Linear Algebra

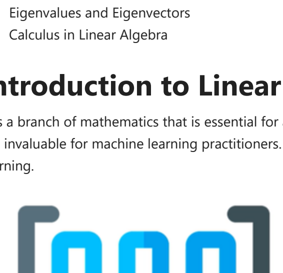
Agenda

In this session, we will cover the following concepts with the help of a business use case:

- Linear Algebra
- Scalars and Vectors
- Matrix and Matrix Operations
- Rank of Matrix
- Determinant of Matrix
- Inverse of Matrix
- Eigenvalues and Eigenvectors
- Calculus in Linear Algebra

Introduction to Linear Algebra

It is a branch of mathematics that is essential for a deep understanding of machine learning. Although the field is vast, its basic concepts are invaluable for machine learning practitioners. In this introduction, linear algebra is considered from the perspective of machine learning.



The data in linear algebra is represented by linear equations. These in turn are represented by matrices and vectors. Matrices and vectors simplify the process of representing large amounts of information.

A matrix consists of rows and columns of numbers, variables, or expressions.

While building machine learning models, it is required to reduce the dimensions of data or select the right hyperparameters. It is here where the role of linear algebra in machine learning arises. This is because notations and formalizations of linear algebra can describe and execute complex operations used in machine learning.

The parts of linear algebra that are used in machine learning are:

1. **Notation:** A clear idea of notations simplifies the understanding of algorithms in papers and books. This is true even while reading Python code.
2. **Operations:** It is easier to understand while working with vectors and matrices at an abstract level. The operations which are useful to perform on matrices and vectors are addition, multiplication, inversion, transpose, etc.

Addition

Multiplication

Inversion

Transpose

1. **Matrix factorization:** Matrix factorization is also essential for machine learning. Matrix decomposition methods like SVD and QR simplify regression algorithms.

Linear algebra is a building block for machine learning algorithms, it helps the ML practitioner tweak the parameters of the algorithms to achieve the desired accuracy. The practitioner will be able to develop algorithms from scratch and devise new ones from existing templates.

Scalars and Vectors

A number of measurable quantities, such as length, area, and volume, can be completely determined by specifying only their magnitude. These quantities are known as scalars.

A vector quantity is defined as the physical quantity that has both directions as well as magnitude. Such as velocity, force, and acceleration require both a magnitude and a direction for their description.

For example: Wind velocity is a vector that has a speed and direction, such as 15 miles/hour northeast. Geometrically, vectors are usually represented by arrows or directed line segments.

Linear algebra is the study of vectors. At the most general level, vectors are ordered finite lists of numbers. Vectors are the most fundamental mathematical objects in machine learning. These are used to represent attributes of entities such as income, test scores, etc.

Bold lower-case letter like **x**

Lower-case letter with an arrow on top like $\vec{x}$

Vector Representation:

$$\vec{x}\vec{x}$$

Vectors are a type of mathematical object that can be added together or multiplied by a number to obtain another object of the same kind.

For example:

If a vector x represents the math score x = math_scores and a second vector y = science_scores, then both can be added together to obtain a third vector

z = x + y.

First Vector | Second Vector
X = math_scores | X = science_scores

Third Vector
Z = x + y

Multiply a Vector by a constant
For example, 2 X x to obtain 2x, again a vector.
The returning object is still a vector.

For example:

If a vector x is multiplied by constant 2 then 2 X x = 2x, will be a vector.

Dot Product of Two Vectors

The dot product of two vectors is the sum of the product of the corresponding elements of two vectors.

$$x.y = x_1y_1+x_2y_2+x_3y_3+.........+x_ny_n$$

In vector algebra, if two vectors are given as:

$\vec{x} = [x_1,x_2,x_3,x_4,.....x_n]$ $\vec{y} = [y_1,y_2,y_3,y_4,.....y_n]$

Dot product:
$x.y = x_1y_1+x_2y_2+x_3y_3+.........+x_ny_n$

$$\vec{x}.\vec{y} = \Sigma_{i=1}^{n} x_i . y_i$$

Example of the dot product of two vectors:

A numerical example of the dot product of two vectors is shown below:

$$\vec{x} = \begin{bmatrix}5\\6\\-7\end{bmatrix} \qquad \vec{y} = \begin{bmatrix}-1\\2\\-5\end{bmatrix}$$

$$\vec{x}.\vec{y} = \begin{bmatrix}5*-1\\6*2\\-7*-5\end{bmatrix} = -5+12+35 = 42$$

Matrix representation of dot product

The dot product of vectors can be easily computed if the vectors are represented by row or column matrices. The first matrix is transposed to get a row vector. It is then multiplied with the second column matrix to get the dot product.

$$\vec{x} = \begin{bmatrix}x1\\x2\\x3\end{bmatrix} \qquad \vec{y} = \begin{bmatrix}y1\\y2\\y3\end{bmatrix} \qquad \vec{x}T = \begin{bmatrix}x1 & x2 & x3\end{bmatrix}$$

$$\vec{x} . \vec{y} = \begin{bmatrix}x1 & x2 & x3\end{bmatrix}\begin{bmatrix}y1\\y2\\y3\end{bmatrix} = x1y1 + x2y2 + x3y3$$

Python code snippet for dot product

Let us see the code for dot product.

In [1]:
```
#Define a function to calculate dot product of two matrices
def dot_product(x,y):
    #Ensure that both the vectors have same length
    return sum(i*j for i,j in zip(x,y))
```

In [22]:
```
dot_product([3,2,6],[1,7,-2])
```

Out[22]:
```
5
```

Note: In the above code zip function ensures that both the vectors have the same length.

When dot_product([3,2,6],[1,7,-2]) will be called then the output will be 5.

Linear Independence of Vectors

A set of vectors is said to be linearly independent if no vector in the set can be expressed as a linear combination of the other vectors in the same set. If not, the set is linearly dependent.

A set of vectors v1,v2,v3,......,vn is linearly dependent if there are scalars c1,c2,c3,.......,cn; at least one of which is not zero, such that

$$C_1V_1 + C_2V_2 + C_3V_3 + ... + C_nV_n = 0$$

A set of vectors that don't satisfy the above condition is called linearly independent.

Numerical example:

Suppose there are three vectors are as follows:

$$\begin{bmatrix}2\\6\end{bmatrix} - 2\begin{bmatrix}1\\3\end{bmatrix} + 0\begin{bmatrix}4\\1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

The above three vectors are linearly dependent, as at least one (in this case, two) of the three scalars 1, - 2, and 0 are nonzero.

Norm of a Vector

The norm of a vector is the length of the vector. For a vector v, it is denoted by and is given by

$$\|v\| = \sqrt{v.v}$$

From the above definition, it follows that to arrive at the norm of a vector, it is required to take the dot product of the vector with itself and then calculate the square root of the result.

Example:
A numerical example of the norm of a vector.

$$V = \begin{bmatrix}2\\5\\-3\end{bmatrix}$$

$$v . v = 2*2 + 5*5 + (-3*-3) = 4 + 25 + 9 = 38$$

Norm of the vector v is $\|v\| = \sqrt{v.v} = \sqrt{38} = 6.16$

Python code to find the norm of a vector

The norm of a vector, by definition, is the square root of its dot product with itself. Thus, in this Python code, the elements of the vector are fed into the system, after which the system calculates the square of every element. The new elements are added to get the dot product. Finally, the system displays the result's square root, or the norm, as the output.

The Python code for finding the norm of a vector is as follows:

In [37]:
```
#Define vector
v = [-1, -2, 3, 4, 5]
```

In [38]:
```
#Define function to calculate norm of a vector
import math
def norm_vector(v):
    dot_product = sum(i*i for i in v)
    return math.sqrt(dot_product)
```

In [39]:
```
#Call the function
norm_vector(v)
```

Out[39]:
```
7.416198487095663
```

Matrix

A matrix is a rectangular array of numbers or expressions arranged into columns and rows. A matrix is used to represent a mathematical

$$\begin{pmatrix}1 & 2 & 2\\2 & 5 & 7\\2 & 7 & 9\end{pmatrix} \qquad \begin{pmatrix}7 & 4 & 2\\6 & 8 & 1\\3 & 7 & 8\end{pmatrix}$$

object or a property of the object.

If X[a_{ij}] and Y[b_{ij}] are $m \times n$ matrices then,

X+Y is the sum of mXn matrix obtained by adding the corresponding elements

$$X+Y = [a_{ij}+b_{ij}]$$

Two matrices of different orders cannot be added.

Example: Matrix Addition of 3 x 3 matrices

$$\mathbf{X} = \begin{pmatrix}2 & -1 & 5\\10 & 3 & 1\\-7 & 0 & 3\end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix}3 & 2 & 9\\-7 & -7 & 6\\1 & 2 & -4\end{pmatrix}$$

$$\mathbf{X + Y} = \begin{pmatrix}2+3 & -1+2 & 5+9\\10+(-7) & 3+(-7) & 1+6\\-7+1 & 0+2 & 3+(-4)\end{pmatrix} = \begin{pmatrix}5 & 1 & 14\\3 & -4 & 7\\-6 & 2 & -1\end{pmatrix}$$

Python implementation for the addition of two matrices with same order

In [1]:
```
#Define a function to add matrices
def matrix_addition(x,y):
    xrows = len(x)
    xcols = len(x[0])
    yrows = len(y)
    ycols= len(y[0])
    if xrows!=yrows or xcols!=ycols:
        print("Sum is not defined as the matrices have different orders")
    else:
        result=[[0 for i in range(xcols)] for i in range(xrows)]
        for i in range(xrows):
            for j in range(xcols):
                result[i][j] = matrix_X[i][j]+matrix_Y[i][j]
        return result
```

In [7]:
```
#Taking input from user
print("Enter the rows and columns of first matrix")
rows1 = int(input("Enter the Number of rows : " ))
column1 = int(input("Enter the Number of Columns: "))

print("Enter the elements of First Matrix:")
matrix_X= [[int(input()) for i in range(column1)] for i in range(rows1)]
print("First Matrix is: ")
for n in matrix_X:
    print(n)
print("Enter the rows and columns of second matrix")
rows2 = int(input("Enter the Number of rows : " ))
column2 = int(input("Enter the Number of Columns: "))

print("Enter the elements of Second Matrix:")
matrix_Y= [[int(input()) for i in range(column2)] for i in range(rows2)]
for n in matrix_Y:
    print(n)
```

```
Enter the rows and columns of first matrix

Enter the elements of First Matrix:

First Matrix is:
[1, 2]
[3, 4]
Enter the rows and columns of second matrix

Enter the elements of Second Matrix:

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

In [10]:
```
#Return the value of function
matrix_addition(matrix_X,matrix_Y)
```

Out[10]:
```
[[2, 4], [6, 8]]
```

The code takes two matrices of the same order and adds them. If the matrices are of a different order, it prints an error code, indicating the same.

Addition of Matrices with different orders

In [11]:
```
#Taking input from user
print("Enter the rows and columns of first matrix")
rows1 = int(input("Enter the Number of rows : " ))
column1 = int(input("Enter the Number of Columns: "))

print("Enter the elements of First Matrix:")
matrix_X= [[int(input()) for i in range(column1)] for i in range(rows1)]
print("First Matrix is: ")
for n in matrix_X:
    print(n)
print("Enter the rows and columns of second matrix")
rows2 = int(input("Enter the Number of rows : " ))
column2 = int(input("Enter the Number of Columns: "))

print("Enter the elements of Second Matrix:")
matrix_Y= [[int(input()) for i in range(column2)] for i in range(rows2)]
for n in matrix_Y:
    print(n)
```

```
Enter the rows and columns of first matrix

Enter the elements of First Matrix:

First Matrix is:
[1, 2, 3]
[4, 5, 6]
Enter the rows and columns of second matrix

Enter the elements of Second Matrix:

[1, 2]
[3, 4]
[5, 6]
```

In [12]:
```
matrix_addition(matrix_X,matrix_Y)
```

```
Sum is not defined as the matrices have different orders
```

As we can see sum will not be performed in the matrices with different orders.

Scalar Multiplication

Scalar multiplication of a matrix refers to each element of the matrix being multiplied by the given scalar.

If X is an mXn matrix and c is a scalar, then cX is the mXn matrix obtained by multiplying every element of X with c.

$$cX = c[a_{ij}] = [ca_{ij}]$$

So formally, cX = c[a_{ij}] = [ca_{ij}].

Scalar Multiplication: Example

$$\mathbf{X} = \begin{pmatrix}4 & 0 & 3\\-7 & 1 & 2\end{pmatrix}$$

$$\mathbf{2X} = \begin{pmatrix}8 & 0 & 6\\-14 & 2 & 4\end{pmatrix} \qquad \mathbf{½ X} = \begin{pmatrix}2 & 0 & 1.5\\-3.5 & 0.5 & 1\end{pmatrix} \qquad \mathbf{-X} = \begin{pmatrix}-4 & 0 & -3\\7 & -1 & -2\end{pmatrix}$$

Python code snippet for scalar multiplication

This snippet takes a scalar value and a matrix as input and gives a resultant matrix, the elements of which are the products of the original matrix and the scalar value.

In [40]:
```
#Define function for scalar multiplication
def scalar_multiplication(c,X):
    cX = X
    for i in range(len(X)):
        for j in range(len(X[0])):
            cX[i][j] = c*cX[i][j]
    return cX
```

In [35]:
```
scalar_multiplication(-3,[[2,6,-1],[2,8,0],[9,8,7]])
```

Out[35]:
```
[[-6, -18, 3], [-6, -24, 0], [-27, -24, -21]]
```

When the matrix 2,6, -1, 2, 8, 0, 9, 8, 7 is multiplied by -3, the resultant matrix will be -6,-18,3,-6,-24,0,-27,-24,-21

Output: [[-6,-18,3], [-6,-24,0], [-27,-24,-21]]

Matrix Operations

1.Matrix Subtraction: Subtraction of matrices involves element-wise subtraction. If X[a_{ij}]and Y[b_{ij}]are $m \times n$ matrices, their difference X-Y is the $m \times n$ matrix obtained by subtracting the corresponding elements of Y from those of X.

So, X-Y = [a_{ij} - b_{ij}]

If, X[a_{ij}] and Y[b_{ij}] are mXn matrices, then:

$$X\text{-}Y = [a_{ij} - b_{ij}]$$

If X and Y are of different orders, then subtraction is not possible.

Matrix Subtraction: Example

$$\mathbf{X} = \begin{pmatrix}5 & -9\\0 & 2\\0 & 4\end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix}3 & 0\\6 & -5\\2 & 5\end{pmatrix}$$

$$\mathbf{X - Y} = \begin{pmatrix}5-3 & -9-0\\0-6 & 2-(-5)\\0-2 & 4-5\end{pmatrix} = \begin{pmatrix}2 & -9\\-6 & 7\\-2 & -1\end{pmatrix}$$

Consider two 3*2 matrices X and Y:

Python code snippet to perform matrix subtraction

In the code, matrix_subtraction will take two matrices and it will first check the order of matrices. If it is the same then, perform a subtraction operation. Else, it prints an error message.

In [42]:
```
#Define function for subtraction of matrices
def matrix_subtraction(x,y):
    xrows = len(x)
    xcols = len(x[0])
    yrows = len(y)
    ycols= len(y[0])
    if xrows!=yrows or xcols!=ycols:
        print("Subtraction is not defined as the matrices have different orders")
    else:
        result=[[0 for i in range(xcols)] for i in range(xrows)]
        for i in range(xrows):
            for j in range(xcols):
                result[i][j] = matrix_X[i][j]-matrix_Y[i][j]
        return result
```

In [43]:
```
#Taking input from user
print("Enter the rows and columns of first matrix")
rows1 = int(input("Enter the Number of rows : " ))
column1 = int(input("Enter the Number of Columns: "))

print("Enter the elements of First Matrix:")
matrix_X= [[int(input()) for i in range(column1)] for i in range(rows1)]
print("First Matrix is: ")
for n in matrix_X:
    print(n)
print("Enter the rows and columns of second matrix")
rows2 = int(input("Enter the Number of rows : " ))
column2 = int(input("Enter the Number of Columns: "))

print("Enter the elements of Second Matrix:")
matrix_Y= [[int(input()) for i in range(column2)] for i in range(rows2)]
for n in matrix_Y:
    print(n)
```

```
Enter the rows and columns of first matrix

Enter the elements of First Matrix:

First Matrix is:
[5, 2, 3]
[3, 4, -9]
Enter the rows and columns of second matrix

Enter the elements of Second Matrix:

[5, 3, 2]
[8, 2, 4]
```

In [44]:
```
matrix_subtraction(matrix_X,matrix_Y)
```

Out[44]:
```
[[0, -1, 1], [-5, 2, -13]]
```

matrix_subtraction([[5,2,3],[3,4,-9]],[[5,3,2],[8,2,4]]) Here, input matrices are 5,2,3,3,4,-9 and 5,3,2,8,2,4 The resultant matrix will be 0,-1,1,-5,2,-13

Output:[[0,-1,1], [-5,2,-13]]

Subtraction of Matrices with Different Orders

In [18]:
```
#Taking input from user
print("Enter the rows and columns of first matrix")
rows1 = int(input("Enter the Number of rows : " ))
column1 = int(input("Enter the Number of Columns: "))

print("Enter the elements of First Matrix:")
matrix_X= [[int(input()) for i in range(column1)] for i in range(rows1)]
print("First Matrix is: ")
for n in matrix_X:
    print(n)
print("Enter the rows and columns of second matrix")
rows2 = int(input("Enter the Number of rows : " ))
column2 = int(input("Enter the Number of Columns: "))

print("Enter the elements of Second Matrix:")
matrix_Y= [[int(input()) for i in range(column2)] for i in range(rows2)]
for n in matrix_Y:
    print(n)
```

```
Enter the rows and columns of first matrix
Enter the Number of rows : 2
Enter the Number of Columns: 1
Enter the elements of First Matrix:
2
3
First Matrix is:
[2]
[3]
Enter the rows and columns of second matrix
Enter the Number of rows : 2
Enter the Number of Columns: 2
Enter the elements of Second Matrix:
2
4
5
6
[2, 4]
[5, 6]
```

In [19]:
```
matrix_subtraction(matrix_X,matrix_Y)
```

```
Subtraction is not defined as the matrices have different orders
```

As we can see, subtraction will not be performed in the matrices with different orders.

2.Matrix Multiplication: The product of two matrices is obtained by multiplying the elements of the rows of the first matrix with the corresponding elements of the columns of the second matrix.

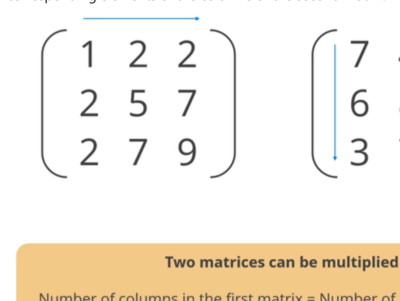
$$\begin{pmatrix}1 & 2 & 2\\2 & 5 & 7\\2 & 7 & 9\end{pmatrix} \qquad \begin{pmatrix}7 & 4 & 2\\6 & 8 & 1\\3 & 7 & 8\end{pmatrix}$$

Two matrices can be multiplied only if:

Number of columns in the first matrix = Number of rows in the second matrix

If X is an $m \times n$ matrix and Y is an $n \times m$, then their product Z = $X \times Y$ is an $m \times r$ matrix, whose elements are given by the following expression.

Z_{ij} = X $_{i1}$Y $_{1j}$ + X $_{i2}$Y $_{2j}$ + X $_{in}$and Y $_{nj}$



$$z_{ij} = x_{i1}y_{1j} + x_{i2}y_{2j} + ... + x_{in}y_{nj}$$

Matrix Multiplication: Example

Consider two matrices X, Y where the order of X is 2X3 and the order of Y is 3X2

$$\mathbf{X} = \begin{pmatrix}2 & 0 & 5\\8 & -3 & 5\end{pmatrix} \qquad \mathbf{Y} = \begin{pmatrix}5 & 1\\2 & 2\\-3 & -8\end{pmatrix}$$

$$\mathbf{XY} = \begin{pmatrix}(2*5)+(0*2)+(5*-3) & (2*1)+(0*2)+(5*-8)\\(8*5)+(-3*2)+(5*-3) & (8*1)+(-3*2)+(5*-8)\end{pmatrix} = \begin{pmatrix}-5 & -38\\19 & -38\end{pmatrix}$$

Python code snippet for the matrix multiplication

In [45]:
```
#Define function to perform matrix multiplication
def matrix_multiplication(x,y):
    xrows = len(x)
    xcols = len(x[0])
    yrows = len(y)
    ycols= len(y[0])
    if xcols!=yrows:
        Print ("Product is not defined as the no. of rows in the first matrix is not equal to the no. of colum
    else:
        z = [ [ 0 for i in range(ycols) ] for j in range(xrows) ]
        for i in range(xrows):
            for j in range(ycols):
                total = 0
                for ii in range(xcols):
                    total += x[i][ii] * y[ii][j]
                z[i][j] = total
        return z
```

In [5]:
```
#Taking input from user
print("Enter the rows and columns of first matrix")
rows1 = int(input("Enter the Number of rows : " ))
column1 = int(input("Enter the Number of Columns: "))

print("Enter the elements of First Matrix:")
matrix_X= [[int(input()) for i in range(column1)] for i in range(rows1)]
print("First Matrix is: ")
for n in matrix_X:
    print(n)
print("Enter the rows and columns of second matrix")
rows2 = int(input("Enter the Number of rows : " ))
column2 = int(input("Enter the Number of Columns: "))

print("Enter the elements of Second Matrix:")
matrix_Y= [[int(input()) for i in range(column2)] for i in range(rows2)]
for n in matrix_Y:
    print(n)
```

```
Enter the rows and columns of first matrix
Enter the Number of rows : 2
Enter the Number of Columns: 3
Enter the elements of First Matrix:
1
2
3
4
5
6
First Matrix is:
[1, 2, 3]
[4, 5, 6]
Enter the rows and columns of second matrix
Enter the Number of rows : 3
Enter the Number of Columns: 2
Enter the elements of Second Matrix:
1
2
3
4
5
6
[1, 2]
[3, 4]
[5, 6]
```

In [6]:

```
matrix_multiplication(matrix_X,matrix_Y)
```

Out[6]:

```
[[22, 28], [49, 64]]
```

When two matrices of order 2X3 and 3X2 will be multiplied then the output will be in 2X2.

3.Transpose of a Matrix: The transpose of an $m \times n$ matrix X is the $n \times m$ matrix X^T obtained by interchanging the rows and columns of X.

The i^{th} column of X^T is the i^{th} row of X for all i.

Example:

$$X = \begin{pmatrix} 1 & 9 & -6 \\ 5 & 3 & -7 \end{pmatrix}$$

$$X^T = \begin{pmatrix} 1 & 5 \\ 9 & 3 \\ -6 & -7 \end{pmatrix}$$

X is a 2X3 matrix.The transpose X^T of X will be 3X2 matrix.

Python implementation for finding the transpose of a matrix

In [46]:

```
#Define function to perform transpose of matrix
def matrix_transpose(x):
    xrows = len(x)
    xcols = len(x[0])
    z = [ [ 0 for i in range(xrows) ] for j in range(xcols) ]
    for i in range(xcols):
        for j in range(xrows):
            z[i][j] = x[j][i]
    return z
```

In [29]:

```
matrix_transpose([[1,9,-6],[5,3,-7]])
```

Out[29]:

```
[[1, 5], [9, 3], [-6, -7]]
```

The code takes a matrix of order m*n and finds its transpose. The transpose of the matrix 1,2,5,3,5,4 gives an output of [[1,3],[2,5],[5,4]] as shown on the screen.

Output: [[1,3],[2,5],[5,4]]

Rank of a Matrix:

The rank of a matrix is the number of nonzero rows in its row echelon form.

$$\begin{pmatrix} 1 & 2 & 2 \\ 2 & 5 & 7 \\ 0 & 0 & 0 \end{pmatrix}$$

Rank of a matrix is denoted by rank(X).

Hence, to find the rank of a matrix, we first convert it into the row echelon form.

Example:

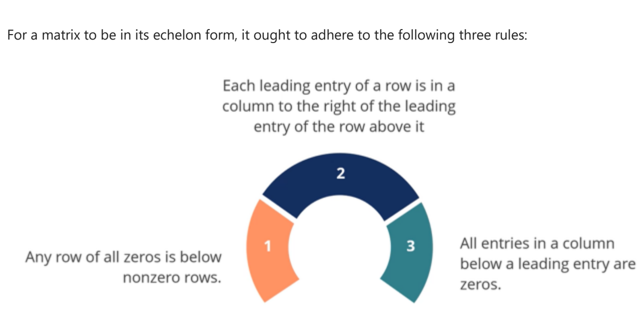
In a matrix, the leading entry of a row is the first nonzero entry in that row.

$$\begin{pmatrix} 3 & 0 & 5 & -3 & 2 \\ 0 & 2 & 1 & 9 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & 6 & 1 & 10 \end{pmatrix}$$

The leading entry in the first row is 3.

Similarly 2, 7, 0 and 5 are the leading entries for the remaining rows.

For a matrix to be in its echelon form, it ought to adhere to the following three rules:



To reduce a matrix to its row echelon form, the following elementary row operations are used:

1.Interchange two rows

2.Multiply a row by a nonzero constant

3.Add a multiple of a row to another row

If the above elementary row operations will be applied to the above matrix, the echelon form can be obtained as below:

$$\begin{pmatrix} 3 & 0 & 5 & -3 & 2 \\ 0 & 2 & 1 & 9 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & 6 & 1 & 10 \end{pmatrix}$$

Stepwise conversion of the matrix to its echelon form are as follows:

Step 1: Multiply the first row by 1/3

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 2 & 1 & 9 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & 6 & 1 & 10 \end{pmatrix}$$

Step 2: Add -5 times the first row to the fifth row

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 2 & 1 & 9 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 43/3 & -4 & 20/3 \end{pmatrix}$$

Step 3: Multiply the second row by half

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 43/3 & -4 & 20/3 \end{pmatrix}$$

Step 4: Add -2 times the second row to the fifth row

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 40/3 & -13 & 20/3 \end{pmatrix}$$

Step 5: Interchange the third row and the fifth row

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 40/3 & -13 & 20/3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \end{pmatrix}$$

Step 6: Multiply the third row by 3/40

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 1 & -39/40 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \end{pmatrix}$$

Step 7: Interchange the fourth and fifth row

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 1 & -39/40 & 1/2 \\ 0 & 0 & 0 & 7 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Step 8: Multiply the fourth row by 1/7

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 1 & -39/40 & 1/2 \\ 0 & 0 & 0 & 1 & 5/7 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

As the number of nonzero rows in the row echelon form is 4, the rank of the given matrix is 4.

$$\begin{pmatrix} 1 & 0 & -5/3 & 1 & 2/3 \\ 0 & 1 & 1/2 & 9/2 & 0 \\ 0 & 0 & 1 & -39/40 & 1/2 \\ 0 & 0 & 0 & 1 & 5/7 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Determinant of a Matrix and Identity Matrix:

The determinant of a matrix is a scalar quantity that is a function of the elements of a matrix. Determinants are defined only for square matrices. These are useful in determining the solution of a system of linear equations.

Let $X = [a_{ij}]$ be an n x n matrix, where n ≥2.

$$\textbf{det X} = a_{11} \det A_{11} - a_{12} \det A_{12} + \ldots\ldots + (-1)^{1+n} a_{1n} \det A_{1n}$$

$$= \sum_{i=1}^{n} (-1)^{1+i} a_{1i} \det A_{1i}$$

If the determinant of a square matrix is 0, then it is not invertible. If the determinant of a matrix is nonzero, the linear system it represents is linearly independent. But when the determinant is zero, its rows are linearly dependent vectors, and its columns are linearly dependent vectors.

Consider the matrices 2X2 and 3X3 matrices:

$$\mathbf{X} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad |\mathbf{X}| = ad - bc$$

$$\mathbf{X} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad |\mathbf{X}| = a \det \begin{pmatrix} e & f \\ h & i \end{pmatrix} - b \det \begin{pmatrix} d & f \\ g & i \end{pmatrix} + c \det \begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

Substitute the expressions for a determinant of a 2×2 matrix in the above equation. So, the output will be shown as below:

|X| = a (ei-fh) –b (di-fg) +c (dh-eg)

Identity Matrix or Operator:

An identity matrix I is a square matrix which when multiplied with a matrix X gives the same result X.

I = identity matrix

XI = IX = X

This is equivalent to the number 1 in the number system.

The diagonal elements of I are all 1 and all its non-diagonal elements are 0.

Two-dimensional identity matrix

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

Three-dimensional identity matrix

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

Inverse of a Matrix, Eigenvalues, and Eigenvectors:

Inverse of a matrix

If X is an $n \times n$ matrix, an inverse of X is an $n \times n$ matrix X^1 such that

$XX^1 = X^1X = I$

I is the $n \times n$ identity matrix. If an X^1 exists for X, then X is described as invertible.

Consider the following example:

X is a 2x2 matrix

$$\mathbf{X} = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix} \qquad \mathbf{X^1} = \begin{pmatrix} 3 & -5 \\ -1 & 2 \end{pmatrix}$$

$$\mathbf{XX^1} = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} 3 & -5 \\ -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{X^1X} = \begin{pmatrix} 3 & -5 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Eigenvalues and Eigenvectors

Eigenvectors are vectors that are fixed in direction under a given linear transformation. The scaling factor of these Eigenvectors is called the Eigenvalue.

Suppose X is an $n \times n$ matrix. When we multiply X with a new vector A, it does two things to the vector A:

1. It scales the vector
2. It rotates the vector

However, when X acts on a certain set of vectors, it results only in scaling the vector and not in any change in the direction of the vector. Those particular vectors are called Eigenvectors and the amount by which these vectors stretch or compress is called the corresponding Eigenvalue.

Let X be an $n \times n$ matrix. A scalar λ is called an Eigenvalue of X, if there is a nonzero vector A such that XA = λA. Such a vector A is called an eigenvector of X corresponding to λ .

Consider the following example:

Suppose X is a matrix and A is is an Eigen vector of X

$$\mathbf{X} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\mathbf{XA} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} = 4\begin{pmatrix} 1 \\ 1 \end{pmatrix} = 4A$$

Here, A is an Eigen vector of X corresponding to Eigen value 4.

Calculus in Linear Algebra:

Calculus is the branch of mathematics that studies continuous changes in quantities. It commonly measures quantities such as slopes of curves or objects.

Calculus can be broadly divided into two parts

Differential Calculus

Integral Calculus

While the former concerns instantaneous rates of change, and the slopes of curves, the latter explores the accumulation of quantities and areas under or between curves.

It is necessary for developing an intuition for machine learning algorithms.

Differential Calculus

Differential calculus are applied in important machine learning algorithms like Gradient Descent.Gradient Descent is vital in the backpropagation of Neural Networks. It measures how the output of a function changes when the input changes in small amounts.

Applications of differential calculus in machine learning algorithms

Finding the maximal margin in support vector machines

Discovering the maximum in the expectation-maximization algorithm

Integral Calculus

Integral calculus is commonly used to determine the probability of events. For example, it helps us find the posterior in a Bayesian model or bound the error in a sequential decision as per the Neyman-Pearson Lemma.