### Why Linear Algebra for Data Science?

Data Science deals with extracting meaningful insights from the data. We need to represent data and perform numerous operations on them to extract insights. **Linear Algebra** helps in representation and operations on data in Data Science and Machine Learning.

Let’s understand the application of Linear algebra in Data Science with the following examples.

* **Deep Learning**  - Representing input to the model and model parameters as vector and matrices and making calculations using Linear Algebraic operations.
* **Image data** - Representing images as matrices and doing various geometrical transformations on them using Linear Algebraic operations.
* **Recommender system**  - using linear algebraic concepts to measure similarity.
* **Multiple linear regression**   - Solving multiple linear  equations using linear algebra
* **Feature extraction (PCA)** - using the linear algebraic concept of eigenvalue and eigenvectors
* **One hot encoding** - A matrix representation of the encoding

### What is Linear Algebra?

Linear Algebra is a branch of mathematics. It provides basic structures to represent data and various numerical methods and tools to solve problems.

The basic building blocks of Linear Algebra are scalar, vector, and matrix. A **Scalar** is a quantity, described by a numeric value. A**Vector** is an ordered collection of scalars. A **Matrix** is a collection of vectors.

2 [2,4,2] [2,4,2]   
 [1,2,2]  
 [3,3,1]

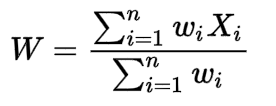
Let us explore an example to understand these building blocks.

Consider a student who has taken 3 exams for a given subject. The scores are 80%, 80%, and 85% respectively. Based on the difficulty level, the professor has allocated weights of 30%, 30%, and 40% to the exams respectively.

### Problem

Calculate the weighted mean of the marks obtained by the student.

The weighted mean 'W' is calculated by the formula:



where '**wi**' represent the weights and '**Xi**' represents the corresponding marks.

Here, individual score and weightage are considered as scalars. The collection of the scores and the corresponding weights can be represented as vectors as shown below:

x = [80, 80, 85]

w = [0.3, 0.3, 0.4]

In Linear Algebra, an operation is defined called ‘dot product’ which helps in multiplying two vectors. The dot product of the vectors w and X is:

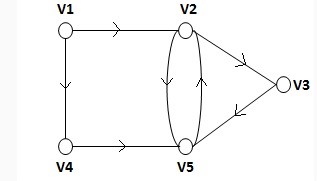
G:\Fifth Semester\Temp\2.png

This dot product will result in finding the weighted mean of the marks (which is a scalar).

Linear Algebraic operations are applied to the entire object (vector/matrix) instead of individual data points one at a time. This technique is called as **Vectorization** and it is more efficient while dealing with operations on large data.

Let us explore another example where Linear Algebra is used for data representation.

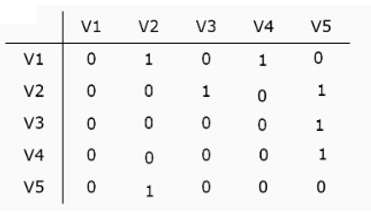
Consider a social networking site where five visitors are linked with each other as depicted in the graph below:



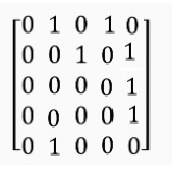
### Problem

How to use these relationships to extract more information about them?

These relationships can be converted into a relationship matrix in which '1' indicates **related** and '0' indicates **not related**as shown below:



From the above relationships, you can create a matrix for the directed graph as below:



This matrix can be used as a data structure for representing graphs in computer programs for further computation.

### Deep Learning

Over the past few decades, path-breaking innovations have transformed the way the world functions. Some of the notable innovations are speech recognition, self-driving cars, automated stores etc.

Deep Learning algorithms play a major role in these innovations. They are based on the concept of Artificial Neural Network (ANN) which involves linear algebraic data structures and operations.

### Image Processing

An important aspect of Artificial Intelligence is **computer vision**. Most computer vision tasks deal with images represented as an n-dimensional array of pixel values. Linear algebraic tools and techniques help us model and process these images to extract meaningful information.

### Recommender Systems

A recommendation system is a predictive modelling technique that provides personalized recommendations to customers. Mostly, it is used in e-commerce websites to recommend products, in media service websites to recommend movies or songs etc.

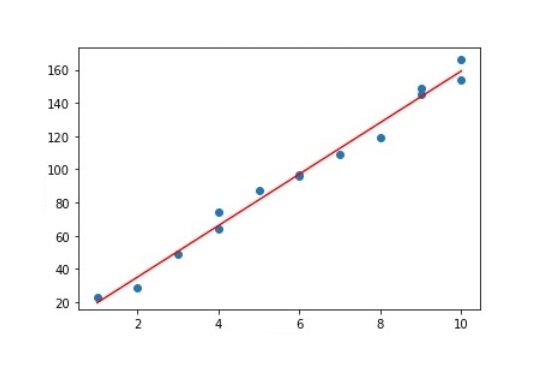
While developing the recommendation algorithms, it is important to find similarity in customer's interests. Linear algebra provides various methods to model this data and find similarities.

### Linear Regression

Linear regression is a statistical method to describe the relationship between variables. It is often used in machine learning to predict the value of a numeric continuous variable from a set of dependent variables. A simple linear regression equation can be written as:

y = ß0 + ß1x

Where, ß0 and ß1 are the coefficients. Solving the linear regression problem involves finding the value of these coefficients. Most machine learning libraries solve regression problem using linear algebraic techniques.



### Principal Component Analysis

Building a machine learning model involves extracting patterns from the given dataset. This dataset may contain massive number of features. The model built on such high dimensional data can be biased and less productive. Hence there is a need to find relevant features. Principal Component Analysis is one of the commonly used feature extraction method in data science which internally uses linear algebraic techniques.

### Representing Dataset

The field of Data Science aims at extracting knowledge and information present in the data. In most of the cases, this data or data files are stored in a table like structure. Each row of this table represents a record and each column represents a feature.

This dataset is stored as a ‘vector’ or ‘matrix’ for easy computation and manipulation. Vectors and Matrices are key data structures in linear algebra which can hold data efficiently and supports various operations.

**Scalar**

A**Scalar** is a quantity which is described by a numerical value. It is used to represent values such as:

* A number like 50
* Mass of sun in kilograms
* Length of the great wall of china in kilometres

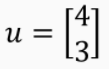
### # Define scalars number\_of\_units = 50 mass\_of\_sun = 1.989e30 length\_great\_wall\_of\_china = 21196 print(number\_of\_units) print(mass\_of\_sun) print(length\_great\_wall\_of\_china) Vector

A vector is an ordered collection of scalars. It can be represented in two forms:

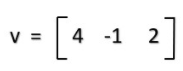
* **Column vector**: Contains n rows and 1 column
* **Row vector:**Contains 1 row and n columns

The number of scalars in a vector is called its **dimension**or**order**.

**Example**: A column vector 'u' of dimension 2 is illustrated below:



**Example:** A row vector 'v' of dimension 3 is illustrated below:



Row and column vectors can be created using the library NumPy as follows:

1. *# importing required library (numpy as np)*
2. import numpy as np
3. *# creating a column vector with 2 rows and 1 column using np.array() function*
4. col\_vec = np.array([[4],[3]])
5. print ("A column vector\n",col\_vec)
6. *# printing the shape of the vector*
7. print("Shape of vector=",col\_vec.shape)
8. *# importing required library (numpy as np)*
9. import numpy as np
10. *# creating a row vector with 1 row and 3 columns using np.array() function*
11. row\_vec = np.array([[4,-1,2]])
12. print("A row vector\n",row\_vec)
13. *# printing the shape of the vector*
14. print("Shape of vector=",row\_vec.shape)

**Zero vector** (z) is a vector in which each element is zero.

**One vector** (o) is a vector in which each element is one.

Zero and one vectors can be created using the library NumPy as follows:

1. *# importing required library (numpy as np)*
2. import numpy as np
3. *#Illustrating a Zero row vector of dimension 3 using np.array() function*
4. zero\_vector\_1=np.array([0,0,0])
5. print(zero\_vector\_1)
6. *#Illustrating a Zero row vector of dimension 3 using np.zeros() function*
7. zero\_vector\_2=np.zeros(3)
8. print(zero\_vector\_2)
9. *#Illustrating a Zero column vector of dimension 3 using np.zeros() and reshape() function*
10. zero\_vector\_3=np.zeros(3).reshape(3,-1)
11. print(zero\_vector\_3)
12. *# importing required library (numpy as np)*
13. import numpy as np
14. *#Illustrating a One vector of dimension 3 using np.array() function*
15. one\_vector\_1=np.array([1,1,1])
16. print(one\_vector\_1)
17. *#Illustrating a One vector of dimension 3 using np.ones() function*
18. one\_vector\_2=np.ones(3)
19. print(one\_vector\_2)
20. *#Illustrating a column vector of ones of dimension 3 using np.ones() and reshape() function*
21. one\_vector\_3=np.ones(3).reshape(3,-1)
22. print(one\_vector\_3)

### ****Indexing****

Let 'v' be the following vector:



You can select element(s) of a vector by specifying the corresponding index.

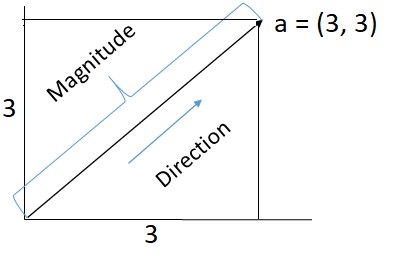
In python, the indexing starts from zero. You can select the second element of the vector 'v' as follows:

G:\Fifth Semester\Temp\10.png

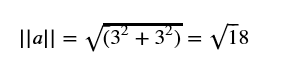
1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. vector=np.array([[4],[-1],[2]])
4. print("The vector\n",vector)
5. *#Addressing a value of vector by index*
6. print("\nElement at position 1=",vector[1])

### Magnitude and direction of a vector

A vector is also defined as an object which has both **magnitude** and **direction**. **Magnitude** is the distance between the origin and the point representing the vector. The**direction** is the path of displacement from the origin to the point which represents the vector.

**Example:** Consider a vector a = (3, 3). This can be plotted in 2-dimensional space as follows:   


The **magnitude (or norm)** of a vector v = [ v1, v2, v3,......vn ] (of dimension n) is computed as follows:    
G:\Fifth Semester\Temp\12.png

 Hence,the magnitude of the vector a = (3, 3) is ****

The norm() function in NumPy helps in finding the magnitude of a vector. It is illustrated below:

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Create a vector*
4. vector=np.array([3,3])
5. *#Magnitude of vector can be found using function np.linalg.norm() function*
6. print("\nMagnitude of the vector = ",np.linalg.norm(vector))

### ****Matrix****

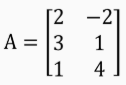
A **matrix** is defined as a rectangular array of numbers. It consists of rows and columns.

The number of rows (m) and columns (n) is defined as the **dimension or order** of the matrix and is represented as (m, n). A matrix is called a **square matrix** if the number of rows(m) is equal to the number of columns(n).

Using the library NumPy, you can create matrices of different orders as shown below:

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *# Creating matrices of various dimension using np.array() function*
4. matrix\_2x2 = np.array([[1, 2],
5. [3, 4]])
6. matrix\_3x3 = np.array([[1, 2, 3],
7. [4, 5, 6],
8. [8, 9, 10]])
9. print("2x2 Matrix\n",matrix\_2x2,"\nshape of matrix -> ",matrix\_2x2.shape)
10. print("\n3x3 Matrix\n",matrix\_3x3,"\nshape of matrix -> ",matrix\_3x3.shape)

### Accessing elements of a matrix

Consider the following matrix A,   


1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *# Creating a 3x2 Matrix "A"*
4. A = np.array([[2,-2],[3,1],[1,4]])

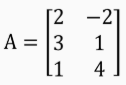
Recall that, indexing starts from 0. Elements of matrix A can be accessed using row-id and column-id as shown below:   
G:\Fifth Semester\Temp\15.png

1. *#selecting an element at row index 2 and column index 1*
2. print("Element of matrix A at index position [2,1] is", A[2,1])

### Transpose of a matrix

**Transpose** of a matrix is obtained by interchanging its rows and columns.

For matrix A,



the transposed matrix (**AT**) is given below,

G:\Fifth Semester\Temp\17.png

**Note**: The matrix A has 3 rows and 2 columns, and its Transpose has 2 rows and 3 columns.

NumPy library has a function **transpose()**which is used to obtain the transpose of a given matrix.

An example is illustrated below:

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *# Creating a 3x2 Matrix "A"*
4. A = np.array([[2,-2],[3,1],[1,4]])
5. print("Matrix A:\n",A)
6. *# Finding transpose of a matrix using the function transpose()*
7. print("Matrix A Transpose: \n",np.transpose(A))

### Types of Matrices

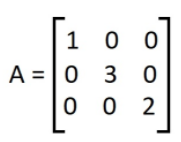
Matrices can be categorized on the basis of the value of their elements, position of elements,  number of rows and columns, etc. as follows:

* Diagonal matrix
* Identity matrix
* Symmetric matrix
* Triangular matrix

### Diagonal Matrix

A Square Matrix (D) is called a Diagonal Matrix, if D has zeros outside the main diagonal or principal diagonal. The Main diagonal or the principal diagonal are the elements on the diagonal that runs from the top left to bottom right.

**Example:** The matrix 'A' shown below illustrates a (3, 3) Diagonal matrix.

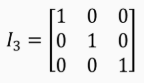


1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *# Creating a diagonal matrix with diagonal elements as (1,3,2)*
4. diagonal\_matrix = np.diag((1,3,2))
5. print(diagonal\_matrix)
6. *# Creating a diagonal matrix with a range of values*
7. matrix\_range= np.diag(np.arange(1,6,2))
8. print(matrix\_range)

### ****Identity Matrix****

An **identity matrix** is a square matrix of dimensions (n, n) having '1' across its main diagonal and '0' everywhere else. It is usually represented as 'In'

**Example:** The matrix shown below illustrates a (3, 3) Identity matrix.



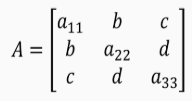
In NumPy, there are many methods to create an identity matrix.

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Creating an Identity matrix of size 2 using np.identity() function*
4. identity\_matrix\_1 = np.identity(2)
5. print("Identity\_matrix 1\n",identity\_matrix\_1)
6. *#Creating an Identity matrix of size 3 using np.eye() fucntion*
7. identity\_matrix\_2 = np.eye(3)
8. print("\nIdentity\_matrix 2\n",identity\_matrix\_2)

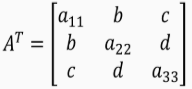
### ****Symmetric Matrix****

A square matrix A of dimension (n, n) is symmetric, if A = ATi.e.  the matrix A is the same as its transpose.

**Example:** Consider the following matrix A:



The Transpose of A (**AT**) is shown below:



Since A= AT, A is a **symmetric matrix**.

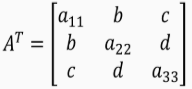
1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Creating matrix A*
4. A = np.array([[2,3,1],
5. [3,4,-1],
6. [1,-1,1]])
7. print("A:\n" , A)
8. *# Finding the Transpose of the matrix*
9. transposed\_matrix = A.transpose()
10. print("Transpose of A:\n" , transposed\_matrix)
11. *#Comparing each element of both matrices (returns a matrix of boolean compared values) and saving it in a variable comparison*
12. comparison = (A == transposed\_matrix)
13. *#Checking if all the elements in the matrix comparision is true*
14. equal\_arrays = comparison.all()
15. print(equal\_arrays)

### ****Triangular Matrix****

A **triangular matrix** can be either a lower triangular or an upper triangular matrix.

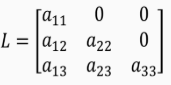
A **lower triangular matrix** is a square matrix in which all the elements above the main diagonal are zero.

**Example:** L is a lower triangular matrix of dimension (3, 3)



An **upper triangular matrix** is a square matrix in which all the elements below the main diagonal are zero.

**Example**: U is an upper triangular matrix of dimension (3, 3)



In NumPy, the functions **np.tril()** and **np.triu()**are used tocreate a lower triangular matrix and an upper triangular matrix, respectively.

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Creating a lower triangular matrix*
4. lower\_triangular\_matrix\_1 = np.tril([[1,2,3],
5. [4,5,6],
6. [7,8,9]])
7. print("Lower triangular matrix\n",lower\_triangular\_matrix\_1)
8. *# importing required libraries (numpy as np)*
9. import numpy as np
10. *#Creating a Upper triangular matrix*
11. upper\_triangular\_matrix\_1 = np.triu([[1,2,3],
12. [4,5,6],
13. [7,8,9]])
14. print("Upper triangular matrix\n",upper\_triangular\_matrix\_1)

**Exercise**

Create a square matrix A of order 3. Solve the following:

1. Convert A to an upper triangular matrix B
2. Convert B to a lower triangular matrix C
3. Find the transpose of matrix C
4. Check whether matrix C is a diagonal matrix
5. Check whether matrix C is symmetric

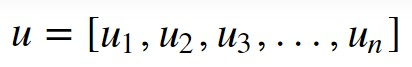
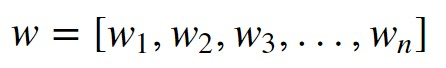
# Introduction - Operations over Vectors and Matrices

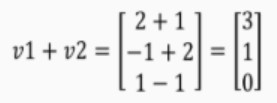
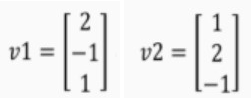
# Introduction - Operations over Vectors and Matrices

**Introduction**

Vectors of the same dimension can be added by adding their corresponding elements.

Consider 2 vectors **u** and **w** of dimension n as follows:

  
  
  
G:\Fifth Semester\Temp\26.png  
  
Now,



**Example:** Consider 2 column vectors v1 and v2 as follows:

The function **add()** of the NumPy library, is used to add vectors. Vector addition is illustrated below:

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Create 2 vectors*
4. vector\_1 = np.array([[2, -1, 1]])
5. vector\_2 = np.array([[1, 2, -1]])
7. print ("1st Vector : ", vector\_1)
8. print ("2nd Vector : ", vector\_2)
9. *# Addition of the vectors using the function np.add()*
10. out = np.add(vector\_1, vector\_2)
11. print ("Added Vector : ", out)

**Vector Addition**

Vectors of the same dimension can be added by adding their corresponding elements.

**Example:** Consider 2 column vectors v1 and v2 as follows:

The function **add()** of the NumPy library, is used to add vectors. Vector addition is illustrated below:

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Create 2 vectors*
4. vector\_1 = np.array([[2, -1, 1]])
5. vector\_2 = np.array([[1, 2, -1]])
7. print ("1st Vector : ", vector\_1)
8. print ("2nd Vector : ", vector\_2)
9. *# Addition of the vectors using the function np.add()*
10. out = np.add(vector\_1, vector\_2)
11. print ("Added Vector : ", out)

### Matrix Addition

Matrices of the same dimension can be added by adding their corresponding elements.

Consider 2 Matrices M1 and M2 of dimension (3, 3) as follows:  Then,

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *# Creation of 2 matrices*
4. matrix\_1 = np.array([[10,20,30],
5. [-30,-40,-50]])
6. matrix\_2 = np.array([[100,-200,300],
7. [30,50,70]])
9. print ("1st Matrix : \n", matrix\_1)
10. print ("2nd Matrix : \n", matrix\_2)
11. *# Addition of the matrices using np.add() function*
12. out = np.add(matrix\_1, matrix\_2)
13. print ("Added Matrix : \n", out)

### Multiplication of Vector by a Scalar

In the multiplication of a vector by a scalar, each element of the vector is multiplied by the given scalar quantity.

Consider a scalar **s** and a vector **v**, where   
The scalar product of s and v (**s.v**) is as given below:

**Example:** Suppose s = 2.5 and

Then,

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Defining a Vector 'v' and scalar 's'*
4. v = np.array([[2],[-1],[3]])
5. s = 2.5
6. *#Scalar Vector Multiplication*
7. vector\_mul = v\*s
8. print("Vector: \n",v,"\nScalar:",s,"\nScalar Vector multiplication:\n",vector\_mul)

### Multiplication of Matrix by a Scalar

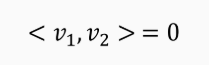
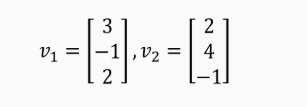
In the multiplication of a matrix by a scalar, each element of the matrix is multiplied by the given scalar quantity.

Consider a scalar quantity **s** and a matrix **A**. Where,   
The scalar product of s and A (**s.A**) is:

**Example:** Suppose s = 2.5 and    
Then,

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Defining a matrix 'A' and scalar 's'*
4. A = np.array([[1,2,1],
5. [-1,1,0],
6. [2,-1,1]])
7. print("Matrix:\n",A)
8. s = 2.5
9. print("Scalar :",s)
10. *#Multiplication of Matrix by a Scalar*
11. matrix\_mul = s \* A
12. print("Multiplication of Matrix by a Scalar:\n",matrix\_mul)

### Orthogonal vectors  If the inner product of two non-zero vectors v1 and v2 is zero, that is

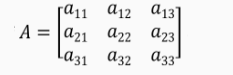
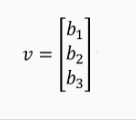
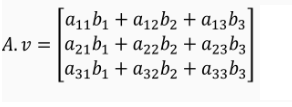
  
then, the vectors v1 and v2 are called**orthogonal** vectors.   
 G:\Fifth Semester\Temp\31.png

**Example:** Let v1 and v2 be two vectors as follows: The inner product of v1 and v2 is: So, the vectors v1 and v2 are orthogonal.

1. *# importing required libraries (numpy as np)*
2. import numpy as np
3. *#Creating vectors*
4. Vector\_1 = np.array([[3],[-1],[2]])
5. Vector\_2 = np.array([[2],[4],[-1]])
6. print("Vector 1\n",Vector\_1)
7. print("Vector 2\n",Vector\_2)
8. *#Finding the transpose of Vector\_1*
9. trans = np.transpose(Vector\_1)
10. *#Finding the dot product*
11. result = np.dot(trans,Vector\_2)
12. print("Dot Product\n",result)

**Matrix - Vector Multiplication**

Consider a matrix **A** of dimension (m, n) and a column vector**v** of dimension n. The product of **A**and **v**isrepresented as **A.v**, whichis a column vector of dimension m.

For example, if **A** is a matrix of dimension (3, 3) and **v** is a column vector of dimension 3, then the resultant matrix will be of dimension (3, 1), which is also a column vector.   
  

Now,

1. *#import the library numpy*
2. import numpy as np
3. *#Define a matrix*
4. A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
5. print("A :\n", A)
6. *#Define a column vector*
7. v = np.array([[1], [2], [3]])
8. print("v :\n", v)
9. *# Find product of the matrix and Vector*
10. product = np.matmul(A,v)
11. print("Product of A and v is: \n", product)

**Tensors**  
Tensorflow is an open-source machine learning framework that is used for **complex numerical computation.** It was developed by the Google Brain team in Google. **Tensorflow can train and run deep neural networks that can be used to develop several AI applications.**

**What is a Tensor?**   
**A tensor can be described as a n-dimensional numerical array.** A tensor can be called a generalized matrix. It could be a 0-D matrix (a single number), 1-D matrix (a vector), 2-D matrix or any higher dimensional structure. A tensor is identified by three parameters viz., rank, shape and size. The number of dimensions of the tensor is said to be its rank. The number of columns and rows that the tensor has, is said to be its shape. And, the data type assigned to the tensor’s elements is said to be its type.

**https://infyspringboard.onwingspan.com/web/en/viewer/video/lex\_auth\_01329493355928780838870\_shared  
  
Creating tensor using the constant() function.**

The most popular function for creating tensors in Tensorflow is the constant() function. We need to give values or list of values as argument for creating tensor. If the values given are of type integer, then **int32** is the default data type. And if the values given are of floating type, then float32 is the default data type.  
  
  
# Program to create tensor using the constant() function  
# importing tensorflow

import tensorflow as tf

# creating nodes in computation graph

node1 = tf.constant(5, dtype=tf.int32)

node2 = tf.constant(5, dtype=tf.int32)

node3 = tf.add(node1, node2)

# create tensorflow session object

sess = tf.compat.v1.Session()

# evaluating node3 and printing the result

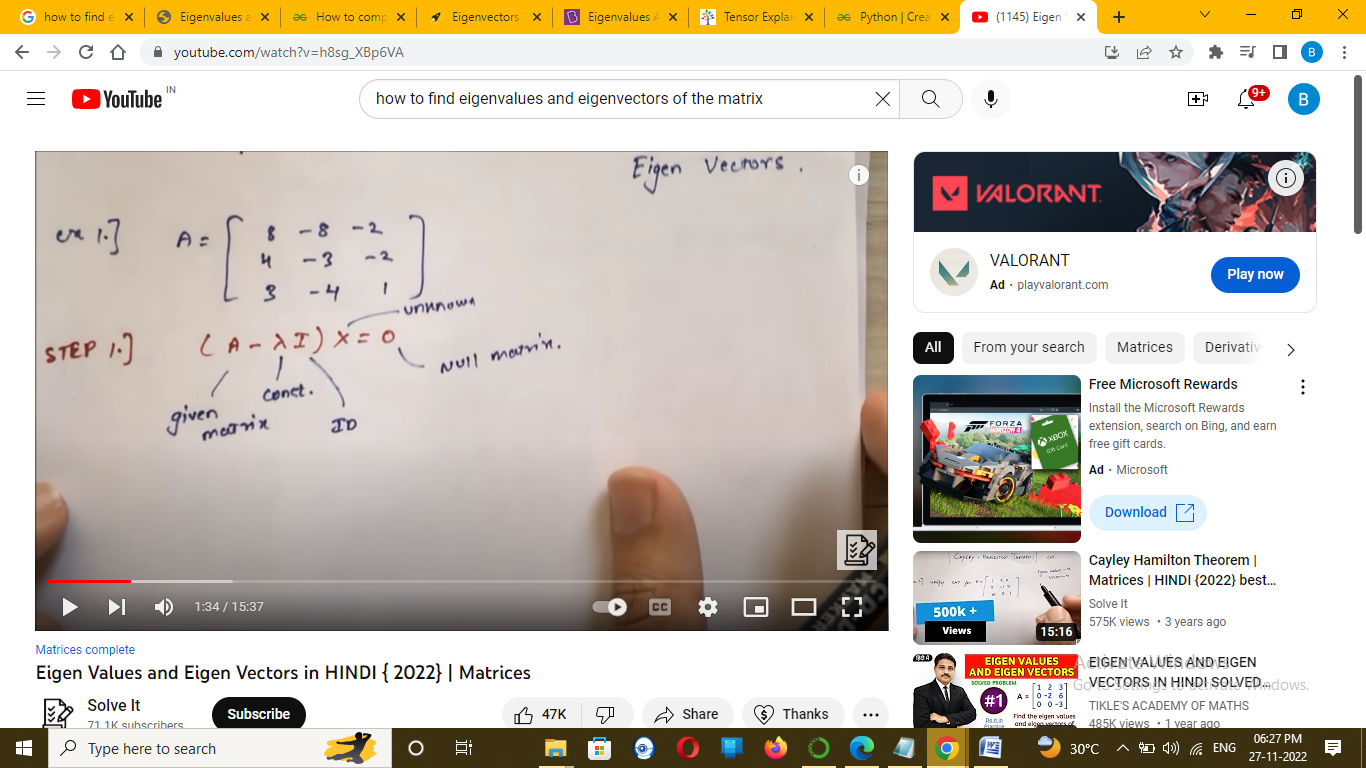
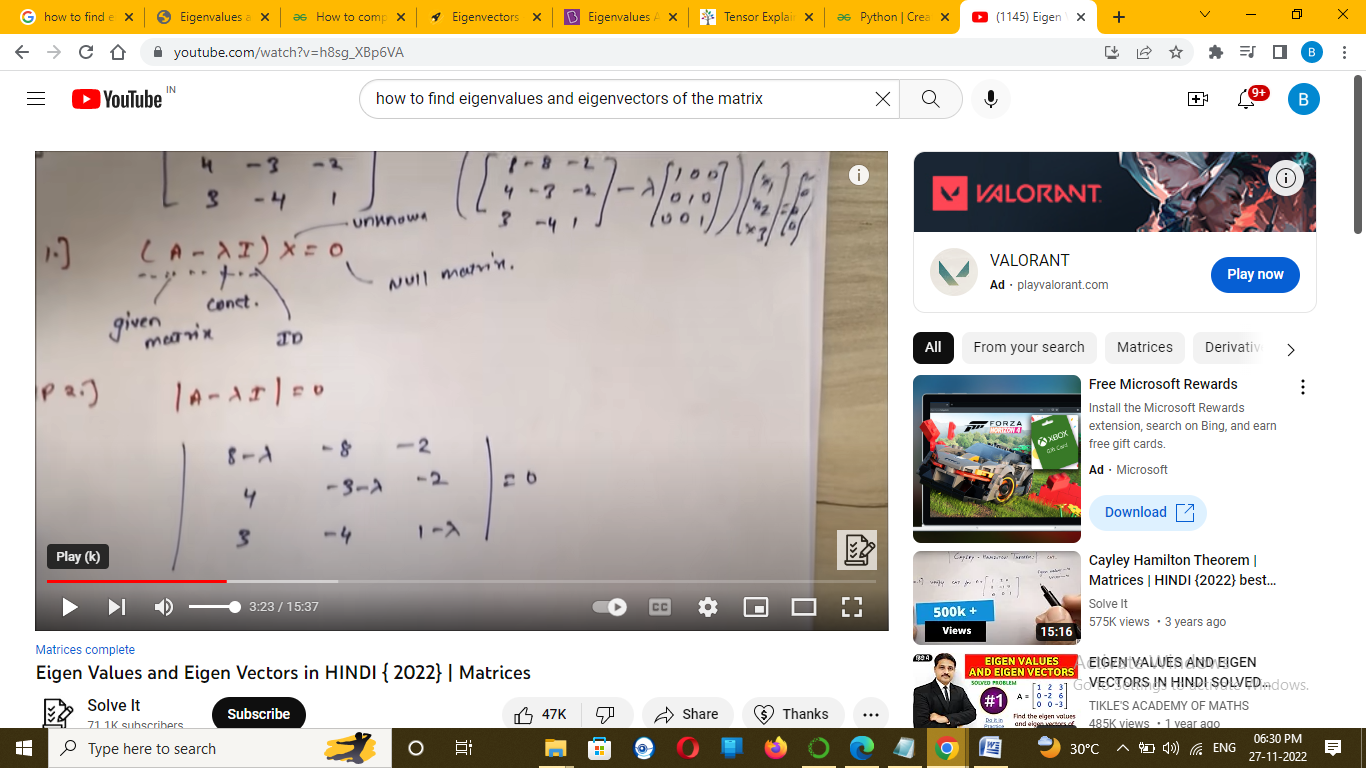
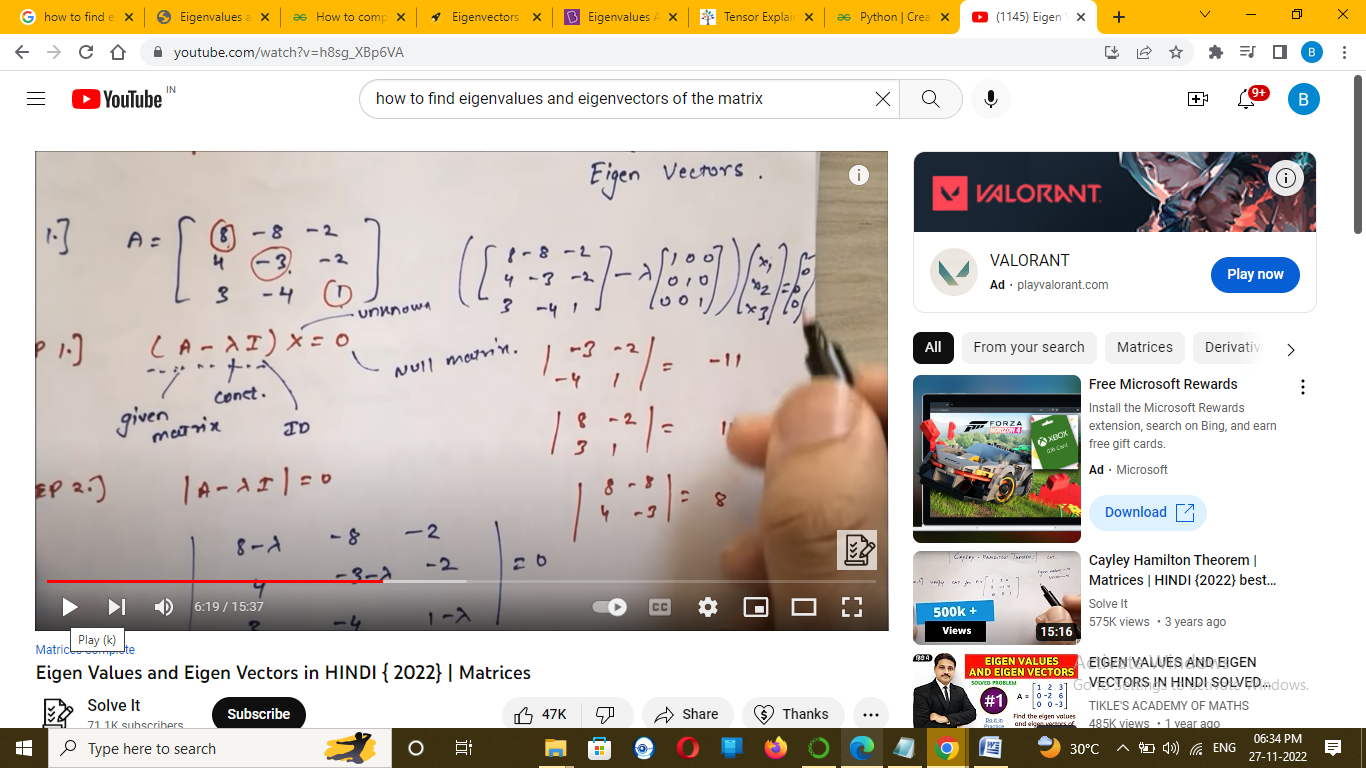
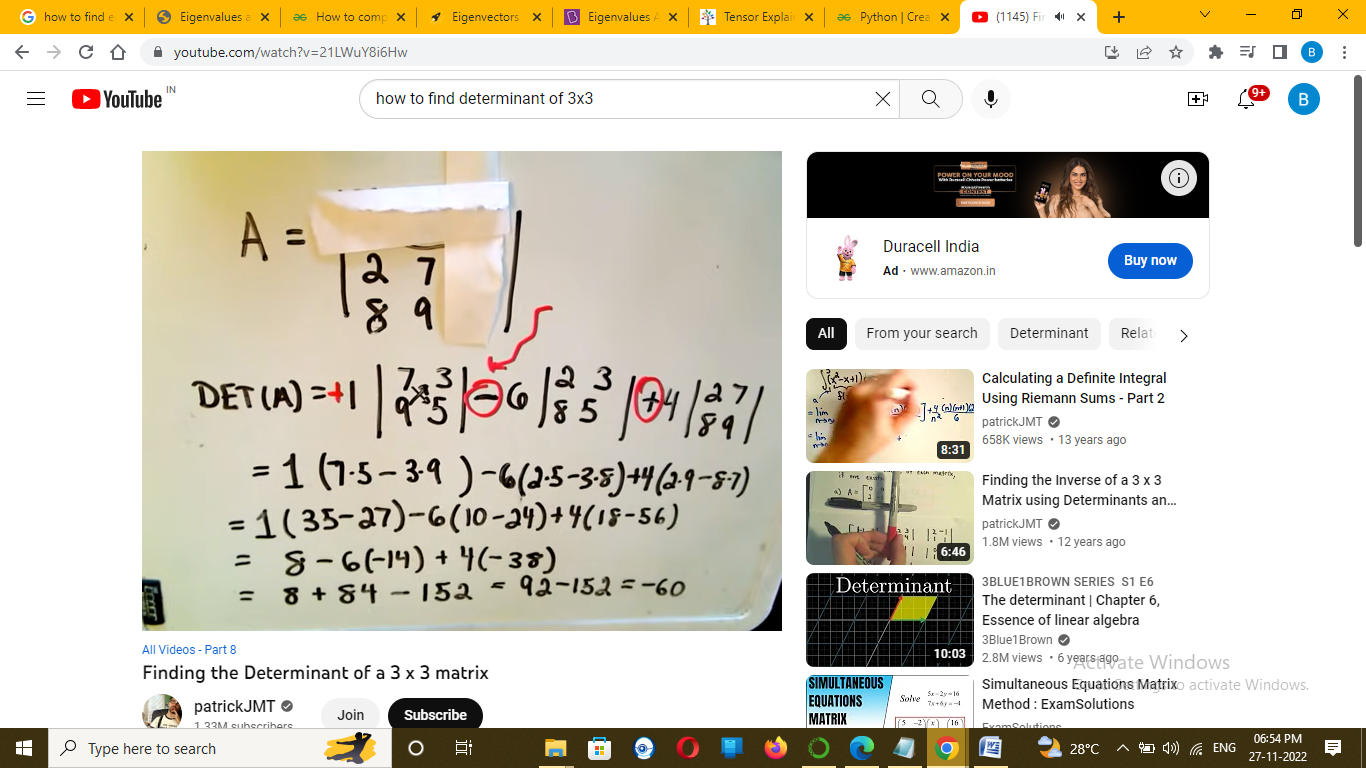
print("sum of node1 and node2 is :",(node3))

# print("sum of node1 and node2 is :",sess.run(node3))

# closing the session

sess.close()

**Output:**sum of node1 and node2 is : tf.Tensor(10, shape=(), dtype=int32)

  
  
  
  
  
<https://www.youtube.com/watch?v=h8sg_XBp6VA>  
  
  
  
  
  
  
  
  
  
Eigenvalues and Eigenvectors have their importance in linear differential equations where **you want to find a rate of change or when you want to maintain relationships between two variables**.   
  
***Syntax:****numpy.linalg.eig()****Parameter:****An square array.*

***Return:****It will return two values first is eigenvalues and second is eigenvectors.*

**Example 1:**

# importing numpy library   
import numpy as np   
# create numpy 2d-array   
m = np.array([[1, 2],[2, 3]])   
print("Printing the Original square array:\n", m)  
# finding eigenvalues and eigenvectors   
w, v = np.linalg.eig(m)   
# printing eigen values   
print("Printing the Eigen values of the given square array:\n",w)   
# printing eigen vectors   
print("Printing Right eigenvectors of the given square array:\n",v)

### 

The **norm of a vector** refers to the length or the magnitude of a vector. There are different ways to calculate the length. The norm of a vector is a non-negative value.

Norm of a vector x is denoted as: ‖**x**‖

The norm of a vector is a measure of its distance from the origin in the vector space.

To calculate the norm, you can either use [Numpy](https://www.digitalocean.com/community/tutorials/python-numpy-tutorial) or [Scipy.](https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.norm.html" \l "scipy.linalg.norm) Both offer a similar function to calculate the norm.

**These are :**

* L1 Norm
* L2 Norm

## How to Calculate the L1 Norm of a Vector?

L1 Norm of a vector is also known as the **Manhattan distance** or **Taxicab norm**. The notation for L1 norm of a vector x is ‖**x**‖1.

To calculate the norm, you need to take the **sum of the absolute vector values.**

Let’s take an example to understand this:

a = [1,2,3,4,5]

For the array above, the L1 norm is going to be:

1+2+3+4+5 = 15

Let’s take another example:

a = [-1,-2,3,4,5]

The L1 norm of this array is :

|-1|+|-2|+3+4+5 = 15

The L1 norm for both the vectors is the same as we consider absolute values while computing it.

### Python Implementation of L1 norm

Let’s see how can we calculate L1 norm of a vector in Python.

### Using Numpy

The Python code for calculating L1 norm using Numpy is as follows :

from numpy import array

from numpy.linalg import norm

arr = array([1, 2, 3, 4, 5])

print(arr)

norm\_l1 = norm(arr, 1)

print(norm\_l1)

Output :

[1 2 3 4 5]

15.0

Let’s try calculating it for the array with negative entries in our example above.

from numpy import array

from numpy.linalg import norm

arr = array([-1, -2, 3, 4, 5])

print(arr)

norm\_l1 = norm(arr, 1)

print(norm\_l1)

Output :

[-1 -2 3 4 5]

15.0

### Using Scipy

To calculate L1 using Scipy is not very different from the implementation above.

The code for same is:

from numpy import array

from scipy.linalg import norm

arr = array([-1, -2, 3, 4, 5])

print(arr)

norm\_l1 = norm(arr, 1)

print(norm\_l1)

Output :

[-1 -2 3 4 5]

15.0

The code is exactly similar to the Numpy one.

## How to Calculate L2 Norm of a Vector?

The notation for the L2 norm of a vector x is ‖**x**‖2.

To calculate the L2 norm of a vector, take the square root of the sum of the squared vector values.

Another name for L2 norm of a vector is **Euclidean distance.** This is often used for calculating the error in machine learning models.

The Root Mean square error is the Euclidean distance between the actual output of the model and the expected output.

The goal of a [machine learning](https://www.digitalocean.com/community/tutorials/introduction-to-machine-learning) model is to reduce this error.

Let’s consider an example to understand it.

a = [1,2,3,4,5]

The L2 norm for the above is :

sqrt(1^2 + 2^2 + 3^2 + 4^2 + 5^2) = 7.416

L2 norm is always a positive quantity since we are squaring the values before adding them.

### Python Implementation

The Python implementation is as follows :

from numpy import array

from numpy.linalg import norm

arr = array([1, 2, 3, 4, 5])

print(arr)

norm\_l2 = norm(arr)

print(norm\_l2)

Output :

[1 2 3 4 5]

7.416198487095663

Eigendecomposition is a technique used in Linear Algebra to break down a matrix into its constituent parts.

**Why is this matrix decomposition important?**

As stated, a matrix is a transformation that maps a vector from one point to another in the vector space.

In machine learning algorithms, we often apply such transformations several times until the final output is obtained at each phase of the algorithm.

However, the application also depends on the complexity of the problem. This means at the end, we will have taken our matrix raised to a power of a certain number.