IV. Movimiento en la Base

1. Librerías

In [1]:
```
import numpy as np
np.set_printoptions(formatter = {'float': lambda x: "{0:0.4f}".format(x)})

import matplotlib.pyplot as plt

from tabulate import tabulate
```

2. Grados de Libertad

In [2]:
```
while True:
    try:
        gdl = int(input(f'* Ingrese el número de grados de libertad: '))
        break
    except ValueError:
        print('✗ Ingrese un número de GDL válido.\n')

print(f'* El modelo es de {gdl} GDL.')
```

```
* El modelo es de 2 GDL.
```

3. Modos de Vibración

3.1 Uso de Modos Normalizados

In [3]:
```
while True:
    modos_normalizados = input(f'- ¿Usará los modos normalizados Φ? (S/N): ')

    if modos_normalizados.upper() == 'S':
        simbolo_modo_vibracion = 'Φ'
        modos_normalizados = True

        print(f'✓ Se usarán modos normalizados. ')
        break

    elif modos_normalizados.upper() == 'N':
        simbolo_modo_vibracion = 'A'
        modos_normalizados = False

        print(f'✗ No se usarán modos normalizados. ')
        break

    else:
        print(f'! Ingrese una respuesta valida.\n')
```

```
✓ Se usarán modos normalizados. 
```

3.2 Vectores de Modos de Vibración

In [4]:
```
phi = np.empty((gdl, gdl))

for i in range(gdl):
    print(f'* Modo de vibración i = {i + 1}\n')

    while True:
        phi = input(f'- Vector {simbolo_modo_vibracion}{i + 1}: ')
        _phi = _phi.split(' ')

        try:
            phi[i] = np.array([float(j) for j in _phi])
            break
        except ValueError:
            print(f'! Ingrese el vector {simbolo_modo_vibracion} separado por espacios.\n')

    print(f'> {simbolo_modo_vibracion}{i + 1} = {phi[i]}\n')
```

```
* Modo de vibración i = 1

> Φ1 = [0.8334 0.0742]

* Modo de vibración i = 2

> Φ2 = [0.0731 -0.0339]
```

4. Matriz de Rigidez

$$\mathbf{K} = \begin{pmatrix} K_1 + K_2 & -K_2 & 0 & \cdots & 0 & 0 \\ -K_2 & K_2 + K_3 & -K_3 & \cdots & 0 & 0 \\ 0 & -K_3 & K_3 + K_4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & K_{i-1} + K_i & -K_i \\ 0 & 0 & 0 & \cdots & -K_i & -K_i \end{pmatrix}$$

4.1 Rigideces de Entrepiso

In [5]:
```
k = np.empty(gdl)

for i in range(gdl):
    while True:
        k[i] = float(input(f'* Rigidez K{i + 1} (kg/cm): '))
        print(f'\n> Rigidez K{i + 1} = {k[i]} kg/cm')
        break
    except ValueError:
        print(f'\n! Ingrese un valor de K{i + 1} valido.')
```

```
> Rigidez K1 = 155555.56 kg/cm

> Rigidez K2 = 87179.49 kg/cm
```

4.2 Formulación de la Matriz de Rigidez

In [6]:
```
K = np.zeros((gdl, gdl))

for i in range(gdl):
    ## Cálculo de rigideces

    # Rigidez actual
    k1 = k[i]

    # Rigidez posterior
    try:
        k2 = k[i + 1]
    except IndexError:
        k2 = 0

    ## Llenado de la matriz

    # Posición actual
    K[i, i] = k1 + k2

    # Posición derecha
    if i + 1 >= 0:
        K[i, i + 1] = -k2

    # Posición izquierda
    if i - 1 >= 0:
        K[i, i - 1] = -k1
```

4.3 Representación de la Matriz de Rigidez

In [7]:
```
K_r = tabulate(K, tablefmt = "fancy_grid")

print(f'* Matriz K =\n')
print(K_r)
```

```
* Matriz K =

╒═══════════╤═══════════╕
│ 242735    │ -87179.5  │
├───────────┼───────────┤
│ -87179.5  │ 87179.5   │
╘═══════════╧═══════════╛
```

5. Matriz de Masas

$$\mathbf{M} = \begin{pmatrix} m_1 & 0 & \cdots & 0 \\ 0 & m_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & m_i \end{pmatrix}$$

5.1 Masas Concentradas

In [8]:
```
m = np.empty(gdl)

for i in range(gdl):
    while True:
        try:
            m[i] = float(input(f'* Masa M{i + 1} (kg): '))
            print(f'\n> Masa M{i + 1} = {m[i]} kg')
            break
        except ValueError:
            print(f'\n! Ingrese un valor de M{i + 1} valido.')
```

```
> Masa M1 = 154.74 kg

> Masa M2 = 150.25 kg
```

5.2 Formulación de la Matriz de Masas

In [9]:
```
M = np.zeros((gdl, gdl))

for i in range(gdl):
    M[i, i] = m[i]
```

5.4 Representación de la Matriz de Masas

In [10]:
```
M_r = tabulate(M, tablefmt = "fancy_grid")

print(f'- Matriz M =\n')
print(M_r)
```

```
- Matriz M =

╒═══════╤═══════╕
│ 154.74 │ 0     │
├────────┼───────┤
│ 0      │ 150.25 │
╘═══════╧═══════╛
```

6. Movimiento en la Base

6.1 Amplitud de la Aceleración en la Base

In [11]:
```
while True:
    try:
        Ug = float(input(f'* Ingrese el la amplitud de la aceleración en la base (cm/s2): '))
        break
    except ValueError:
        print(f'! Ingrese un valor de Ug valido.\n')

print(f'* La amplitud de la aceleración basal es de {Ug:.2f} cm/s2.')
```

```
* La amplitud de la aceleración basal es de 200.00 cm/s2.
```

6.2 Frecuencia del Movimiento en la Base

In [12]:
```
while True:
    es_mov_armonico = input(f'- ¿El movimiento en la base es armónico? (S/N): ')

    if es_mov_armonico.upper() == 'S':
        es_mov_armonico = True

        while True:
            try:
                omega = float(input(f'* Frecuencia del movimiento en la base (rad/s): '))
                break
            except ValueError:
                print(f'! Ingrese un valor de Ω valido.\n')

        print(f'✓ La base acelera con una frecuencia Ω = {omega:.3f} rad/s.')
        break

    elif es_mov_armonico.upper() == 'N':
        es_mov_armonico = False

        print(f'✓ La base acelera a un pulso constante.')
        break

    else:
        print(f'! Ingrese una respuesta valida.\n')
```

```
* La base acelera a un pulso constante.
```

6.3 Amortiguamiento en la Base

In [13]:
```
if es_mov_armonico:
    while True:
        try:
            beta = float(input(f'* Ingrese el coeficiente de amortiguamiento critico : '))
            break
        except ValueError:
            print(f'! Ingrese un valor de β valido.\n')

    print(f'** β = {beta:.3f}.')

else:
    print(f'* No hay amortiguamiento en la base.')
```

```
* No hay amortiguamiento en la base.
```

7.Factores de Participación Estática

$$\Gamma_i = \frac{X_i^T \cdot \mathbf{M} \cdot I_x}{X_i^T \cdot \mathbf{M} \cdot X_i} \quad \forall \quad \Gamma_i = \phi_i^T \cdot \mathbf{M}$$

In [14]:
```
gamma = np.empty(gdl)

for i in range(gdl):
    gamma[i] = phi[i] @ M @ np.ones(gdl)

    if not son_modos_normalizados:
        gamma[i] /= phi[i] @ M @ phi[i]

    print(f'- Factor Γ{i + 1} = {gamma[i]:.4f}\n')
```

```
- Factor Γ1 = 16.3216

- Factor Γ2 = 6.2120
```

8. Superposición Modal

- El desplazamiento de cada uno de los grados de libertad U_j es una *combinación lineal* de cada uno de los modos de vibración X.

$$\mathbf{U} = \sum_{i=1}^{n} a_i(t) X_i \quad \rightarrow \quad \mathbf{U} = a_{(1)} \cdot X_1 + a_{(2)} \cdot X_2 + \cdots + a_{(j)} \cdot X_i + \cdots + a_{n(j)} \cdot X_n$$

- Donde para un pulso súbito en la base:

$$a_{(i)} = \frac{\Gamma_i \cdot \ddot{U}_{(t)}}{\omega_i^2} \cdot FAD\,(t) \quad ; \quad FAD\,(t) = 1 - \cos(w, t)$$

- Y para el caso de una aceleración armónica:

$$a_{(i)} = \frac{\Gamma_i \cdot \sin(\Omega\, t)}{\omega_i^2} \cdot FAD \quad ; \quad FAD = \frac{1}{\sqrt{\left(1 - \frac{\Omega^2}{\omega^2}\right)^2 + 4\beta^2 \frac{\Omega^2}{\omega^2}}}$$

8.1 Frecuencias Circulares

In [15]:
```
w = np.empty(gdl)

for i in range(gdl):
    while True:
        try:
            w[i] = float(input(f'* Frecuencia w{i + 1} (rad/s): '))
            print(f'\n> w{i + 1} = {w[i]} rad/s')
            break
        except ValueError:
            print(f'\n! Ingrese un valor de w{i + 1} valido.')
```

```
> w1 = 17.8523 rad/s

> w2 = 42.7807 rad/s
```

8.2 Cálculo de Coeficientes

In [16]:
```
a = np.empty(gdl)

if es_mov_armonico:
    for i in range(gdl):
        a[i] = (gamma[i] / w[i]**2) * np.power(omega / w[i], 2) * 4 * np.power(beta * omega / w[i], 2)

        print(f'* a{i + 1} = {a[i]:.4f} * sin(omega:.3f)t\n')
else:
    for i in range(gdl):
        a[i] = (gamma[i] * Ug) / np.power(w[i], 2)

        print(f'* a{i + 1} = {a[i]:.4f} * (1 - cos(w[i]:.3f)t)\n')
```

```
> a1 =  10.2424 * (1 - cos(17.8523)t)

> a2 =  0.6788 * (1 - cos(42.7813)t)
```

8.3 Desplazamientos Absolutos

In [17]:
```
U = np.empty((gdl, gdl))
Uabs = np.empty(gdl)

for i in range(gdl):
    print(f'* Desplazamiento del nivel {i + 1}:\n')

    for j in range(gdl):
        U[i][j] = a[j] * phi[i][j]

        if es_mov_armonico:
            print(f'\t- Modo {j + 1} =  U{i + 1}{j + 1} = {U[i][j]:.3f} * sin((omega:.2f)t) cm\n')

        else:
            print(f'\t- Modo {j + 1} =  U{i + 1}{j + 1} = {U[i][j]:.3f} * (1 - cos(w[j]:.3f)t) cm\n')

    Uabs[i] = sum(U[i])

    if es_mov_armonico:
        print(f'\t\tU{i + 1} = |Uabs[i]:.3f| * sin((omega:.2f)t) cm\n')
```

```
* Desplazamiento del nivel 1:

        - Modo 1 =  U11 = 0.343 * (1 - cos(17.852t)) cm

        - Modo 1 = 2 =  U12 = 0.050 * (1 - cos(42.781t)) cm

* Desplazamiento del nivel 2:

        - Modo 1 =  U21 = 0.749 * (1 - cos(17.852t)) cm

        - Modo 1 = 2 =  U22 = -0.023 * (1 - cos(42.781t)) cm
```