Python Program for Modal Analysis of Torsional Oscillations in a Multi-part Rotor System with a desired number of parts

MAY 18, 2023

University of Kashan

Course: Power System Dynamics

Lecturer: Prof. Taher

Student: Hamed Najafi

Table of Contents

Introduction

This report presents an overview of the implemented code, its functionality, and the obtained results. It serves as a comprehensive explanation of the modal analysis of torsional oscillations in a rotor system.

The provided code performs modal analysis on a rotor system to determine the natural torsional frequencies and mode shapes of the system. The rotor system consists of multiple parts connected by torsional stiffness elements. The code calculates the torsional stiffness matrix (K) and torsional mass matrix (M) based on user-defined input for the number of rotor parts, their masses, and the stiffness between them. The eigenvalue problem is then solved to obtain the eigenvalues (L) and eigenvectors (V) of the system. From the eigenvalues, the natural frequencies (Freqs) of torsional oscillations are computed.

Implementation

The code is implemented using the Python programming language and utilizes the NumPy and Matplotlib libraries. The first step is to construct the torsional stiffness matrix (K) and torsional mass matrix (M) based on the user-defined input for the number of rotor parts, their masses, and stiffness between them. The torsional stiffness matrix (K) is formed by considering the stiffness values between adjacent parts, while the torsional mass matrix (M) is diagonalized based on the masses of individual parts.

Next, the eigenvalue problem is solved by calculating the matrix product of the inverse of the torsional mass matrix (M) and the torsional stiffness matrix (K). The eigenvalues (L) and eigenvectors (V) are obtained from this calculation using NumPy's eig function.

To obtain the natural frequencies (Freqs) of torsional oscillations, the eigenvalues are multiplied by the base frequency (fb) defined by the user. The square root of the absolute values of the eigenvalues multiplied by the base frequency yields the frequencies in Hz. These frequencies represent the natural torsional frequencies of the rotor system.

The eigenvectors (V) are normalized column-wise using the normalize_columns function. This function divides each column by the maximum absolute value within the column. The normalized eigenvectors (normalized_V) represent the mode shapes of the rotor system.

Results and Analysis

The code provides the following outputs:

- ✓ The torsional stiffness matrix (K)
- ✓ The state Matrix (A)
- ✓ The eigenvalues (L) of the system
- ✓ The natural frequencies (Frequencies) in Hz
- ✓ The eigenvectors (V) representing the mode shapes
- ✓ The normalized eigenvectors (normalized_V) representing the normalized mode shapes
- ✓ The mode numbers (Modenumber) calculated based on the count of sign changes in the normalized eigenvectors

The matrix (A1) shows the state matrix (multiplied by -1) that is in the state-space form of differential equations of rotor parts ($\ddot{\boldsymbol{\vartheta}} = -[\mathbf{A1}] \times \boldsymbol{\vartheta}$).

The eigenvalues (L) represent the characteristic values of the system, and their magnitudes determine the natural frequencies of torsional oscillations. The frequencies (Freqs) are obtained by multiplying the square root of the absolute values of the eigenvalues by the base frequency (fb) defined by the user ($\boldsymbol{f_{ti}} = \frac{\sqrt{|\lambda_i|.2\pi f_b}}{2\pi}$). These frequencies represent the natural torsional frequencies of the rotor system.

The eigenvectors (V) indicate the mode shapes of the rotor system. Each column of the eigenvector matrix represents a specific mode shape. The normalized eigenvectors (normalized_V) are obtained by dividing each column of the eigenvector matrix by the maximum absolute value within that column. These normalized mode shapes represent the relative amplitudes of torsional oscillations at different parts of the rotor system.

The mode numbers (Modenumber) are calculated based on the count of sign changes in the normalized eigenvectors. Each mode number corresponds to a specific mode shape, indicating the number of nodal regions in the mode shape.

Conclusion

The provided code allows for the modal analysis of a rotor system to determine its natural torsional frequencies and mode shapes. By inputting the number of rotor parts, their masses, and the stiffness between them, the code calculates the torsional stiffness matrix (K) and torsional mass matrix (M). The eigenvalue problem is then solved to obtain the eigenvalues (L) and eigenvectors (V) of the system. The natural frequencies (Freqs) of torsional oscillations are derived from the eigenvalues, and the normalized mode shapes (normalized_V) provide insights into the relative amplitudes of torsional oscillations at different parts of the rotor system. The visualization aspect of the code helps in interpreting and understanding the mode shapes and their corresponding frequencies.

Code

```python
import numpy as np
import matplotlib.pyplot as plt

def normalize_columns(matrix):
    max_vals = np.max(np.abs(matrix), axis=0)
    max_vals[max_vals == 0] = 1  # To avoid division by zero
    normalized_matrix = matrix / max_vals
    return normalized_matrix

def count_sign_changes(matrix):
    sign_changes = np.sum(np.diff(np.sign(matrix), axis=0) != 0, axis=0)
    return sign_changes

# User Input
"""
num_parts = int(input("Enter number of rotor parts: "))
fb = float(input("Enter the base frequency in Hz: "))
mass = np.array([float(input(f"Enter mass of part {i+1}: ")) for i in range(num_parts)])
#stiffness = np.array([float(input(f"Enter stiffness between parts {i+1} and {i+2}: ")) for i in range(num_parts-1)])
"""

"""
#Problem 0
fb = 50
num_parts = 6
mass = [0.129 , 0.2161, 1.1926, 1.2281, 1.2062, 0.0045]
stiffness = [16.0858, 29.1075, 43.365,  59.0483,  2.3517]
"""
"""
#Problem 1
fb = 60
num_parts = 5
mass = 2*np.array([0.124 , 0.232, 1.155, 1.192, 0.855])
stiffness = [21.8, 48.4, 75.6,  62.3,  1.98]
"""
"""
#Problem 2
fb = 60
num_parts = 5
mass = 2*np.array([0.176 , 1.427, 1.428, 1.428, 0.869])
stiffness = [17.78, 27.66, 31.31,  37.25]
"""
"""
#Problem 3
fb = 60
num_parts = 4
mass = 2*np.array([0.099 , 0.337, 3.68, 0.946])
stiffness = [37.95, 81.91, 82.74]
"""
#Problem 4
fb = 60
num_parts = 6
mass = 2*np.array([0.254 , 0.983, 1.001, 1.009, 1.035, 0.013])
stiffness = [13.9, 18.2, 25.2,  54.9,  5.7]


# Torsional Stiffness Matrix
K = np.zeros((num_parts, num_parts))
for i in range(num_parts-1):
    K[i][i] += stiffness[i]
    K[i+1][i] -= stiffness[i]
    K[i][i+1] -= stiffness[i]
    K[i+1][i+1] += stiffness[i]

# Torsional Mass Matrix
M = np.zeros((num_parts, num_parts))
for i in range(num_parts):
    M[i][i] += mass[i]

```

```python
# Eigenvalue Problem
A1=np.matmul(np.linalg.inv(M), K)

eigenvalues, eigenvectors = np.linalg.eig(A1)

wb=2*np.pi*fb
freqs = np.sqrt(abs(eigenvalues)*wb) / (2*np.pi)

normalized_V = normalize_columns(eigenvectors)
n=num_parts
Modenumber = count_sign_changes(normalized_V)

print("A1=\n",np.round(A1, decimals=2))
print("L=\n",np.round(eigenvalues, decimals=2))
print("Frequencies(Hz)=\n",np.round(freqs, decimals=2))
print("V=\n",np.round(eigenvectors, decimals=2))
print("normalized_V=\n",np.round(normalized_V, decimals=2))

# Create subplots for each column

fig, axes = plt.subplots(n, 1, figsize=(n*1.2, 7))
for i, ax in enumerate(axes):
    x_values = np.arange(1, n+1)
    line, = ax.plot(x_values,normalized_V[:, i], marker='o', linestyle='-', label= "Column{}, Mode{},f = {} Hz".format(i+1, Modenumber[i],np.round(freqs[i], decimals=2)))

    # Add vertical grid with dashed lines
    ax.grid(axis='x', linestyle='dashed')

     # Remove tick numbers on both axes
    ax.set_xticklabels([])
    #ax.set_yticklabels([])
    ax.set_yticks([-1,0,1])
    ax.set_ylim(-1.5, 1.5)

    # Remove the black box around each subplot
    ax.spines['top'].set_visible(False)
    ax.spines['bottom'].set_visible(True)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(True)

    # Position the x-axis in the middle
    ax.spines['bottom'].set_position('center')
    ax.spines['bottom'].set_color('gray')

    # Create the legend at the right side
    legend_text = 'f ' + r'$_{{{}}}$'.format(Modenumber[i]) + ' = {} Hz'.format(np.round(freqs[i], decimals=2))
    #ax.legend([legend_text], loc='center left', bbox_to_anchor=(1, 0.5))

    # Add the legend_text as a text in the plot
    x_range = ax.get_xlim()
    y_range = ax.get_ylim()
    x_text = x_range[1] + 0.05 * (x_range[1] - x_range[0])  # Adjust the position
    y_text = (y_range[0] + y_range[1]) / 2  # Middle of the y-axis
    ax.text(x_text, y_text, legend_text, ha='left', va='center')

    # Label each point with its y-axis value
    for x, y in zip(x_values, normalized_V[:, i]):
        ax.text(x, y, '{:.2f}'.format(y), ha='center', va='bottom')

plt.tight_layout()
plt.show()

```

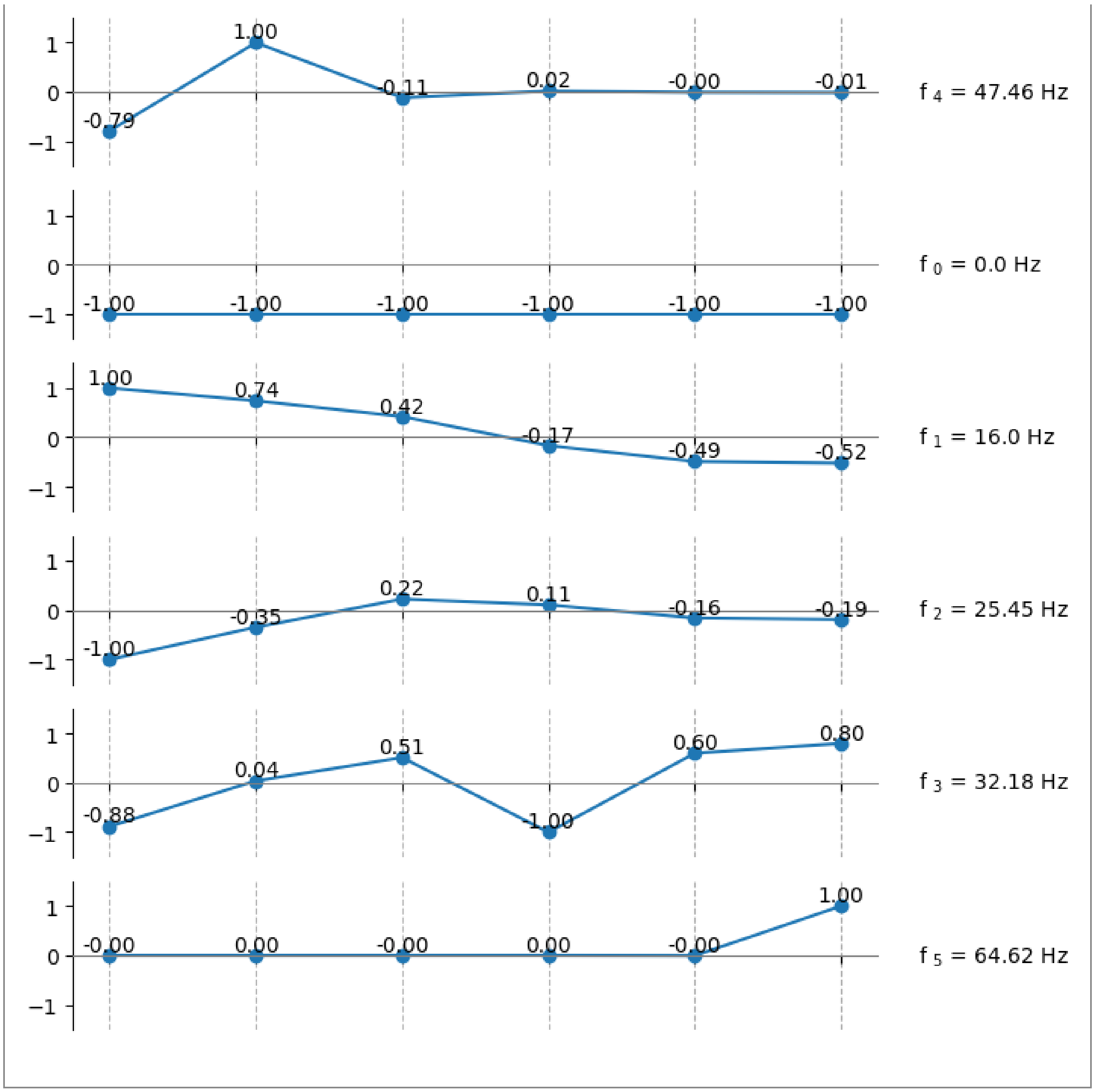
Code Outputs for Example Problems

1st Problem:

```
fb = 50
num_parts = 6
mass = [0.129, 0.2161, 1.1926, 1.2281, 1.2062, 0.0045]
stiffness = [16.0858, 29.1075, 43.365, 59.0483, 2.3517]
```

Output:

```
A1=
 [[ 124.7   -124.7      0.       0.       0.       0.   ]
 [ -74.44   209.13 -134.69    0.       0.       0.   ]
 [   0.     -24.41   60.77  -36.36    0.       0.   ]
 [   0.       0.    -35.31   83.39  -48.08    0.   ]
 [   0.       0.       0.    -48.95   50.9    -1.95]
 [   0.       0.       0.       0.   -522.6   522.6 ]]
L=
 [283.02  -0.     32.16  81.38 130.17 524.77]
Frequencies(Hz)=
 [47.46  0.    16.    25.45 32.18 64.62]
V=
 [[-0.62 -0.41  0.66 -0.9   -0.51 -0.   ]
 [ 0.78 -0.41  0.49 -0.31  0.02  0.   ]
 [-0.09 -0.41  0.28  0.2    0.29 -0.   ]
 [ 0.02 -0.41 -0.11  0.1   -0.57  0.   ]
 [-0.   -0.41 -0.32 -0.14  0.34 -0.   ]
 [-0.01 -0.41 -0.34 -0.17  0.46  1.   ]]
normalized_V=
 [[-0.79 -1.    1.   -1.   -0.88 -0.   ]
 [ 1.   -1.    0.74 -0.35  0.04  0.   ]
 [-0.11 -1.    0.42  0.22  0.51 -0.   ]
 [ 0.02 -1.   -0.17  0.11 -1.    0.   ]
 [-0.   -1.   -0.49 -0.16  0.6  -0.   ]
 [-0.01 -1.   -0.52 -0.19  0.8   1.   ]]
```

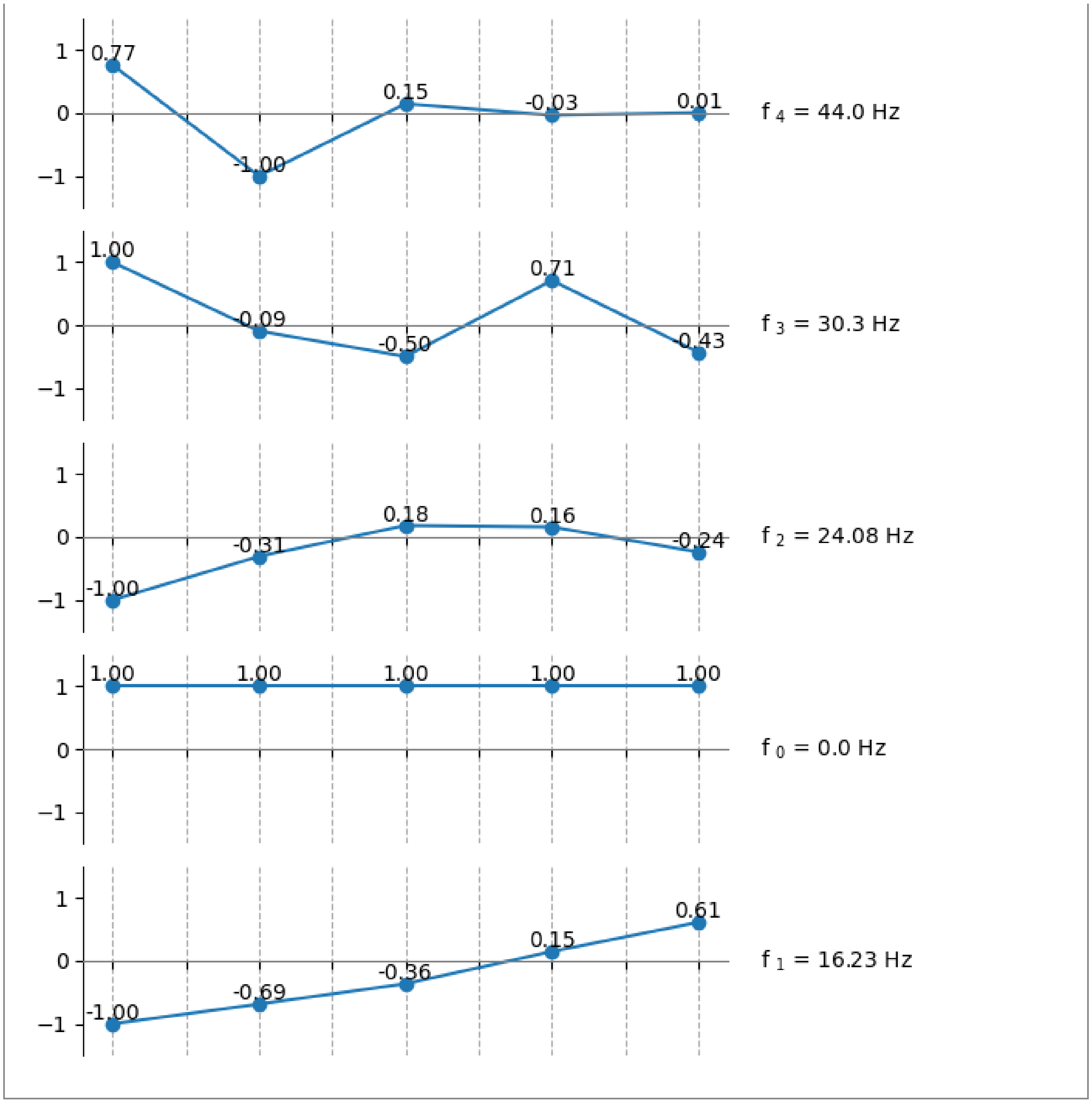

1
0
−1
-0.79
1.00
-0.11
0.02
-0.00
-0.01
f 4 = 47.46 Hz
-1.00
-1.00
-1.00
-1.00
-1.00
-1.00
f 0 = 0.0 Hz
1.00
0.74
0.42
-0.17
-0.49
-0.52
f 1 = 16.0 Hz
-1.00
-0.35
0.22
0.11
-0.16
-0.19
f 2 = 25.45 Hz
-0.88
0.04
0.51
-1.00
0.60
0.80
f 3 = 32.18 Hz
-0.00
0.00
-0.00
0.00
-0.00
1.00
f 5 = 64.62 Hz

2nd Problem:

```
fb = 60
num_parts = 5
mass = 2*np.array([0.124 , 0.232, 1.155, 1.192, 0.855])
stiffness = [21.8, 48.4, 75.6,  62.3,  1.98]
```

Output:

```
A1=
 [[  87.9    -87.9      0.       0.       0.   ]
 [ -46.98  151.29 -104.31     0.       0.   ]
 [   0.    -20.95   53.68  -32.73     0.   ]
 [   0.      0.    -31.71   57.84  -26.13]
 [   0.      0.      0.    -36.43   36.43]]
L=
 [202.71  96.15  60.72  -0.     27.57]
Frequencies(Hz)=
 [44.   30.3  24.08  0.   16.23]
V=
 [[ 0.6   0.72 -0.91  0.45 -0.71]
 [-0.79 -0.07 -0.28  0.45 -0.48]
 [ 0.12 -0.36  0.17  0.45 -0.26]
 [-0.03  0.51  0.14  0.45  0.11]
 [ 0.01 -0.31 -0.22  0.45  0.43]]
normalized_V=
 [[ 0.77  1.   -1.    1.   -1.  ]
 [-1.   -0.09 -0.31  1.   -0.69]
 [ 0.15 -0.5   0.18  1.   -0.36]
 [-0.03  0.71  0.16  1.    0.15]
 [ 0.01 -0.43 -0.24  1.    0.61]]
```



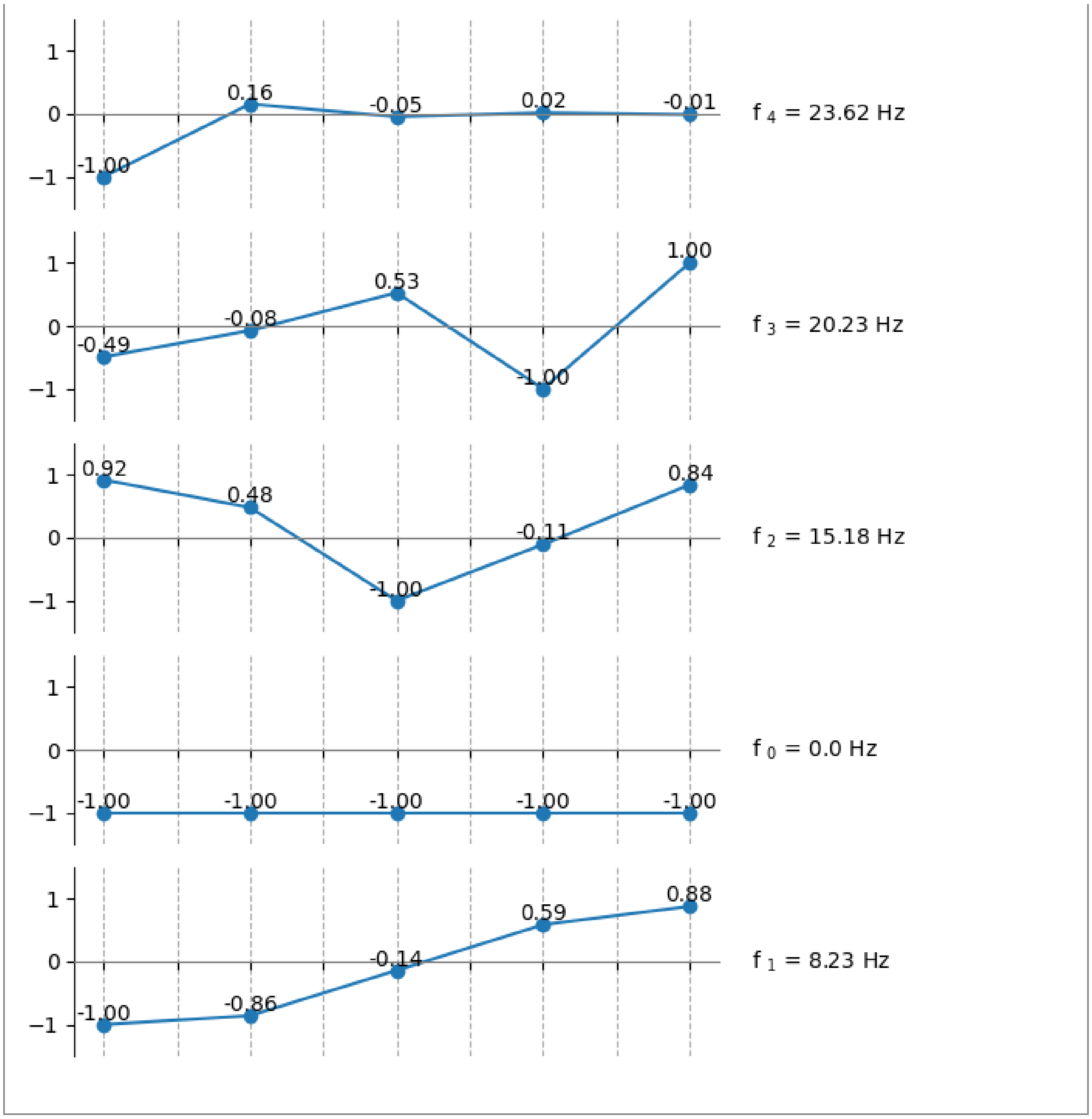
1
0
−1
0.77
-1.00
0.15
-0.03
0.01
f 4 = 44.0 Hz
1.00
-0.09
-0.50
0.71
-0.43
f 3 = 30.3 Hz
-1.00
-0.31
0.18
0.16
-0.24
f 2 = 24.08 Hz
1.00
1.00
1.00
1.00
1.00
f 0 = 0.0 Hz
-1.00
-0.69
-0.36
0.15
0.61
f 1 = 16.23 Hz

3rd Problem:

```
fb = 60
num_parts = 5
mass = 2*np.array([0.176 , 1.427, 1.428, 1.428, 0.869])
stiffness = [17.78, 27.66, 31.31,  37.25]
```

Output:

```
A1=
 [[ 50.51 -50.51   0.      0.      0.   ]
 [ -6.23  15.92  -9.69   0.      0.   ]
 [  0.    -9.68  20.65 -10.96   0.   ]
 [  0.     0.   -10.96  24.01 -13.04]
 [  0.     0.     0.   -21.43  21.43]]
L=
 [58.44 42.84 24.14 -0.    7.1 ]
Frequencies(Hz)=
 [23.62 20.23 15.18  0.    8.23]
V=
 [[-0.99 -0.31  0.55 -0.45 -0.59]
 [ 0.15 -0.05  0.29 -0.45 -0.51]
 [-0.05  0.33 -0.6  -0.45 -0.08]
 [ 0.02 -0.63 -0.06 -0.45  0.35]
 [-0.01  0.63  0.5  -0.45  0.52]]
normalized_V=
 [[-1.   -0.49  0.92 -1.   -1.  ]
 [ 0.16 -0.08  0.48 -1.   -0.86]
 [-0.05  0.53 -1.   -1.   -0.14]
 [ 0.02 -1.   -0.11 -1.    0.59]
 [-0.01  1.    0.84 -1.    0.88]]
```



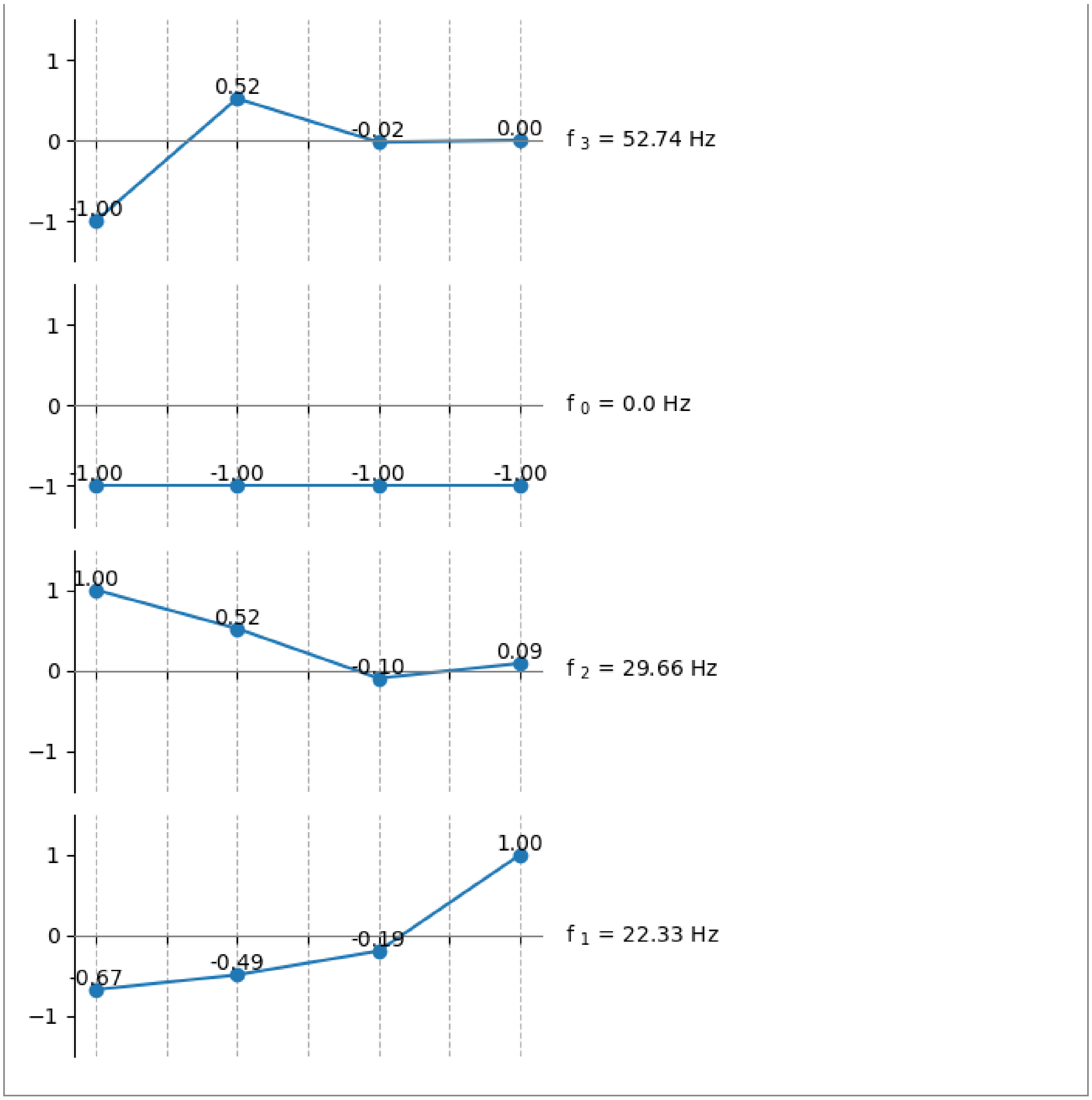
f 4 = 23.62 Hz
-1.00
0.16
-0.05
0.02
-0.01
f 3 = 20.23 Hz
-0.49
-0.08
0.53
-1.00
1.00
f 2 = 15.18 Hz
0.92
0.48
-1.00
-0.11
0.84
f 0 = 0.0 Hz
-1.00
-1.00
-1.00
-1.00
-1.00
f 1 = 8.23 Hz
-1.00
-0.86
-0.14
0.59
0.88

4th Problem:

```
fb = 60
num_parts = 4
mass = 2*np.array([0.099 , 0.337, 3.68, 0.946])
stiffness = [37.95, 81.91, 82.74]
```

Output:

```
A1=
 [[ 191.67 -191.67    0.       0.  ]
 [ -56.31  177.83 -121.53    0.  ]
 [   0.    -11.13   22.37  -11.24]
 [   0.      0.    -43.73   43.73]]
L=
 [291.26  -0.    92.12  52.22]
Frequencies(Hz)=
 [52.74  0.   29.66 22.33]
V=
 [[-0.89 -0.5   0.88 -0.51]
 [ 0.46 -0.5   0.46 -0.37]
 [-0.02 -0.5  -0.09 -0.15]
 [ 0.   -0.5   0.08  0.76]]
normalized_V=
 [[-1.   -1.    1.   -0.67]
 [ 0.52 -1.    0.52 -0.49]
 [-0.02 -1.   -0.1  -0.19]
 [ 0.   -1.    0.09  1.  ]]
```



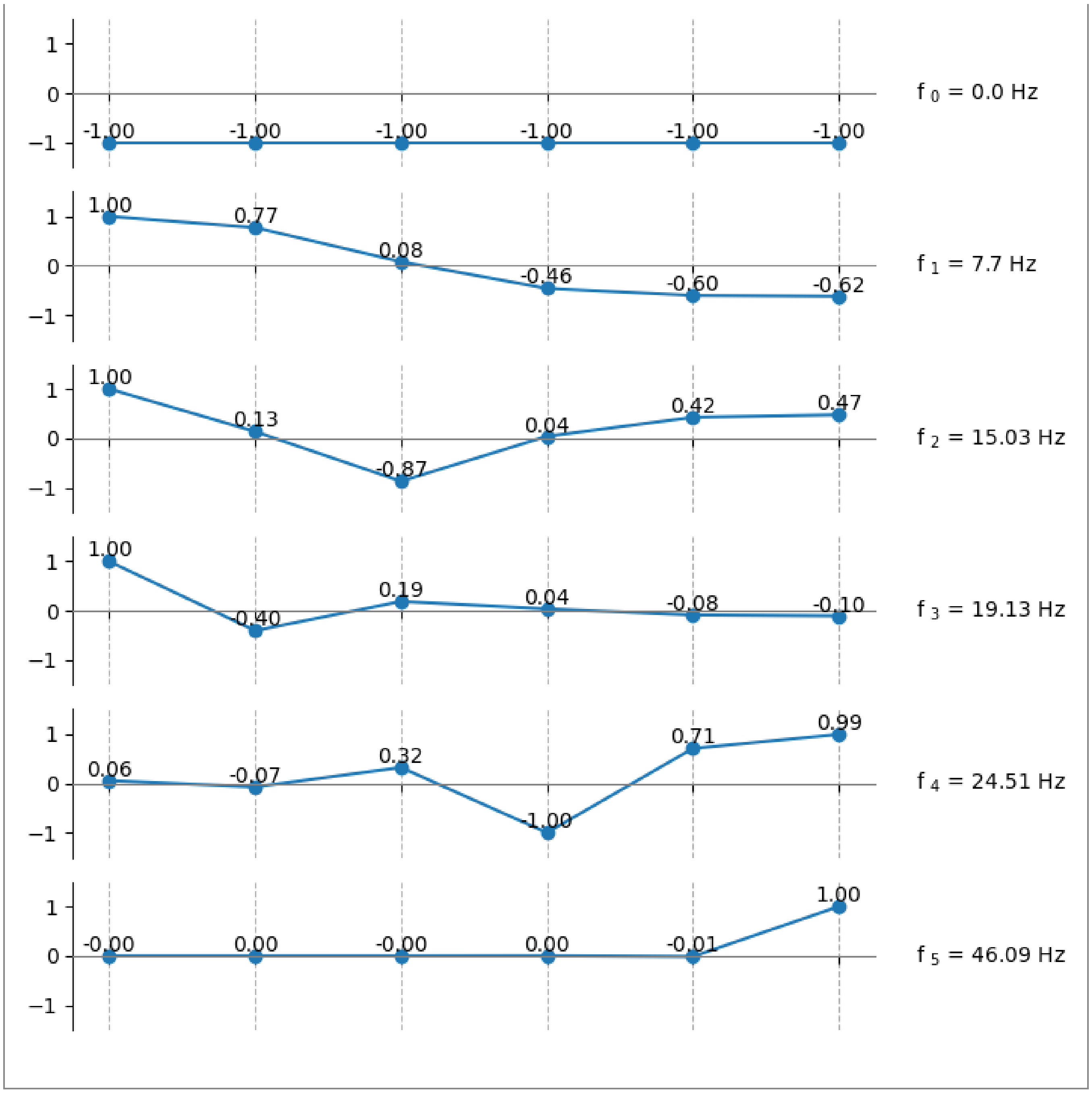
f 3 = 52.74 Hz
-1.00
0.52
-0.02
0.00
f 0 = 0.0 Hz
-1.00
-1.00
-1.00
-1.00
f 2 = 29.66 Hz
1.00
0.52
-0.10
0.09
f 1 = 22.33 Hz
-0.67
-0.49
-0.19
1.00

5th Problem:

```
fb = 60
num_parts = 6
mass = 2*np.array([0.254 , 0.983, 1.001, 1.009, 1.035, 0.013])
stiffness = [13.9, 18.2, 25.2,  54.9,  5.7]
```

Output:

```
A1=
 [[  27.36  -27.36    0.       0.       0.       0.   ]
 [  -7.07   16.33   -9.26    0.       0.       0.   ]
 [   0.     -9.09   21.68  -12.59    0.       0.   ]
 [   0.      0.    -12.49   39.69  -27.21    0.   ]
 [   0.      0.      0.    -26.52   29.28   -2.75]
 [   0.      0.      0.      0.   -219.23  219.23]]
L=
 [ -0.     6.21   23.67   38.33   62.93 222.42]
Frequencies(Hz)=
 [ 0.    7.7   15.03 19.13 24.51 46.09]
V=
 [[-0.41  0.62  0.68  0.91  0.03 -0.  ]
 [-0.41  0.48  0.09 -0.36 -0.04  0.  ]
 [-0.41  0.05 -0.59  0.17  0.2  -0.  ]
 [-0.41 -0.29  0.03  0.04 -0.62  0.  ]
 [-0.41 -0.38  0.29 -0.08  0.44 -0.01]
 [-0.41 -0.39  0.32 -0.09  0.62  1.  ]]
normalized_V=
 [[-1.    1.    1.    1.    0.06 -0.  ]
 [-1.    0.77  0.13 -0.4  -0.07  0.  ]
 [-1.    0.08 -0.87  0.19  0.32 -0.  ]
 [-1.   -0.46  0.04  0.04 -1.    0.  ]
 [-1.   -0.6   0.42 -0.08  0.71 -0.01]
 [-1.   -0.62  0.47 -0.1   0.99  1.  ]]
```


f 0 = 0.0 Hz
-1.00 -1.00 -1.00 -1.00 -1.00 -1.00
f 1 = 7.7 Hz
1.00 0.77 0.08 -0.46 -0.60 -0.62
f 2 = 15.03 Hz
1.00 0.13 -0.87 0.04 0.42 0.47
f 3 = 19.13 Hz
1.00 -0.40 0.19 0.04 -0.08 -0.10
f 4 = 24.51 Hz
0.06 -0.07 0.32 -1.00 0.71 0.99
f 5 = 46.09 Hz
-0.00 0.00 -0.00 0.00 -0.01 1.00

The End