Determinant and trace

In [1]:
```
import numpy as np
# creating a 2X2 Numpy matrix
n_array = np.array([[50, 29], [30, 44]])

# Displaying the Matrix
print("Numpy Matrix is:")
print(n_array)

# calculating the determinant of matrix
det = np.linalg.det(n_array)

print("\nDeterminant of given 2X2 matrix:")
print(int(det))
```

```
Numpy Matrix is:
[[50 29]
 [30 44]]

Determinant of given 2X2 matrix:
1330
```

In [2]:
```
# importing Numpy package
import numpy as np

# creating a 3X3 Numpy matrix
n_array = np.array([[55, 25, 15],
                    [30, 44, 2],
                    [11, 45, 77]])

# Displaying the Matrix
print("Numpy Matrix is:")
print(n_array)

# calculating the determinant of matrix
det = np.linalg.det(n_array)

print("\nDeterminant of given 3X3 square matrix:")
print(int(det))
```

```
Numpy Matrix is:
[[55 25 15]
 [30 44  2]
 [11 45 77]]

Determinant of given 3X3 square matrix:
137180
```

In [3]:
```
# creating a 5X5 Numpy matrix
n_array = np.array([[5, 2, 1, 4, 6],
                    [9, 4, 2, 5, 2],
                    [11, 5, 7, 3, 9],
                    [5, 6, 6, 7, 2],
                    [7, 5, 9, 3, 3]])

# Displaying the Matrix
print("Numpy Matrix is:")
print(n_array)

# calculating the determinant of matrix
det = np.linalg.det(n_array)

print("\nDeterminant of given 5X5 square matrix:")
print(int(det))
```

```
Numpy Matrix is:
[[ 5  2  1  4  6]
 [ 9  4  2  5  2]
 [11  5  7  3  9]
 [ 5  6  6  7  2]
 [ 7  5  9  3  3]]

Determinant of given 5X5 square matrix:
-2003
```

In [4]:
```
# Let's create a square matrix (NxN matrix)
mx = np.array([[1,1,1],[0,1,2],[1,5,3]])
mx

# Let's get trace(sum of diagonal elements)
mx.trace()
```

Out[4]: 5

Eigenvalues and eigenvectors

Definition

Let A be a square matrix. A non-zero vector $\mathbf{v}$ is an eigenvector for A with eigenvalue λ if

$$A\mathbf{v} = \lambda\mathbf{v}$$

Rearranging the equation, we see that $\mathbf{v}$ is a solution of the homogeneous system of equations

$$(A - \lambda I)\,\mathbf{v} = \mathbf{0}$$

where I is the identity matrix of size n. Non-trivial solutions exist only if the matrix $A - \lambda I$ is singular which means $\det(A - \lambda I) = 0$. Therefore eigenvalues of A are roots of the characteristic polynomial

$$p(\lambda) = \det(A - \lambda I)$$

In [7]:
```
# create numpy 2d-array
m = np.array([[1, 6, 3],
              [0, -2, 0],
              [3, 6, 1]])

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)
```

```
Printing the Original square array:
 [[ 1  6  3]
 [ 0 -2  0]
 [ 3  6  1]]
Printing the Eigen values of the given square array:
 [ 4. -2. -2.]
Printing Right eigenvectors of the given square array:
 [[ 0.70710678 -0.70710678 -0.14644661]
 [ 0.          0.          0.5       ]
 [ 0.70710678  0.70710678 -0.85355339]]
```

According to results above we could see Eigenvalues of A are W1= 4 , W2=-2 and W3=-2 and Eigenvectors are V1=[0.70710678,0,0.70710678],V2=[-0.70710678,0,0.70710678],and V3=[-0.14644661,0.5,-0.85355339]

In [35]:
```
# Test Results for W1 and V1
m = np.array([[1, 6, 3],
              [0, -2, 0],
              [3, 6, 1]])
W1=4
V1=np.array([[0.70710678],[0],[0.70710678]])
a=np.dot(m,V1)
b=np.dot(W1,V1)
print(a)
print(b)
```

```
[[2.82842712]
 [0.        ]
 [2.82842712]]
[[2.82842712]
 [0.        ]
 [2.82842712]]
```

Exercise

Determin Which of following vectors are eigenvectors of A =np.array([[2, -3, -1], [1, -2, -1], [1, -3, 0]])

v1=np.array([[3],[1],[0]]) v2=np.array([[-1],[2],[-1]]) v3=np.array([[1],[1],[1]])

In [22]:
```
A =np.array([[2, -3, -1],
             [1, -2, -1],
             [1, -3, 0]])
v1=np.array([[3],
             [1],
             [0]])
```

In [24]:
```
a=A@v1
a

# so v1 is an eigenvector with eigenvalue w1=1
```

Out[24]:
```
array([[3],
       [1],
       [0]])
```

In [28]:
```
A =np.array([[2, -3, -1],
             [1, -2, -1],
             [1, -3, 0]])
v2=np.array([[-1],
             [2],
             [-1]])
```

In [29]:
```
b=A@v2
b

# So A@v2 is not a scalar multiple of v2,so it is not an eigenvector
```

Out[29]:
```
array([[-7],
       [-4],
       [-7]])
```

In [32]:
```
A =np.array([[2, -3, -1],
             [1, -2, -1],
             [1, -3, 0]])
v3=np.array([[1],
             [1],
             [1]])

c=A@v3
c
#so v3 is an eigenvector with eigenvalue w2=-2
```

Out[32]:
```
array([[-2],
       [-2],
       [-2]])
```

Eigenspace

What is an Eigenspace?

For a square matrix A, the **eigenspace** of A is the span of eigenvectors associated with an eigenvalue, λ.

The eigenspace can be defined mathematically as follows:

$$E_\lambda(A) = N(A - \lambda I)$$

Where:

- A is a square matrix of size n
- the scalar λ is an eigenvalue associated with some eigenvector, v
- $N(A - \lambda I)$ is the null space of $A - \lambda I$.

A Numerical Example:

Let's consider a simple example with a matrix A:

$$A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

Step 1: Obtain eigenvalues using the characteristic polynomial given by $det(A - \lambda I) = 0$.

$$\begin{aligned} \det(A - \lambda I_n) &= 0 \\ \det\left(\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) &= 0 \\ \det\left(\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) &= 0 \\ \begin{vmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{vmatrix} &= 0 \\ (2-\lambda)(1-\lambda) - 2 \cdot 3 &= 0 \\ \lambda^2 - 3\lambda - 4 &= 0 \\ (\lambda - 4)(\lambda + 1) &= 0 \\ \lambda_1 = 4,\ \lambda_2 &= -1 \end{aligned}$$

The roots of the characteristic polynomial give eigenvalues $\lambda_1 = 4$ and $\lambda_2 = -1$

Step 2: The associated eigenvectors can now be found by substituting eigenvalues λ into $(A - \lambda I)$. Eigenvectors that correspond to these eigenvalues are calculated by looking at vectors $\vec{v}$ such that

$$\begin{bmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{bmatrix} \vec{v} = 0$$

Eigenvector for $\lambda_1 = 4$ is found by solving the following homogeneous system of equations:

$$\begin{bmatrix} 2-4 & 3 \\ 2 & 1-4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -2 & 3 \\ 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

After solving the above homogeneous system of equations, the first eigenvector is obtained:

$$v_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Similarly, eigenvector for $\lambda_2 = -1$ is found.

$$\begin{bmatrix} 2+1 & 3 \\ 2 & 1+1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

After solving the above homogeneous system of equations, the second eigenvector is obtained.

$$v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Step 3: After obtaining eigenvalues and eigenvectors, the eigenspace can be defined. The eigenvectors form the basis and define the eigenspace of matrix A with associated eigenvalue λ. In this way, the solution space can be defined as:

$$E_4(A) = span(\begin{bmatrix} 3 \\ 2 \end{bmatrix}), \qquad E_{-1}(A) = span(\begin{bmatrix} 1 \\ -1 \end{bmatrix})$$

Recall that the **span** of a single vector is an infinite line through the vector. Thus, $E_4(A)$ is the line through the origin and the point (3, 2), and $E_{-1}(A)$ is the line through the origin and the point (1, -1).

In [37]:
```
import numpy as np
import numpy.linalg as la
A = np.array([[2, 3],[2, 1]])
eig_vals, eig_vecs = la.eig(A)

# The eigenvalues are:
lambda_1 = np.real(eig_vals[0]) # First eigenvalue
lambda_2 = np.real(eig_vals[1]) # Second eigenvalue
print('Eigenvalues: \n','lambda_1 = ', lambda_1, '\n','lambda_2 =', lambda_2, '\n')

# The corresponding eigenvectors are:
v_1 = eig_vecs[:, 0] # First eigenvector
v_2 = eig_vecs[:, 1] # Second eigenvector
print('Eigenvectors: \n', 'v_1 =', v_1,'\n', 'v_2 =', v_2)
```

```
Eigenvalues:
 lambda_1 =  4.0
 lambda_2 = -1.0

Eigenvectors:
 v_1 = [0.83205029 0.5547002 ]
 v_2 = [-0.70710678  0.70710678]
```

In [ ]:
```
## The eigenvectors form the basis and define the eigenspace of matrix with associated eigenvalue .
```

Eigendecomposition and Diagonalization

Similarity

If $A = PBP^{-1}$, we say A is *similar* to B, decomposing A into PBP^{-1} is also called a *similarity transformation*.

If $n \times n$ matrices A and B are similar, they have the *same eigenvalues*.

The *diagnoalization*, which we will explain below, is a process of finding similar matrices.

Diagonalizable Matrix

Let A be an $n \times n$ matrix. If there exists an $n \times n$ invertible matrix P and a diagonal matrix D, such that

$$A = PDP^{-1}$$

then matrix A is called a *diagonalizable matrix*.

And further, the columns of P are linearly independent eigenvectors of A, and its corresponding eigenvalues are on the diagonal of D. In other words, A is diagonalizable if and only if the dimension of eigenspace basis is n.

Let's show why this equation holds.

Let

$$P = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$
$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

where $v_i, i \in (1, 2, \ldots n)$ is an eigenvector of A, $\lambda_i, i \in (1, 2, \ldots n)$ is an eigenvalue of A.

$$AP = A \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} Av_1 & Av_2 & \cdots & Av_n \end{bmatrix}$$

$$PD = P \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} = \begin{bmatrix} \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_n v_n \end{bmatrix}$$

We know that $Av_i = \lambda_i v_i$, i.e.

$$AP = PD$$

Since P has all independent eigenvectors, then

$$A = PDP^{-1}$$

Strictly speaking, if A is symmetric, i.e. $A = A^T$, what we have just shown is called **Spectral decomposition**, the similar matrix D holds all the eigenvalues on its diagonal. And P is orthogonal matrix, which means any of of its two columns are perpendicular. Therefore it could be rewritten as

$$A = PDP^T$$

Diagonalizing a Matrix

Consider a matrix

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

We seek to diagonalize the matrix A.

Following these steps:

1. Compute the eigenvalues of A
2. Compute the eigenvectors of A
3. Construct P.
4. Construct D from the corresponding columns of P.

In [26]:
```
pip install sympy
```

```
Requirement already satisfied: sympy in d:\anacondasucks\lib\site-packages (1.9)
Requirement already satisfied: mpmath>=0.19 in d:\anacondasucks\lib\site-packages (from sympy) (1.2.1)
Note: you may need to restart the kernel to use updated packages.
```

In [27]:
```
import numpy as np
import sympy as sy
A = sy.Matrix([[1,3,3], [-3, -5, -3], [3,3,1]])
eig = sy.matrices.matrices.MatrixEigen.eigenvects(A)
eig
```

Out[27]:
```
[(-2,
  2,
  [Matrix([
   [-1],
   [ 1],
   [ 0]]),
   Matrix([
   [-1],
   [ 0],
   [ 1]])]),
 (1,
  1,
  [Matrix([
   [ 1],
   [-1],
   [ 1]])])]
```

In [28]:
```
# Construct P
P = sy.zeros(3, 3)
P[:, 0] = eig[0][2][0]
P[:, 1] = eig[0][2][1]
P[:, 2] = eig[1][2][0]
P
```

Out[28]:

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

In [29]:
```
# Construct D
D = sy.diag(eig[0][0], eig[0][0], eig[1][0])
D
```

Out[29]:

$$\begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can verify if $PDP^{-1} = A$ holds:

In [30]:
```
P * D * P.inv() == A
```

Out[30]:
```
True
```

In [31]:
```
# In Sympy, there is diagonalize method in SymPy.

P, D = A.diagonalize()
```

In [32]:
```
P
```

Out[32]:

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

In [33]:
```
D
```

Out[33]:

$$\begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In [34]:
```
# sometimes, you could directly test if a matrix is diagonalizable.
A.is_diagonalizable()
```

Out[34]:
```
True
```

Cholesky Decomposition

Recall that a square matrix A is positive definite if

$$u^T Au > 0$$

for any non-zero n-dimensional vector u,

and a symmetric, positive-definite matrix A is a positive-definite matrix such that

$$A = A^T$$

Let A be a symmetric, positive-definite matrix. There is a unique decomposition such that

$$A = LL^T$$

where L is lower-triangular with positive diagonal elements and L^T is its transpose. This decomposition is known as the Cholesky decompostion, and L may be interpreted as the 'square root' of the matrix A.

In [ ]:
```
pip install scipy
```

In [1]:
```
import numpy as np
import scipy.linalg as la

A = np.array([[1,3,5],[3,13,23],[5,23,42]])
L = la.cholesky(A)
print(np.dot(L.T, L))

print(L)
print(A)
```

```
[[ 1.  3.  5.]
 [ 3. 13. 23.]
 [ 5. 23. 42.]]
[[1. 3. 5.]
 [0. 2. 4.]
 [0. 0. 1.]]
[[ 1  3  5]
 [ 3 13 23]
 [ 5 23 42]]
```

Singular value decomposition

Singular Value Decomposition (SVD) is a powerful technique widely used in solving dimensionality reduction problems. This algorithm works with a data matrix of the form, `m x n`, i.e., a rectangular matrix.

The idea behind the SVD is that a rectangular matrix can be broken down into a product of three other matrices that are easy to work with. This decomposition is of the form as the one shown in the formula below:

$$A = U \Sigma V^T$$

Where:

- A is our m x n data matrix we are interested in decomposing.
- U is an m x m orthogonal matrix whose bases are orthonormal.
- Σ is an m x n diagonal matrix.
- V^T is the transpose of an orthogonal matrix.

In [53]:

```
pip install scipy
```

```
Requirement already satisfied: scipy in d:\anacondasucks\lib\site-packages (1.7.1)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in d:\anacondasucks\lib\site-packages (from scipy) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [54]:

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

"""
Singular Value Decomposition
"""
# define a matrix
X = np.array([[3, 3, 2], [2, 3, -2]])
print(X)
# perform SVD
U, singular, V_transpose = svd(X)
# print different components
print("U: ", U)
print("Singular array", singular)
print("V^{T}", V_transpose)
```

```
[[ 3  3  2]
 [ 2  3 -2]]
U:  [[ 0.7815437 -0.6238505]
 [ 0.6238505  0.7815437]]
Singular array [5.54801894 2.86696457]
V^{T} [[ 0.64749817  0.7599438   0.05684667]
 [-0.10759258  0.16501062 -0.9804057 ]
 [-0.75443354  0.62869461  0.18860838]]
```

In [ ]: