Shweta_Shinde_Math_Assignment7

November 2, 2024

1 Shweta Ajay Shinde 017548687

MSDA, SJSU , Data 220- Math Method for DA

Make a python code for:

1) Calculating the inverse matrix of A (1 pts)

[2]:
```
import numpy as np
A = np.array([
    [4, -2, 2, 1],
    [1, 1, 0, 1],
    [-2, 1, 3, -1],
    [1, 3, -1, 2]
            ])
# Calculate the inverse of matrix A
A_inv = np.linalg.inv(A)

# Print the inverse matrix
print("The inverse of matrix A is:")
print(A_inv)
```

```
The inverse of matrix A is:
[[ 5.00000000e+00 -3.00000000e+01  1.00000000e+00  1.30000000e+01]
 [ 4.00000000e+00 -2.50000000e+01  1.00000000e+00  1.10000000e+01]
 [-1.00000000e+00  7.00000000e+00 -7.77156117e-16 -3.00000000e+00]
 [-9.00000000e+00  5.60000000e+01 -2.00000000e+00 -2.40000000e+01]]
```

2) Calculating Eigenvalues and Eigenvectors for the 4x4 Matrix (2 pts).

[3]:
```
# Calculate the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

# Print the results
print("Eigenvalues:")
print(eigenvalues)

print("\nEigenvectors:")
print(eigenvectors)
```

```
Eigenvalues:
[-0.02231724+0.j          3.6104532 +1.72034086j  3.6104532 -1.72034086j
  2.80141085+0.j        ]

Eigenvectors:
[[-0.43676159+0.j          0.57598271+0.j          0.57598271-0.j
   0.26748859+0.j        ]
 [-0.36900111+0.j          0.00962337-0.2105225j   0.00962337+0.2105225j
  -0.34893941+0.j        ]
 [ 0.10239683+0.j         -0.00821729+0.55142262j -0.00821729-0.55142262j
  -0.06120796+0.j        ]
 [ 0.81399778+0.j         -0.18869091-0.53300366j -0.18869091+0.53300366j
  -0.89607183+0.j        ]]
```

3) After having the Eigenvalues and Eigenvectors,

   - Constructing the eigen matrix P from eigenvectors: P=eigenvectors
   - Constructing the diagonal matrix D from eigenvalues: D=np.diag(eiganvalues)
   - Calculating the inverse of the eigen matrix: P_inv = np.linalg.inv(P)

Verify if the diagonalized matrix `A_diagonalized` is identical to the original matrix `A` (2 pts). A_diagnalized = P @ D @ P_inv

[11]:
```
# Calculate eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

# Construct the eigen matrix P from eigenvectors
P = eigenvectors

# Construct the diagonal matrix D from eigenvalues
D = np.diag(eigenvalues)

# Calculate the inverse of the eigen matrix
P_inv = np.linalg.inv(P)

# Calculate the diagonalized matrix
A_diagonalized = P @ D @ P_inv

# Print the results
print("\nEigenvectors Matrix P:")
print(P)

print("\nDiagonal Matrix D:")
print(D)

print("\nOriginal Matrix A:")
print(A)
```

```python
print("\nDiagonalized Matrix A_diagonalized:")
print(A_diagonalized)

# Check if A_diagonalized is approximately equal to A
if np.allclose(A, A_diagonalized):
    print("\nA_diagonalized is identical to the original matrix A.")
else:
    print("\nA_diagonalized is NOT identical to the original matrix A.")
```

```
Eigenvectors Matrix P:
[[-0.43676159+0.j          0.57598271+0.j          0.57598271-0.j
   0.26748859+0.j        ]
 [-0.36900111+0.j          0.00962337-0.2105225j   0.00962337+0.2105225j
  -0.34893941+0.j        ]
 [ 0.10239683+0.j         -0.00821729+0.55142262j -0.00821729-0.55142262j
  -0.06120796+0.j        ]
 [ 0.81399778+0.j         -0.18869091-0.53300366j -0.18869091+0.53300366j
  -0.89607183+0.j        ]]

Diagonal Matrix D:
[[-0.02231724+0.j          0.        +0.j          0.        +0.j
   0.        +0.j        ]
 [ 0.        +0.j          3.6104532 +1.72034086j  0.        +0.j
   0.        +0.j        ]
 [ 0.        +0.j          0.        +0.j          3.6104532 -1.72034086j
   0.        +0.j        ]
 [ 0.        +0.j          0.        +0.j          0.        +0.j
   2.80141085+0.j        ]]

Original Matrix A:
[[ 4 -2  2  1]
 [ 1  1  0  1]
 [-2  1  3 -1]
 [ 1  3 -1  2]]

Diagonalized Matrix A_diagonalized:
[[ 4.00000000e+00+4.59988659e-16j -2.00000000e+00-1.25679570e-16j
   2.00000000e+00-5.55060892e-18j  1.00000000e+00-5.87274714e-17j]
 [ 1.00000000e+00-1.45688883e-16j  1.00000000e+00+2.06157796e-17j
   2.22044605e-16-4.41427991e-17j  1.00000000e+00-4.48739398e-17j]
 [-2.00000000e+00+2.70950622e-16j  1.00000000e+00-2.04476175e-16j
   3.00000000e+00-1.22420428e-16j -1.00000000e+00+6.26632910e-17j]
 [ 1.00000000e+00-4.16733564e-16j  3.00000000e+00+1.63580536e-16j
  -1.00000000e+00-2.23091523e-16j  2.00000000e+00-5.52727360e-17j]]

A_diagonalized is identical to the original matrix A.
```

4) Explain why this result (a diagonalized matrix is identical to the original matrix) is important in linear algebra (1 pts).

The result that a diagonalized matrix is identical to the original matrix is important because:

- **Easier Calculations**: Diagonal matrices are simpler to work with. For example, it's much easier to calculate powers of a diagonal matrix. Additionally, multiplying diagonal matrices is straightforward since you only multiply the diagonal elements.For example, raising a diagonal matrix to a power simply involves raising each diagonal entry to that power.
- **Matrix Insights**: Diagonalization helps us see the key characteristics of a matrix, like how it stretches or shrinks space, making it easier to understand its effects. It also reveals whether a transformation preserves angles and lengths.
- **Solving Problems**: In systems of equations, diagonalization can make finding solutions quicker and simpler. It allows for easy back substitution in linear systems.
- **Stability Analysis**: In engineering and control systems, it helps determine if a system will behave steadily or go out of control. It also indicates how sensitive a system is to changes in input.
- **Data Analysis**: In statistics, diagonalization is used in techniques like PCA to simplify data while keeping important information. It aids in identifying the most significant features in datasets.

Overall, diagonalization makes complex problems easier to handle and provides valuable insights across many fields.

5) Make a python code using SVD example to get the same A (AR) (4 pts)

- Decide your own range for the sigma

[8]:
```python
# Perform Singular Value Decomposition
U, sigma, VT = np.linalg.svd(A)

# Choose a range for sigma
Sigma = np.zeros(A.shape)
for i in range(len(sigma)):
    Sigma[i, i] = sigma[i]

# Reconstruct the original matrix A from U, Sigma, and VT
A_reconstructed = U @ Sigma @ VT

# Print the results
print("Original Matrix A:")
print(A)

print("\nReconstructed Matrix A_reconstructed:")
print(A_reconstructed)

# Check if the reconstructed matrix is approximately equal to the original
if np.allclose(A, A_reconstructed):
    print("\nA_reconstructed is identical to the original matrix A.")
else:
    print("\nA_reconstructed is NOT identical to the original matrix A.")
```

```
Original Matrix A:
[[ 4 -2  2  1]
 [ 1  1  0  1]
 [-2  1  3 -1]
 [ 1  3 -1  2]]

Reconstructed Matrix A_reconstructed:
[[ 4.00000000e+00 -2.00000000e+00  2.00000000e+00  1.00000000e+00]
 [ 1.00000000e+00  1.00000000e+00 -4.18216367e-16  1.00000000e+00]
 [-2.00000000e+00  1.00000000e+00  3.00000000e+00 -1.00000000e+00]
 [ 1.00000000e+00  3.00000000e+00 -1.00000000e+00  2.00000000e+00]]

A_reconstructed is identical to the original matrix A.
```