12.62

Sai Krishna Bakki - EE25BTECH11049

Question

The eigenvalues of the matrix

$$\begin{pmatrix} 2 & 3 & 0 \\ 3 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

are

Theoretical Solution

Given

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 0 \\ 3 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

To find eigenvalues of the matrix $\mathbf{A}$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{2}$$

$$\left(\mathbf{A} - \lambda\mathbf{I}\right)\mathbf{x} = 0 \tag{3}$$

$$\left|\mathbf{A} - \lambda\mathbf{I}\right| = 0 \tag{4}$$

Theoretical Solution

$$\begin{vmatrix} 2-\lambda & 3 & 0 \\ 3 & 2-\lambda & 0 \\ 0 & 0 & 1-\lambda \end{vmatrix} = 0 \tag{5}$$

$$(2-\lambda)\left((2-\lambda)(1-\lambda)-0\right) - 3(3)(1-\lambda) = 0 \tag{6}$$

$$(1-\lambda)\left((2-\lambda)^2 - 9\right) = 0 \tag{7}$$

$$\lambda = 1, -1, 5 \tag{8}$$

$\therefore$ The eigenvalues of the matrix are 1,-1 and 5.

C Code

```c
#include<math.h>
    void find_2x2_eigenvalues(double a, double b, double c,
        double d, double* eig1, double* eig2) {
    // For the equation x^2 + Bx + C = 0, the solutions are (-B
        +/- sqrt(B^2 - 4C)) / 2.
    // Here, B = -(a+d) and C = (ad-bc).
    double trace = a + d;
    double determinant = a * d - b * c;

    // Calculate the discriminant: sqrt(trace^2 - 4*determinant)
    double discriminant_sqrt = sqrt(trace * trace - 4 *
        determinant);

    // Calculate the two eigenvalues using the formula
    *eig1 = (trace + discriminant_sqrt) / 2.0;
    *eig2 = (trace - discriminant_sqrt) / 2.0;
}
```

Python Code Through Shared Output

```
import ctypes
import numpy as np
import os
import platform

# --- Step 1: Compile the C code into a shared library ---
# This script will attempt to compile the C code automatically.
# The C source file is expected to be 'eigenv.c'.

c_file_name = 'eigenv.c'

# Determine the correct file extension for the shared library
    based on the OS
if platform.system() == Windows:
    lib_name = 'eigen_lib.dll'
    compile_command = fgcc -shared -o {lib_name} -fPIC {
        c_file_name}
elif platform.system() == Darwin: # macOS
    lib_name = 'eigen_lib.dylib'
```

Python Code Through Shared Output

```
    compile_command = fgcc -shared -o {lib_name} -fPIC {c_file_name}
else: # Linux
    lib_name = 'eigenv.so'
    compile_command = fgcc -shared -o {lib_name} -fPIC {c_file_name}

# Compile the C code if the library file doesn't exist
if not os.path.exists(lib_name):
    print(fShared library '{lib_name}' not found. Attempting to compile '{c_file_name}'...)
    exit_code = os.system(compile_command)
    if exit_code != 0:
        print(f\nError: Compilation failed. Please ensure GCC is installed and in your system's PATH.)
        print(fManual compile command: {compile_command})
        exit()
    print(Compilation successful.)
```

Python Code Through Shared Output

```
# --- Step 2: Load the shared library using ctypes ---
try:
    # Use the absolute path to ensure the library is found
    eigen_lib = ctypes.CDLL(os.path.abspath(lib_name))
except OSError as e:
    print(fError loading shared library: {e})
    exit()

# --- Step 3: Define the function signature (argument and return types) ---
# The C function is:
# void find_2x2_eigenvalues(double a, double b, double c, double d, double* eig1, double* eig2)
find_2x2_eigenvalues_c = eigen_lib.find_2x2_eigenvalues
find_2x2_eigenvalues_c.argtypes = [
    ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double,
    ctypes.POINTER(ctypes.c_double),
    ctypes.POINTER(ctypes.c_double)]
```

Python Code Through Shared Output

```
find_2x2_eigenvalues_c.restype = None # Corresponds to a 'void'
    return type in C
# --- Step 4: Prepare data and call the C function ---

# The full 3x3 matrix is block-diagonal, so we can analyze it in
    parts.
# [[2, 3, 0],
# [3, 2, 0],
# [0, 0, 1]]
# One eigenvalue is 1. The other two come from the top-left 2x2
    sub-matrix.
sub_matrix = np.array([[2, 3], [3, 2]])
a, b = sub_matrix[0]
c, d = sub_matrix[1]

# Create C-compatible double variables to hold the results from
    the C function
eig1_c = ctypes.c_double()
eig2_c = ctypes.c_double()
```

Python Code Through Shared Output

```
print(fCalling C function to find eigenvalues of the sub-matrix:\n{sub_matrix}\n)
# Call the C function, passing pointers to the result variables
find_2x2_eigenvalues_c(a, b, c, d, ctypes.byref(eig1_c), ctypes.byref(eig2_c))

# --- Step 5: Retrieve the results and combine them ---
eigenvalues_from_c = [eig1_c.value, eig2_c.value]
third_eigenvalue = 1.0
all_eigenvalues = eigenvalues_from_c + [third_eigenvalue]

# Sort for consistent output
all_eigenvalues.sort(reverse=True)
print(fEigenvalues from C function: {eigenvalues_from_c})
print(fThird eigenvalue from observation: {third_eigenvalue})
print(f\nFinal eigenvalues for the 3x3 matrix are: {all_eigenvalues})
```

Python Code

```python
import numpy as np

A = np.array([[2, 3, 0],
              [3, 2, 0],
              [0, 0, 1]])

# Use numpy's linear algebra module (linalg) to find the eigenvalues.
# The function eigvals() returns the eigenvalues of a square matrix.
eigenvalues = np.linalg.eigvals(A)

# Print the original matrix and the calculated eigenvalues.
print(Matrix:)
print(A)
print(\nEigenvalues:)
print(eigenvalues)
```