PHY1112 Lab 8

Matrix Madness

March 5th, 2024

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | 1 | 2 | 3 | Total |
| Points | 8 | 8 | 6 | 22 |
| Score |  |  |  |  |

Objectives

1. Solve a system of equations using matrices.
2. Finding eigenvalues and eigenvectors
3. Determine whether a complex matrix is Hermitian or not.

Part 1: Solving a System of Equations – The Power of Matrices!

1. (2 points) Solve the follow system of linear equations by hand. You can type your answer, or include a scan/picture of your hand-written work in this document.

.

--------------------

:.

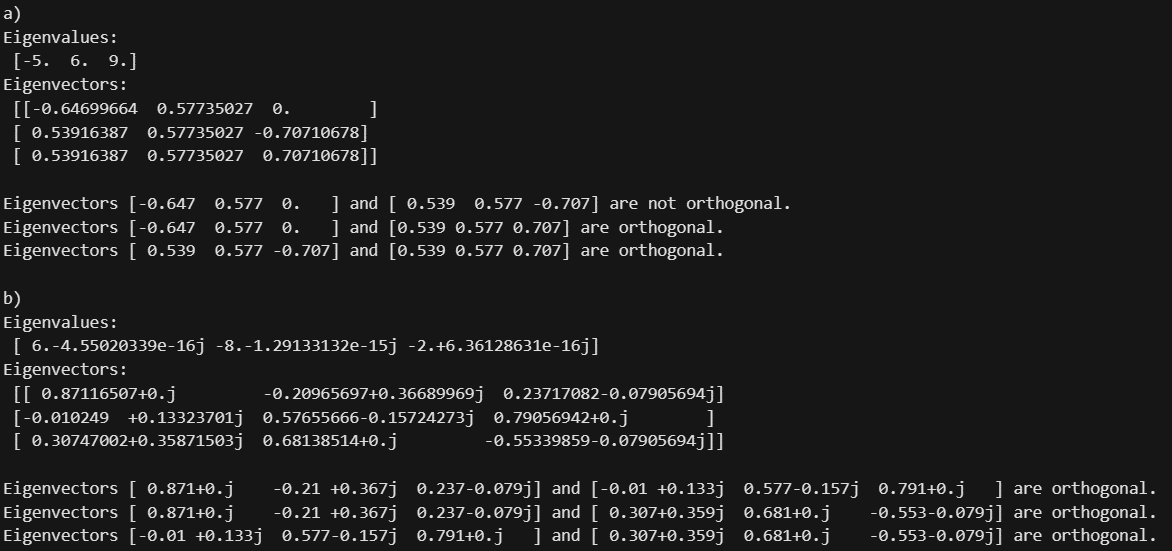
1. (6 points) Using NumPy, solve the follow systems of linear equations. Print the results out to the terminal and include a snapshot of your work in this document.
2. .
3. .
4. .

A black screen with white numbers

Description automatically generated

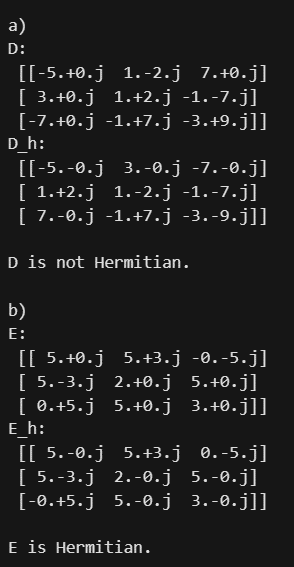
Part 2: Finding Eigenvalues and Eigenvectors

1. (8 points) Using NumPy, find the eigenvalues and eigenvectors of the following matrices, and test whether or not their eigenvectors are orthogonal. Print your results to the terminal and include a snapshot in this document.



Part 3: Hermitian Matrices – Maybe Matrices Really Are Complex!

1. (6 points) Using NumPy, show whether the following matrices are Hermitian or not. Print your results to the terminal and include a snapshot in this document.

 A black screen with white text

Description automatically generated

**CODE:**

'''

Filename:       lab8.py

Author:         Patrick Geraghty

Date Created:   2024-03-05

Date Modified:  2024-03-05

Description:    Lab 8

'''

import numpy as np

# Part 1

print('Part 1: \n')                         # Print Part 1

# a) 3x + 2y = 3, 3x - 6y = 6

a\_coeff = np.array([[-3, 2], [3, -6]])      # Matrix of coefficients

a\_const = np.array([3, 6])                  # Matrix of constants

a = np.linalg.solve(a\_coeff, a\_const)       # Solve the system of equations

print('a) ', a, *end*='\n\n')                 # Print the solution

# b) 2a + 7c + 4d + 2e = 7, 4a - 6b - 6c + 4d -6e = 7, 7b - 7c - 2d + 8e = 1, a - 8c - 3d - 6e = -7, -a + 8b + 6c = 6

b\_coeff = np.array([[2, 0, 7, 4, 2], [4, -6, -6, 4, -6], [0, 7, -7, -2, 8], [1, 0, -8, -3, -6], [-1, 8, 6, 0, 0]])      # Matrix of coefficients

b\_const = np.array([7, 7, 1, -7, 6])                                                                                    # Matrix of constants

b = np.linalg.solve(b\_coeff, b\_const)                                                                                   # Solve the system of equations

print('b) ', b, *end*='\n\n')

# c) -5a + (1 - 2j)b + 7c = 4 - 4j, 3a + (1 + 2j)b - (1 + 7j)c = 9 + 3j, -7a - (1 + 7j)b - (3 + 9j)c = -9

c\_coeff = np.array([[-5, 1 - 2*j*, 7], [3, 1 + 2*j*, -1 - 7*j*], [-7, -1 + 7*j*, -3 + 9*j*]])     # Matrix of coefficients

c\_const = np.array([4 - 4*j*, 9 + 3*j*, -9])                                                # Matrix of constants

c = np.linalg.solve(c\_coeff, c\_const)                                                   # Solve the system of equations

print('c) ', c, *end*='\n\n')

# Part 2

print('Part 2: \n')                                                                                     # Print Part 2

# a) A = [[0, 3, 3], [5, 5, -4], [5, -4, 5]]

A = np.array([[0, 3, 3], [5, 5, -4], [5, -4, 5]])                                                       # Define matrix A

a\_eigenvalues, a\_eigenvectors = np.linalg.eig(A)                                                        # Calculate eigenvalues and eigenvectors

print('a) \nEigenvalues: \n', a\_eigenvalues, '\nEigenvectors: \n', a\_eigenvectors, *end*='\n\n')                 # Print the eigenvalues and eigenvectors

# Check if eigenvectors are orthogonal

for i in range(len(a\_eigenvectors)):                                                                    # Iterate through the eigenvectors

    for j in range(i+1, len(a\_eigenvectors)):

        dot\_product = np.vdot(a\_eigenvectors[:, i], a\_eigenvectors[:, j])                                # Calculate the dot product of the eigenvectors at [i] and [j]

        if np.isclose(dot\_product, 0):                                                                  # If the dot product is close to 0, the eigenvectors are orthogonal

            print(*f*"Eigenvectors {np.round(a\_eigenvectors[i], 3)} and {np.round(a\_eigenvectors[j], 3)} are orthogonal.")

        else:                                                                                           # If the dot product is not close to 0, the eigenvectors are not orthogonal

            print(*f*"Eigenvectors {np.round(a\_eigenvectors[i], 3)} and {np.round(a\_eigenvectors[j], 3)} are not orthogonal.")

print()                                                                                                 # Round to 3 decimal places for readability

# b) B = [[3, 1 - 2j, 3 - 4j], [1 + 2j, -4, -2 + 1j], [3 + 4j, -2 - 1j, -3]]

B = np.array([[3, 1 - 2*j*, 3 - 4*j*], [1 + 2*j*, -4, -2 + 1*j*], [3 + 4*j*, -2 - 1*j*, -3]])                       # Define matrix B

b\_eigenvalues, b\_eigenvectors = np.linalg.eig(B)                                                        # Calculate eigenvalues and eigenvectors

print('b) \nEigenvalues: \n', b\_eigenvalues, '\nEigenvectors: \n', b\_eigenvectors, *end*='\n\n')                 # Print the eigenvalues and eigenvectors

# Check if eigenvectors are orthogonal

for i in range(len(b\_eigenvectors)):                                                                    # Iterate through the eigenvectors

    for j in range(i+1, len(b\_eigenvectors)):

        dot\_product = np.vdot(b\_eigenvectors[:, i], b\_eigenvectors[:, j])                                # Calculate the dot product of the eigenvectors at [i] and [j]

        if np.isclose(dot\_product, 0):                                                                  # If the dot product is close to 0, the eigenvectors are orthogonal

            print(*f*"Eigenvectors {np.round(b\_eigenvectors[i], 3)} and {np.round(b\_eigenvectors[j], 3)} are orthogonal.")

        else:                                                                                           # If the dot product is not close to 0, the eigenvectors are not orthogonal

            print(*f*"Eigenvectors {np.round(b\_eigenvectors[i], 3)} and {np.round(b\_eigenvectors[j], 3)} are not orthogonal.")

print()                                                                                                 # Round to 3 decimal places for readability

# Part 3

print('Part 3: \n')                                                             # Print Part 3

# a) D = [[-5, 1 - 2j, 7], [3, 1 + 2j, -1 - 7j], [-7, -1 + 7j, -3 + 9j]]

D = np.array([[-5, 1 - 2*j*, 7], [3, 1 + 2*j*, -1 - 7*j*], [-7, -1 + 7*j*, -3 + 9*j*]])   # Define matrix D

D\_h = D.conj().T                                                                # Calculate the Hermitian of matrix D

print('a) \nD: \n', D, '\nD\_h: \n', D\_h, *end*='\n\n')                            # Print D and its Hermitian matrix

if np.array\_equal(D, D\_h):                                                      # Check if D is Hermitian

    print('D is Hermitian.')

else:

    print('D is not Hermitian.')

print()

# b) E =

E = np.array([[5, 5 + 3*j*, -5*j*], [5 - 3*j*, 2, 5], [5*j*, 5, 3]])                    # Define matrix E

E\_h = E.conj().T                                                                # Calculate the Hermitian of matrix E

print('b) \nE: \n', E, '\nE\_h: \n', E\_h, *end*='\n\n')                            # Print E and its Hermetian matrix

if np.array\_equal(E, E\_h):                                                      # Check if E is Hermitian

    print('E is Hermitian.')

else:

    print('E is not Hermitian.')

print()

# c) F =

F = np.array([[3, 1 - 2*j*, 3 - 4*j*], [1 + 2*j*, -4, -2 + 1*j*], [3 + 4*j*, -2 - 1*j*, -3]])   # Define matrix F

F\_h = F.conj().T                                                                    # Calculate the Hermitian of matrix F

print('c) \nF: \n', F, '\nF\_h: \n', F\_h, *end*='\n\n')                                # Print F and its Hermetian matrix

if np.array\_equal(F, F\_h):                                                          # Check if F is Hermitian

    print('F is Hermitian.')

else:

    print('F is not Hermitian.')

print()