12.784

EE25BTECH11043 - Nishid Khandagre

October 11, 2025

Question

Let $A = (a_{ij})$ be a 3×3 real matrix such that $A\begin{pmatrix}1\\2\\1\end{pmatrix} = 2\begin{pmatrix}1\\2\\1\end{pmatrix}$, $A\begin{pmatrix}0\\1\\1\end{pmatrix} = 2\begin{pmatrix}0\\1\\1\end{pmatrix}$ and $A\begin{pmatrix}-1\\1\\0\end{pmatrix} = 4\begin{pmatrix}-1\\1\\0\end{pmatrix}$. If m is the degree of the minimal polynomial of A, then $a_{11} + a_{21} + a_{31} + m$ equals

Theoretical Solution

Solution: Given eigen relations:

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

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

$$\mathbf{A}\begin{pmatrix}-1\\1\\0\end{pmatrix} = 4\begin{pmatrix}-1\\1\\0\end{pmatrix} \tag{3}$$

Theoretical Solution

The eigenvectors are :

$$\mathbf{v_1} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \quad \mathbf{v_2} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{v_3} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \tag{4}$$

as $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$ are linearly independent.

Form the matrix $\mathbf{P}$ with these eigenvectors as columns:

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \tag{5}$$

Theoretical Solution

The inverse of $\mathbf{P}$ is:

$$\mathbf{P}^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{3}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix} \tag{6}$$

$$\mathbf{P} = \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} \end{pmatrix} \tag{7}$$

$$\mathbf{AP} = \begin{pmatrix} \mathbf{Av_1} & \mathbf{Av_2} & \mathbf{Av_3} \end{pmatrix} \tag{8}$$

$$\mathbf{AP} = \begin{pmatrix} 2\mathbf{v_1} & 2\mathbf{v_2} & 4\mathbf{v_3} \end{pmatrix} \tag{9}$$

$$\mathbf{AP} = \mathbf{PD} \tag{10}$$

$$\mathbf{D} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix} \tag{11}$$

Theoretical Solution

$$\mathbf{PD} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix} \tag{12}$$

$$= \begin{pmatrix} 2 & 0 & -4 \\ 4 & 2 & 4 \\ 2 & 2 & 0 \end{pmatrix} \tag{13}$$

Now, compute $\mathbf{A} = \mathbf{PDP}^{-1}$:

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & -4 \\ 4 & 2 & 4 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{3}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix} \tag{14}$$

$$\mathbf{A} = \begin{pmatrix} 3 & -1 & 1 \\ -1 & 3 & -1 \\ 0 & 0 & 2 \end{pmatrix} \tag{15}$$

The sum of the first column elements is:

$$a_{11} + a_{21} + a_{31} = 3 + (-1) + 0 = 2 \tag{16}$$

The eigenvalues of **A** are 2 (with algebraic multiplicity 2) and 4 (with algebraic multiplicity 1).

Theoretical Solution

Since there are three linearly independent eigenvectors, the matrix $\mathbf{A}$ is diagonalizable. The minimal polynomial is the product of distinct linear factors corresponding to the eigenvalues.

$$m_A(x) = (x-2)(x-4) \tag{17}$$

The degree of the minimal polynomial, m is 2.

$$a_{11} + a_{21} + a_{31} + m = 2 + 2 = 4 \tag{18}$$

C Code

```c
#include <stdio.h>
#include <math.h> // For fabs() for determinant calculation
    tolerance

// Function to calculate the determinant of a 3x3 matrix
double determinant_3x3(double m[3][3]) {
    return m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1]) -
           m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0]) +
           m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);
}
// Function to calculate the inverse of a 3x3 matrix
// Returns 1 on success, 0 on failure (singular matrix)
int inverse_3x3(double m[3][3], double inv[3][3]) {
    double det = determinant_3x3(m);
    if (fabs(det) < 1e-9) { // Check for singularity (determinant
        close to zero)
        return 0; // Singular matrix, inverse does not exist
    }
```

C Code

```c
    double invDet = 1.0 / det;

    inv[0][0] = (m[1][1] * m[2][2] - m[1][2] * m[2][1]) * invDet;
    inv[0][1] = (m[0][2] * m[2][1] - m[0][1] * m[2][2]) * invDet;
    inv[0][2] = (m[0][1] * m[1][2] - m[0][2] * m[1][1]) * invDet;

    inv[1][0] = (m[1][2] * m[2][0] - m[1][0] * m[2][2]) * invDet;
    inv[1][1] = (m[0][0] * m[2][2] - m[0][2] * m[2][0]) * invDet;
    inv[1][2] = (m[0][2] * m[1][0] - m[0][0] * m[1][2]) * invDet;

    inv[2][0] = (m[1][0] * m[2][1] - m[1][1] * m[2][0]) * invDet;
    inv[2][1] = (m[0][1] * m[2][0] - m[0][0] * m[2][1]) * invDet;
    inv[2][2] = (m[0][0] * m[1][1] - m[0][1] * m[1][0]) * invDet;

    return 1; // Success
}
```

C Code

```c
// Function to multiply a 3x3 matrix by a 3x1 vector
void multiply_mat_vec_3x3(double mat[3][3], double vec[3], double
     result[3]) {
    result[0] = mat[0][0] * vec[0] + mat[0][1] * vec[1] + mat
        [0][2] * vec[2];
    result[1] = mat[1][0] * vec[0] + mat[1][1] * vec[1] + mat
        [1][2] * vec[2];
    result[2] = mat[2][0] * vec[0] + mat[2][1] * vec[1] + mat
        [2][2] * vec[2];
}
```

C Code

```
// Main function to solve the problem
// It takes pointers to return the calculated sum and minimal
    polynomial degree
void solve_matrix_problem_simplified(double *result_sum, int *
    minimal_poly_degree) {
    // 1. Determine minimal polynomial degree (m)
    // Distinct eigenvalues are 2 and 4.
    // Minimal polynomial m(x) = (x - 2)(x - 4), degree = 2.
    *minimal_poly_degree = 2;
    //2.Find coefficients c1, c2, c3 for e1 = c1*v1+c2*v2+c3*v3
    // v1 = [1, 2, 1]^T, v2 = [0, 1, 1]^T, v3 = [-1, 1, 0]^T
    // Equation: P * [c1, c2, c3]^T = [1, 0, 0]^T, where P = [v1
         | v2 | v3]
    double P_matrix[3][3] = {
        {1.0, 0.0, -1.0},
        {2.0, 1.0, 1.0},
        {1.0, 1.0, 0.0}
    };
```

C Code

```
double e1_vec[3] = {1.0, 0.0, 0.0};
double P_inverse[3][3];
int success = inverse_3x3(P_matrix, P_inverse); // Calculate
    inverse
if (!success) { /* Handle error for singular P_matrix */
    return; }
double c_vec[3]; // To store [c1, c2, c3]^T

// [c1, c2, c3]^T = P_inverse * e1_vec
multiply_mat_vec_3x3(P_inverse, e1_vec, c_vec);

double c1 = c_vec[0];
double c2 = c_vec[1];
double c3 = c_vec[2];
```

C Code: Solution Logic

```
// 3. Calculate A*e1 = a11, a21, a31
// A*e1 = c1*(A*v1) + c2*(A*v2) + c3*(A*v3)
// A*e1 = c1*(lambda1*v1) + c2*(lambda2*v2) + c3*(lambda3*v3)
// A*e1 = c1*(2*v1) + c2*(2*v2) + c3*(4*v3)

double v1_arr[3] = {1.0, 2.0, 1.0};
double v2_arr[3] = {0.0, 1.0, 1.0};
double v3_arr[3] = {-1.0, 1.0, 0.0};

double a11_a21_a31_vec[3] = {0.0, 0.0, 0.0};

// Add c1 * 2 * v1
a11_a21_a31_vec[0] += c1 * 2.0 * v1_arr[0];
a11_a21_a31_vec[1] += c1 * 2.0 * v1_arr[1];
a11_a21_a31_vec[2] += c1 * 2.0 * v1_arr[2];
```

C Code

```c
    // Add c2 * 2 * v2
    a11_a21_a31_vec[0] += c2 * 2.0 * v2_arr[0];
    a11_a21_a31_vec[1] += c2 * 2.0 * v2_arr[1];
    a11_a21_a31_vec[2] += c2 * 2.0 * v2_arr[2];

    // Add c3 * 4 * v3
    a11_a21_a31_vec[0] += c3 * 4.0 * v3_arr[0];
    a11_a21_a31_vec[1] += c3 * 4.0 * v3_arr[1];
    a11_a21_a31_vec[2] += c3 * 4.0 * v3_arr[2];

    double a11 = a11_a21_a31_vec[0];
    double a21 = a11_a21_a31_vec[1];
    double a31 = a11_a21_a31_vec[2];

    // 4. Calculate the final sum
    *result_sum = a11 + a21 + a31 + (*minimal_poly_degree);
}
```

Python Code (using C shared library)

```python
import ctypes
import numpy as np

# Load the shared library
lib_matrix = ctypes.CDLL(./code26.so)

# Define argument types and return type for the C function
lib_matrix.solve_matrix_problem_simplified.argtypes = [
    ctypes.POINTER(ctypes.c_double), # result_sum
    ctypes.POINTER(ctypes.c_int) # minimal_poly_degree
]
lib_matrix.solve_matrix_problem_simplified.restype = None

# Create ctypes variables to hold the results
result_sum = ctypes.c_double()
minimal_poly_degree = ctypes.c_int()
```

Python Code (using C shared library)

```python
# Call the C function to solve the problem
lib_matrix.solve_matrix_problem_simplified(
    ctypes.byref(result_sum),
    ctypes.byref(minimal_poly_degree)
)

# Extract the values from the ctypes variables
final_sum = result_sum.value
degree_m = minimal_poly_degree.value

print(fThe degree of the minimal polynomial (m) is: {degree_m})
print(fThe value of a11 + a21 + a31 + m is: {final_sum:.2f})
```

Python Code (Pure Python Solution)

```
import numpy as np

def solve_matrix_problem_pure_python():

    Solves the given matrix problem using pure Python and NumPy.
    Calculates a11 + a21 + a31 + m.

    # 1. Analyze the given information to find eigenvalues and
        eigenvectors
    v1 = np.array([1, 2, 1]); lambda1 = 2
    v2 = np.array([0, 1, 1]); lambda2 = 2
    v3 = np.array([-1, 1, 0]); lambda3 = 4

    print(--- Problem Analysis ---)
    print(fEigenvector v1: {v1}, Eigenvalue: {lambda1})
    print(fEigenvector v2: {v2}, Eigenvalue: {lambda2})
    print(fEigenvector v3: {v3}, Eigenvalue: {lambda3})
```

Python Code (Pure Python Solution)

```
# Check for linear independence of eigenvectors
P_check = np.array([v1, v2, v3]).T
det_P = np.linalg.det(P_check)
print(fDeterminant of eigenvector matrix P: {det_P:.2f})
if np.isclose(det_P, 0):
    print(Warning: Eigenvectors are linearly dependent.)
else:
    print(Eigenvectors are linearly independent, so A is
        diagonalizable.)

# 2. Determine the degree of the minimal polynomial (m)
distinct_eigenvalues = {lambda1, lambda2, lambda3}
m = len(distinct_eigenvalues)
print(f\nDistinct Eigenvalues: {distinct_eigenvalues})
print(fDegree of the minimal polynomial (m): {m})
```

Python Code (Pure Python Solution)

```
# 3. Find the first column of matrix A (a11, a21, a31)
# This is A * e1, where e1 = [1, 0, 0]^T.
# e1 = c1 * v1 + c2 * v2 + c3 * v3 => P * [c1, c2, c3]^T = e1
P_matrix = np.array([v1, v2, v3]).T
e1_vector = np.array([1, 0, 0])

P_inverse = np.linalg.inv(P_matrix)
print(f\nMatrix P:\n{P_matrix})
print(fInverse of P:\n{P_inverse})

c_coefficients = np.dot(P_inverse, e1_vector)
c1, c2, c3 = c_coefficients
print(fCoefficients (c1, c2, c3): c1={c1:.4f}, c2={c2:.4f},
    c3={c3:.4f})
```

Python Code (Pure Python Solution)

```
# Calculate A * e1 = c1*(lambda1*v1) + c2*(lambda2*v2) + c3*(
    lambda3*v3)
first_column_of_A = (c1 * lambda1 * v1) + \
                    (c2 * lambda2 * v2) + \
                    (c3 * lambda3 * v3)

a11 = first_column_of_A[0]
a21 = first_column_of_A[1]
a31 = first_column_of_A[2]

print(f\nCalculated first column of A: {first_column_of_A})
print(fa11: {a11:.4f}, a21: {a21:.4f}, a31: {a31:.4f})
```

Python Code (Pure Python Solution)

```
    # 4. Calculate the final sum: a11 + a21 + a31 + m
    final_sum = a11 + a21 + a31 + m

    print(f\n--- Final Result ---)
    print(fa11 + a21 + a31 + m = {final_sum:.2f})
    return final_sum

if __name__ == __main__:
    solve_matrix_problem_pure_python()
```