrix multiplication: rank changes if we multiply a matrix by a full rank matrix or the product of two full rank square matrices
- Transformation into row echelon form: the rank of a matrix is equal to the number of non-zero rows in the echelon form

II.

```
import numpy as np
import numpy.linalg
import math

matrixA = [[1,6,7],[5,9,-2],[3,4,0]]
matrixB = [[-2,3,1],[8,-2,3],[6,-3,-6]]

print("Matrix A:\n",matrixA)
print("Matrix B:\n",matrixB)
print()

print("A+B:\n",np.add(matrixA,matrixB))
print()

print("Determinant of matrix A:",np.linalg.det(matrixA))
print()
print("Transporse of matrix A:\n",np.transpose(matrixA))
print()

#checking weather matrix A is symmetric
if(np.array_equal(np.transpose(matrixA), matrixA)):
    print("Matrix A is symmetric")
else:
    print("Matrix A is not symmetric")
print()

print("Product of the two matrices:\n",numpy.dot(matrixA, matrixB))
```

```
Matrix A:
 [[1, 6, 7], [5, 9, -2], [3, 4, 0]]
Matrix B:
 [[-2, 3, 1], [8, -2, 3], [6, -3, -6]]

A+B:
 [[-1  9  8]
 [13  7  1]
 [ 9  1 -6]]

Determinant of matrix A: -77.00000000000001

Transporse of matrix A:
 [[ 1  5  3]
 [ 6  9  4]
 [ 7 -2  0]]

Matrix A is not symmetric

Product of the two matrices:
 [[ 88 -30 -23]
 [ 50   3  44]
 [ 26   1  15]]
```

III.

a.) $A = XX^T + YY^T$

$$A = \begin{pmatrix}\cos\theta \\ \sin\theta\end{pmatrix}(\cos\theta \quad \sin\theta) + \begin{pmatrix}\sin\theta \\ -\cos\theta\end{pmatrix}(\sin\theta \quad -\cos\theta)$$

$$A = \begin{pmatrix}1 & 0 \\ 0 & 1\end{pmatrix}$$

```python
#partIII
#a
import numpy as np
import numpy.linalg
import math

matrixX = [[math.cos(np.pi/6)],[math.sin(np.pi/6)]]
matrixY = [[math.sin(np.pi/6)],[-math.cos(np.pi/6)]]

A=numpy.dot(matrixX, np.transpose(matrixX))+numpy.dot(matrixY, np.transpose(matrixY))
print("matrix A:\n",A)
```

```
matrix A:
 [[1. 0.]
 [0. 1.]]
```

b.) $C = \begin{pmatrix} 2 & 1 & -3 \\ 1 & 2 & -3 \\ -3 & -3 & -3 \end{pmatrix}$

$$C^T = \begin{pmatrix} 2 & 1 & -3 \\ 1 & 2 & -3 \\ -3 & -3 & -3 \end{pmatrix}$$

$$C = C^T$$

Therefore, C is symmetric

```
#b
matrixC = [[2,1,-3],[1,2,-3],[-3,-3,-3]]
print("C:",matrixC)
print()

print("Transporse of matrix C:\n",np.transpose(matrixC))
print()

if(np.array_equal(np.transpose(matrixC), matrixC)):
    print("Matrix C is symmetric")
else:
    print("Matrix C is not symmetric")
print()
```

```
C: [[2, 1, -3], [1, 2, -3], [-3, -3, -3]]

Transporse of matrix C:
 [[ 2  1 -3]
 [ 1  2 -3]
 [-3 -3 -3]]

Matrix C is symmetric
```

c.) $U = e^{\frac{i\pi}{3}} \begin{pmatrix} e^{\frac{i\pi}{8}} & 0 \\ 0 & e^{\frac{i\pi}{8}} \end{pmatrix} \begin{pmatrix} \cos\frac{3\pi}{4} & -\sin\frac{3\pi}{4} \\ \sin\frac{3\pi}{4} & \cos\frac{3\pi}{4} \end{pmatrix} \begin{pmatrix} e^{\frac{-i\pi}{8}} & 0 \\ 0 & e^{\frac{-i\pi}{8}} \end{pmatrix}$

$$U = \begin{pmatrix} -0.68 - 0.18j & -0.68 - 0.18j \\ 0.35 + 0.61j & -0.35 - 0.61j \end{pmatrix}$$

```
# Defining the parameters
alpha = np.pi / 3
beta = np.pi / 4
delta = np.pi / 4
gamma = 3 * np.pi / 2

i = 1j    #complex number

expiAlpha = np.exp(i * alpha)

matrix1 = np.array([
    [np.exp(-i * beta / 2), 0],
    [0, np.exp(i * beta / 2)]
])
matrix2 = np.array([
    [np.cos(gamma / 2), -np.sin(gamma / 2)],
    [np.sin(gamma / 2), np.cos(gamma / 2)]
])
matrix3 = np.array([
    [np.exp(-i * delta / 2), 0],
    [0, np.exp(-i * delta / 2)]
])

matrixU = expiAlpha * np.dot(np.dot(matrix1, matrix2), matrix3)
print("Matrix U:\n",matrixU)
```

```
Matrix U:
 [[-0.6830127 -0.1830127j  -0.6830127 -0.1830127j ]
 [ 0.35355339+0.61237244j -0.35355339-0.61237244j]]
```

3. Eigenvalues and eigenvectors

I.

a.) Eigenvalue is a scaler that is used to transform the eigenvector.

$$AX = \boldsymbol{\lambda X}$$

$\boldsymbol{\lambda}$ is called an eigenvalue. When eigenvalue is negative, the direction of the transformation is also negative.

Eigenvectors are the non-zero vectors that do not change direction when any linear transformation is applied.

b.) Characteristic equation means an equation which is algebraic equation of degree n which depends on the solution of a given n^{th} differential equation. It can be obtained by equating to zero the characteristic polynomial of a matrix.

Eigenvalues are the roots of the characteristic polynomial. The equation $\det(M - xl) = 0$ is a polynomial equation in the variable x for given M. It is called the characteristic equation of the matrix M. You can solve it to find the eigenvalues x, of M.

c.) The vectors that a linear transformation works on are rotated, stretched, diagonalization or sheared.

- An eigenvector of a matrix A is a non-zero vector v such that the resultant vector, Av, is a scalar multiple of v when A is applied to v. To put it another way, under transformation A, v does not change direction; rather, it is only scaled by a certain amount.
- In terms of geometry, the factor that is used to scale the eigenvector v during the transformation is represented by the eigenvalue $\lambda\lambda$. The vector is stretched if $\lambda > 1$ $\lambda > 1$ and compressed if $0 < \lambda < 1$ $0<\lambda<1$. When $\lambda = 1$ $\lambda = 1$, the vector's length stays constant, but its direction can only vary in rotation or reflection scenarios. Conversely, when $\lambda = -1$ $\lambda = -1$, the vector's direction is inverted.

II.

```
#Question3
#partII
import numpy as np
from numpy.linalg import eig

def findEigen(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    return eigenvalues, eigenvectors

matrixA = np.array([[2, 2, 4], [1, 3, 5],[2, 3, 4]])

eigenvalues, eigenvectors = findEigen(matrixA)      #calling the function

print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
```

```
Eigenvalues: [ 8.80916362  0.92620912 -0.73537273]
Eigenvectors:
 [[-0.52799324 -0.77557092 -0.36272811]
 [-0.604391    0.62277013 -0.7103262 ]
 [-0.59660259 -0.10318482  0.60321224]]
```

III.

a.) M = 10kg

R = 2 x 10^{-2} m

Inertia = $\frac{2}{5}Mr^2$

$= \frac{2}{5} \times 10kg \times (2 \text{ x } 10^{-2})^2$

$= 1.6 \times 10^{-3}$ kgm^2

```
Moments of inertia in x direction: 0.0016 kg m^2
Moments of inertia in x direction: 0.0016 kg m^2
Moments of inertia in x direction: 0.0016 kg m^2

Eigenvalues: [0.0016 0.     0.    ]

Eigenvectors:
 [[ 1.         -0.70710678 -0.70710678]
 [ 0.          0.70710678  0.        ]
 [ 0.          0.          0.70710678]]
```

b.)

```
Hamiltonian matrix:
 [[1.41421356 2.12132034 2.82842712]
 [2.12132034 2.82842712 3.53553391]
 [2.82842712 3.53553391 4.24264069]]

Eigenvalues of the Hamiltonian matrix:
 [ 8.82521638e+00 -3.39935008e-01 -2.16226403e-16]

Eigenvectors of the Hamiltonian matrix:
 [[-0.43036222 -0.80506005  0.40824829]
 [-0.56654216 -0.11119045 -0.81649658]
 [-0.70272209  0.58267915  0.40824829]]
```

4. System of Linear Equations (SLE)

I.

a.) Each equation is written in standard form and the coefficients of the variables, and the constant of each equation becomes a row in the matrix. Each column then would be the coefficients of one of the variables in the system.

b.) If the determinant is zero, there exist infinitely many or no solutions to the given system. So, there is no unique solution for the matrix.
If the determinant is nonzero, the matrix is invertible and there exists an unique solution.

c.) First write the system as a matrix equation.

$$[coefficient\ matrix][variables] = [constants]$$

Find the inverse of the coefficient matrix.

Cancel the coefficient matrix by multiplying both sides by the inverse.

$$[variables] = [inverse][constants]$$

II.

```
#Question4
import numpy as np

#Equations

#2*I1-I5-I4=5
#I3-2*I1+2*I2=5
#2*I2-2*I6-I4=5
#I3-I5+2*I6=5
#2*I2+I3-I5-I4=10

matrixA = ([[2,0,0,-1,1,0],
            [-2,2,1,0,0,0],
            [0,0,2,-1,0,-2],
            [0,0,1,0,-1,2],
            [0,2,1,-1,-1,0],
            [0,0,0,-1,1,1]])
matrixB = ([5,5,5,5,10,0])

#solving the two matrices
x = np.linalg.solve(matrixA, matrixB)
print(x)
```

```
[ 2.85714286  3.57142857  3.57142857  0.71428571 -0.
```