PHY1112: Assignment 8

> Matrix Madness: Episode 2

Assigned: March 5th, 2024

Due: March 12th, 2024

Learning Objectives

1. Practice linear algebra in Python
2. Learn about the two-dimensional rotation matrix

Grade Breakdown

|  |  |  |
| --- | --- | --- |
| Part | 1 | Total |
| Points | 23 | 23 |
| Score |  |  |

**Question 1: Rotation Repetition.**

Consider the following matrix,

that rotates a two-dimensional vector in a Cartesian basis, by some angle . is called a rotation matrix.

If the two dimensions of the vector are , then the rotation will occur around the axis, like in the following diagram (where the direction is out of the page),

axis

axis

1. Write a function in Python that returns the rotation matrix as a NumPy array of shape (2,2) for an arbitrary angle , where is to be provided as input to your function.

Use your function to calculate for radians. Print your result to the terminal and include a snapshot in your solutions document.  
**(3 marks)**

**A black background with white numbers

Description automatically generated**

1. Rotate the vector

by an angle of radians to obtain a new vector Perform the matrix-vector multiplication using `np.dot`. Print the answer to the terminal and provide a snapshot of your results.   
**(2 marks)**

****

1. Using `np.linalg.inv`, calculate , which is the inverse of . Then, using `np.transpose`, calculate , which is the transpose of . Print out both results to the terminal and include a snapshot of your results. What do you notice?

**(3 marks)**

****

**A black background with white numbers

Description automatically generated**

1. Now, take the matrix product of and its inverse . Then, take the matrix product of and Print out both results to the screen and include a snapshot of your results. What do you notice?  
   **(3 marks)**





1. Based on your result in part ‘d’, what result would you expect when you apply to your rotated vector from part ‘b’, Verify your hypothesis by using your script to calculate .   
   **(2 marks)**

U2 will be rotated by π/4, such that u1 is then rotated by π/2.



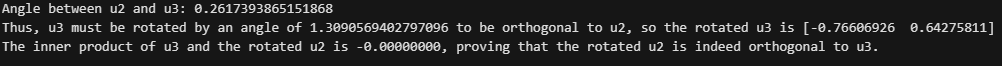
1. The formula for the angle between two vectors and is given by:

Write a function that takes in two vectors as input and returns the angle , using NumPy to implement the formula. Use your function to determine by which angle calculated in part ‘b’ needs to be rotated to make it orthogonal to

If necessary, add or subtract factors of to your result to report a value in the range .

**Hint**: It is easier to use this formula to determine what angles these vectors make with the x-axis, rather than with each other.

Test that your determined angle does indeed make the rotated orthogonal to , by applying the rotation matrix to and taking the inner product of and . Print the result of the inner product to the screen, such that only 8 decimal places are displayed.  
**(5 marks)**

****

1. Using `np.linalg.eig`, determine the eigenvalues and eigenvectors for What do you notice about the eigenvectors?   
   **(4 marks)**

A black background with white text

Description automatically generated

The eigenvalues are both one, while the angle vectors form a 2D identity matrix.

**(23 marks total, 1 for docstrings/file header/variable naming/comments)**

**CODE:**

'''

Filename:       a8.py

Author:         Patrick Geraghty

Date Created:   2024-03-14

Date Modified:  2024-03-14

Description:    Assignment 8. This program contains functions that perform operations on 2D vectors and matrices.

'''

import numpy as np

*def* rotation\_matrix(*theta*):

    '''

    (float) -> np.array([[float, float], [float, float]])

    Returns a rotation matrix for a given angle in radians

    Precondition: theta is a float

    '''

    # The rotation matrix is defined as [[cos(theta), -sin(theta)], [sin(theta), cos(theta)]]. Return the numpy array that reflects this.

    return np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])

*def* vector\_rotation(*v*, *theta*):

    '''

    (np.array([float, float]), float) -> np.array([float, float])

    Returns a vector rotated by a given angle in radians

    Precondition: v is a 2D np.array, theta is a float

    '''

    # The rotated vector is defined as the dot product of the rotation matrix and the vector. Return the numpy array that reflects this.

    return np.dot(rotation\_matrix(theta), v)

*def* angle\_between(*v1*, *v2*):

    '''

    (np.array([float, float]), np.array([float, float])) -> float

    Returns the angle between two vectors in radians

    Precondition: v1 and v2 are 2D np.arrays

    '''

    # The angle between two vectors is defined as the arccos of the dot product of the vectors divided by the product of their magnitudes. Return the result of this calculation.

    return np.arccos(np.dot(v1, v2) / (np.linalg.norm(v1) \* np.linalg.norm(v2)))

*def* orthogonal\_converter(*v1*, *v2*):

    '''

    (np.array([float, float]), np.array([float, float])) -> float

    Returns the angle necessary to rotate v1 to be orthogonal to v2 in radians

    Precondition: v1 and v2 are 2D np.arrays

    '''

    # The angle necessary to rotate v1 to be orthogonal to v2 is defined as the difference between pi/2 and the angle between the vectors. Return the result of this calculation.

    return (np.pi / 2) - angle\_between(v1, v2)

*def* main():

    # Set values for u1 and u2, and calculate the rotation matrix for 45 degrees (pi/4 radians). Print the results.

    r\_45 = rotation\_matrix(np.pi / 4)

    u1 = np.array([0.9397, 0.342])

    u2 = r\_45.dot(u1)

    print(*f*'r\_45: {r\_45}')

    print(*f*'u1: {u1}')

    print(*f*'u2: {u2}')

    print()

    # Calculate the inverse and transpose of r\_45, and the matrix multiplication of r\_45 by its inverse and transpose. Print the results.

    inverse\_r\_45 = np.linalg.inv(r\_45)

    transposed\_r\_45 = r\_45.T

    print(*f*'Inverse of r\_45: {inverse\_r\_45}')

    print(*f*'Transpose of r\_45: {transposed\_r\_45}')

    print()

    # Calculate the matrix multiplication of r\_45 by its inverse and transpose. Print the results.

    r\_45\_by\_r\_45\_inverse = np.matmul(r\_45, inverse\_r\_45)

    r\_45\_by\_transposed\_r\_45 = np.matmul(r\_45, transposed\_r\_45)

    print(*f*' The matrix multiplication of r\_45 by the inverse of r\_45 is {r\_45\_by\_r\_45\_inverse}')

    print(*f*' The matrix multiplication of r\_45 by the transpose of r\_45 is {r\_45\_by\_transposed\_r\_45}')

    print()

    # Calculate the matrix multiplication of u2 by the inverse of r\_45. Print the results.

    u2\_by\_r\_45\_inverse = np.matmul(u2, inverse\_r\_45)

    print(*f*' The matrix multiplication of u2 by the inverse of r\_45 is {u2\_by\_r\_45\_inverse}')

    print()

    # Calculate the angle between u2 and u1, and the angle necessary to rotate u1 to be orthogonal to u2. Print the results.

    u3 = np.array([0.6427, 0.766])

    rotation\_angle = orthogonal\_converter(u2, u3)

    u2\_rotated = vector\_rotation(u2, rotation\_angle)

    print(*f*'u3: {u3}')

    print(*f*'Angle between u2 and u3: {angle\_between(u2, u3)}')

    print(*f*'Thus, u3 must be rotated by an angle of {rotation\_angle} to be orthogonal to u2, so the rotated u3 is {u2\_rotated}')

    # Print the inner product of u3 and the rotated u2, proving that the rotated u2 is indeed orthogonal to u3.

    print(*f*'The inner product of u3 and the rotated u2 is {np.inner(u3, u2\_rotated)*:.8f*}, proving that the rotated u2 is indeed orthogonal to u3.')

    print()

    # Calculate the eigenvalues and eigenvectors of the rotation matrix for 0 degrees (0 radians). Print the results.

    r0 = rotation\_matrix(0)

    eigenvalues, eigenvectors = np.linalg.eig(r0)

    print(*f*'Eigenvalues of r0: {eigenvalues}')

    print(*f*'Eigenvectors of r0: {eigenvectors}')