Assignment 1 Basic Linear Algebra in Python

Use Numpy library functions and methods to complete given tasks.

Given: $x \in R^n$; $A \in R^{n \times n}$; $B \in R^{n \times n}$

Read dimension of all variables and their values from keyboard.

Find and Display:

- A^T and B^T
- A.B
- Verify $(AB)^T = B^T . A^T$
- |A | and | B |
- Check for singularity of A and B
- Find A^{-1} and B^{-1} (if possible) (Read again values of A and B if inverses are not possible and do it till inverses are available)
- Verify $AA^{-1} = I$ and $A^{-1}A = I$
- Trace of A and B, A^{-1} and B^{-1}, A^T and B^T
- Verify trace (AB) = trace (BA)
- Verify $trace(A^TB) = trace(AB^T) = trace(BA^T) =trace(B^TA)$
- Eigen values and Eigen vectors of A, B, AB, A^T ,B^T, A^{-1}, B^{-1}
- Verify trace and determinant property related to Eigen Values for both Matrix A and B
- Check definiteness of Matrices A, B, AB, A^T ,B^T, A^{-1}, B^{-1}
- $y = Ax; y \in R^n$
- Retrieve x from y
- Inner product of x and y and verify they are orthogonal or not
- L_2-Norm of x and y
- Normalize x and y
- Verify CS inequality for x and y
- Verify $x^Ty = y^Tx$
- Verify $y^TAx = x^TA^Ty$

Import Numpy library

In [1]:
```
import numpy as np
```

Take Input and Verify Singularity of Matrix

In [2]:
```
n = int(input("Enter dimension (n) for matrix A and B (n x  n) and vector x (n x 1): ")

x = list(map(int,input(f"Enter Vector x of size {n}. x 1 in a single line row wise elem
x = np.matrix(x).reshape(n,1)
print("\nVector x:\n", x)

while (True):
    A = list(map(int,input(f"Enter matrix A of size {n}. x {n}. in a single line row wi
    A = np.array(A).reshape(n,n)
    print("\nMatrix A:\n", A)
    # |A| and singularity check
    det_A = np.linalg.det(A)
    print(f"\nDeterminant of A: {det_A}")
    if np.isclose(det_A, 0):
       print("\nA is singular, Reading again.")
       continue
    else:
       print("\nA is non-singular")
       break

while (True):
     B = list(map(int,input(f"Enter matrix B of size {n}. x {n}. in a single line row w
     B = np.array(B).reshape(n,n)
     print("\nMatrix B:\n", B)
     # |B| and singularity check
     det_B = np.linalg.det(B)
     print(f"\nDeterminant of B: {det_B}")
     if np.isclose(det_B, 0):
        print("\nB is singular, Reading again.")
        continue
     else:
        print("\nB is non-singular")
        break
```

```
Vector x:
 [[1]
 [2]]
Matrix A:
 [[1 2]
 [3 4]]

Determinant of A: -2.0000000000000004

A is non-singular
Matrix B:
 [[5 6]
 [7 8]]

Determinant of B: -2.000000000000005

B is non-singular
```

Transpose of Matrix

In [3]:
```
# Transpose
print("\nTranspose of A:\n", A.T)
print("\nTranspose of B:\n", B.T)
```

```
Transpose of A:
 [[1 3]
 [2 4]]

Transpose of B:
 [[5 7]
 [6 8]]
```

Matrix Multiplication

In [4]:
```
# Matrix multiplication
AB = A @ B
print("\nMatrix AB:\n", AB)
print("\nMatrix AB using numpy:\n", np.dot(A,B))
```

```
Matrix AB:
 [[19 22]
 [43 50]]

Matrix AB using numpy:
 [[19 22]
 [43 50]]
```

Verify $(AB)^T = B^T . A^T$

In [5]:
```
print("\nVerify (AB)^T = B^T A^T:", np.allclose(AB.T, B.T @ A.T))
```

```
Verify (AB)^T = B^T A^T: True
```

Finding Inverse Of Matrix

In [6]:
```
# Inverse
A_inv = np.linalg.inv(A)
print("\nInverse of A:\n", A_inv)
B_inv = np.linalg.inv(B)
print("\nInverse of B:\n", B_inv)
```

```
Inverse of A:
 [[-2.   1. ]
 [ 1.5 -0.5]]

Inverse of B:
 [[-4.   3. ]
 [ 3.5 -2.5]]
```

Verify $AA^{-1} = I$ and $A^{-1}A = I$

In [7]:
```
print("Verify AA^-1 = I:", np.allclose(A @ A_inv, np.eye(n)))
print("Verify A^-1A = I:", np.allclose(A_inv @ A, np.eye(n)))
```

```
Verify AA^-1 = I: True
Verify A^-1A = I: True
```

Finding Trace of Matrices

In [8]:
```
# Trace
print(f"\nTrace of A: {np.trace(A)}")
print(f"Trace of B: {np.trace(B)}")
print(f"Trace of A^-1: {np.trace(A_inv)}")
print(f"Trace of B^-1: {np.trace(B_inv)}")
print(f"Trace of A^T: {np.trace(A.T)}")
print(f"Trace of B^T: {np.trace(B.T)}")
```

```
Trace of A: 5
Trace of B: 13
Trace of A^-1: -2.4999999999999996
Trace of B^-1: -6.499999999999982
Trace of A^T: 5
Trace of B^T: 13
```

Trace Verification

In [9]:
```
# Trace verification
print(f"\nVerify trace(AB) = trace(BA): {np.trace(A @ B) == np.trace(B @ A)}")
print("\nVerify trace(A^T.B) = trace(A.B^T) = trace(B.A^T) = trace(B^T.A):",
      np.trace(A.T @ B) == np.trace(A @ B.T) == np.trace(B @ A.T) == np.trace(B.T @ A))
```

```
Verify trace(AB) = trace(BA): True

Verify trace(A^T.B) = trace(A.B^T) = trace(B.A^T) = trace(B^T.A): True
```

Finding Eigenvalues and Eigenvectors

In [10]:
```
np.linalg.eig(A)
```

Out[10]:
```
EigResult(eigenvalues=array([-0.37228132,  5.37228132]), eigenvectors=array([[-0.824564
84, -0.41597356],
       [ 0.56576746, -0.90937671]]))
```

In [11]:
```
# Eigenvalues and Eigenvectors
eigvals_A, eigvecs_A = np.linalg.eig(A)
eigvals_B, eigvecs_B = np.linalg.eig(B)
print("\nEigenvalues of A:\n", eigvals_A)
print("Eigenvectors of A:\n", eigvecs_A)
print("\nEigenvalues of B:\n", eigvals_B)
print("Eigenvectors of B:\n", eigvecs_B)
eigvals_AT, eigvecs_AT = np.linalg.eig(A.T)
eigvals_BT, eigvecs_BT = np.linalg.eig(B.T)
print("\nEigenvalues of AT:\n", eigvals_AT)
print("Eigenvectors of AT:\n", eigvecs_AT)
print("\nEigenvalues of BT:\n", eigvals_BT)
print("Eigenvectors of BT:\n", eigvecs_BT)
eigvals_AB, eigvecs_AB = np.linalg.eig(AB)
print("\nEigenvalues of AB:\n", eigvals_AB)
print("Eigenvectors of AB:\n", eigvecs_AB)
eigvals_A_1, eigvecs_A_1 = np.linalg.eig(A_inv)
eigvals_B_1, eigvecs_B_1 = np.linalg.eig(B_inv)
print("\nEigenvalues of A^-1:\n", eigvals_A_1)
print("Eigenvectors of A^-1:\n", eigvecs_A_1)
print("\nEigenvalues of B^-1:\n", eigvals_B_1)
print("Eigenvectors of B^-1:\n", eigvecs_B_1)
```

```
Eigenvalues of A:
 [-0.37228132  5.37228132]
Eigenvectors of A:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]

Eigenvalues of B:
 [-0.15206735 13.15206735]
Eigenvectors of B:
 [[-0.75868086 -0.59276441]
 [ 0.65146248 -0.80537591]]

Eigenvalues of AT:
 [-0.37228132  5.37228132]
Eigenvectors of AT:
 [[-0.90937671 -0.56576746]
 [ 0.41597356 -0.82456484]]

Eigenvalues of BT:
 [-0.15206735 13.15206735]
Eigenvectors of BT:
 [[-0.80537591 -0.65146248]
 [ 0.59276441 -0.75868086]]

Eigenvalues of AB:
 [5.80198014e-02 6.89419802e+01]
Eigenvectors of AB:
 [[-0.75781077 -0.40313049]
 [ 0.65247439 -0.91514251]]

Eigenvalues of A^-1:
 [-2.68614066  0.18614066]
Eigenvectors of A^-1:
 [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]

Eigenvalues of B^-1:
 [-6.57603367  0.07603367]
Eigenvectors of B^-1:
 [[-0.75868086 -0.59276441]
 [ 0.65146248 -0.80537591]]
```

Verify trace and determinant properties of Eigen values for A and B

In [12]:
```
print("\nVerify det(A) = Multiplication of eigvals_A:", np.isclose(det_A, np.prod(eigva
print("\nVerify det(B) = Multiplication of eigvals_B:", np.isclose(det_B, np.prod(eigva
print("\nVerify trace(A) = Summation of eigvals_A:", np.isclose(np.trace(A), np.sum(eig
print("\nVerify trace(B) = Summation of eigvals_B:", np.isclose(np.trace(B), np.sum(eig
```

```
Verify det(A) = Multiplication of eigvals_A: True

Verify det(B) = Multiplication of eigvals_B: True

Verify trace(A) = Summation of eigvals_A: True

Verify trace(B) = Summation of eigvals_B: True
```

Definiteness Test for A and B (In general not symmetic)

In [13]:
```
Asym = A+A.T
Bsym = B+B.T
eigvals_Asym, eigvecs_Asym = np.linalg.eig(Asym)
```

```
eigvals_Bsym, eigvecs_Bsym = np.linalg.eig(Bsym)
if(np.all(eigvals_Asym > np.finfo(np.float32).eps)):
    print("\nA is Positive definite")
elif(np.all(eigvals_Asym >= 0)):
    print("\nA is Positive semi-definite")
elif(np.all(eigvals_Asym <0)):
    print("\nA is Negative semi-definite")
elif(np.all(eigvals_Asym <= 0)):
    print("\nA is Negative semi-definite")
else:
    print("\nA is indefinite")

if(np.all(eigvals_Bsym > 0)):
    print("\nB is Positive definite")
elif(np.all(eigvals_Bsym >= 0)):
    print("\nB is Positive semi-definite")
elif(np.all(eigvals_Bsym <0)):
    print("\nB is Negative semi-definite")
elif(np.all(eigvals_Bsym <= 0)):
    print("\nB is Negative semi-definite")
else:
    print("\nB is indefinite")
```

```
A is indefinite

B is indefinite
```

Vector Operations

In [14]:
```
A
```

Out[14]:
```
array([[1, 2],
       [3, 4]])
```

In [15]:
```
x
```

Out[15]:
```
matrix([[1],
        [2]])
```

$y = Ax; y \in R^n$

In [16]:
```
# Vector operations
y = A @ x
print("\ny = Ax:\n", y)
```

```
y = Ax:
 [[ 5]
 [11]]
```

Retrieved x from y

In [17]:
```
np.linalg.solve(A,y)
```

Out[17]:
```
matrix([[1.],
        [2.]])
```

In [18]:
```
x_retrieved = A_inv @ y
print("Retrieved x from y:\n", x_retrieved)
```

```
Retrieved x from y:
 [[1.]
 [2.]]
```

Inner product, norm, and orthogonality

In [19]:
```
inner_product = x.T @ y
print(f"\nInner product of x and y: {inner_product}")
print("x and y are orthogonal:" if np.isclose(inner_product, 0) else "x and y are not o
print(f"\nNorm of x: {np.linalg.norm(x)}")
print(f"Norm of y: {np.linalg.norm(y)}") #L2-norm by default
```

```
Inner product of x and y: [[27]]
x and y are not orthogonal.

Norm of x: 2.23606797749979
Norm of y: 12.083045973594572
```

L_1-Norm of x and y

In [20]:
```
print(f"\nNorm of x: {np.linalg.norm(x,ord=1)}")
print(f"Norm of y: {np.linalg.norm(y,ord=1)}")
```

```
Norm of x: 3.0
Norm of y: 16.0
```

Normalizing Vector x & y

In [21]:
```
# Normalize
x_normalized = x / np.linalg.norm(x)
y_normalized = y / np.linalg.norm(y)
print("\nNormalized x:\n", x_normalized)
print("Normalized y:\n", y_normalized)
```

```
Normalized x:
 [[0.4472136 ]
 [0.89442719]]
Normalized y:
 [[0.41380294]
 [0.91036648]]
```

Verifying CS(Cauchy-Schwarz\Cauchy-Bunyakovsky-Schwarz inequality) inequality i.e. |⟨a,b⟩|≤‖a‖·‖b‖

In [22]:
```
print("\nVerify CS inequality for x and y:",
inner_product <= np.linalg.norm(x) * np.linalg.norm(y))
if abs(inner_product) <= np.linalg.norm(x) * np.linalg.norm(y) :
    print("CS inequality (|⟨a,b⟩|≤‖a‖·‖b‖) verified")
else :
    print("CS inequality not verified")
```

```
Verify CS inequality for x and y: [[ True]]
CS inequality (|⟨a,b⟩|≤‖a‖·‖b‖) verified
```

Additional verifications

In [23]:
```
print("\nVerify x^T y = y^T x:", np.isclose(inner_product, y.T @ x))
print("\nVerify y^T A x = x^T A^T y:",
np.isclose((y.T @ A) @ x, (x.T @ A.T) @ y))
```

```
Verify x^T y = y^T x: [[ True]]

Verify y^T A x = x^T A^T y: [[ True]]
```