III. Vibración Forzada

$$\ddot{a}_i(t) + \omega_i^2 \cdot a_i(t) = \Omega_i \cdot f(t)$$

1. Librerías

In [1]:

```python
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]:

```python
while True:
    try:
        gdl = int(input('* 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]:

```python
while True:
    son_modos_normalizados = input('- ¿Usará los modos normalizados Φ? (S/N): ')

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

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

    elif son_modos_normalizados.upper() == 'N':
        simbolo_modo_vibracion = 'x'
        son_modos_normalizados = False

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

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

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

3.2 Vectores de Modos de Vibración

In [4]:

```python
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 = [0.0334 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]:

```python
k = np.empty(gdl)

for i in range(gdl):
    while True:
        try:
            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} válido.')
```

```
> Rigidez K1 = 155555.56 kg/cm

> Rigidez K2 = 87179.49 kg/cm
```

4.2 Formulación de la Matriz de Rigidez

In [6]:

```python
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 < gdl:
        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]:

```python
K_r = tabulate(K, tablefmt = 'fancy_grid')

print('• 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]:

```python
if son_modos_normalizados:
    print('- Se usarán los vectores normalizados Φi.')
    pass
else:
    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} válido.')
```

```
- Se usarán los vectores normalizados Φi.
```

5.2 Formulación de la Matriz de Masas

In [9]:

```python
if son_modos_normalizados:
    print('- Se usarán los vectores normalizados Φi.')
    pass
else:
    M =  np.zeros((gdl, gdl))

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

```
- Se usarán los vectores normalizados Φi.
```

5.3 Representación de la Matriz de Masas

In [10]:

```python
if son_modos_normalizados:
    print('- Se usarán los vectores normalizados Φi.')
    pass
else:
    M_r = tabulate(M, tablefmt = 'fancy_grid')

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

```
- Se usarán los vectores normalizados Φi.
```

6. Fuerzas Externas

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{bmatrix} \quad ; \quad f(t) = \sin(\Omega t)$$

6.1 Vector de Fuerzas

In [11]:

```python
F = np.empty(gdl)

for i in range(gdl):
    while True:
        try:
            F[i] = float(input(f'* Fuerza F{i + 1} (kg): '))
            print(f'\n> Fuerza F{i + 1} = {F[i] /1000 :.2f} ton')
            break
        except ValueError:
            print(f'\n! Ingrese un valor de F{i + 1} válido.')

F = F.reshape(-1, 1)
```

```
> Fuerza F1 = 20.00 ton

> Fuerza F2 = 40.00 ton
```

6.2 Frecuencia de la Acción Externa

In [12]:

```python
while True:
    try:
        omega = float(input('* Frecuencia de la acción externa (rad/s): '))
        break
    except ValueError:
        print('! Ingrese un valor de Ω válido.\n')

print(f'* La acción externa tiene una frecuencia Ω = {omega:.3f} rad/s.')
```

```
* La acción externa tiene una frecuencia Ω = 12.566 rad/s.
```

7.Factores de Participación Estática

$$\Gamma_i = \frac{X_i^T\, F}{X_i^T \cdot \mathbf{M} \cdot X_i} \quad \vee \quad \Gamma_i = \phi_i^T\, F$$

In [13]:

```python
gamma = np.empty(gdl)

for i in range(gdl):
    gamma[i] = phi[i] @ F

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

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

```
- Factor Γ1 = 3636.4000

- Factor Γ2 = 104.4000
```

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.

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

- Donde para una fuerza armónica:

$$a_{i(t)} = \frac{\Gamma_i \cdot \sin(\Omega\, t)}{\omega_i^2} \cdot FAD \quad ; \quad FAD = \frac{1}{1 - \frac{\Omega}{\omega_i}^2}$$

8.1 Frecuencias Circulares

In [14]:

```python
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} válido.')
```

```
> w1 = 17.852 rad/s

> w2 = 42.781 rad/s
```

8.2 Cálculo de Coeficientes

In [15]:

```python
a = np.empty(gdl)

for i in range(gdl):
    a[i] = gamma[i] / np.power(w[i], 2)
    a[i] /= 1 - np.power(omega / w[i], 2)

    print(f'> a{i + 1} →  {a[i]:.4f} • sin({omega:.3f}t)\n')
```

```
> a1 →  22.6172 • sin(12.566t)

> a2 →  0.0624 • sin(12.566t)
```

8.3 Desplazamientos Absolutos

In [16]:

```python
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]
        print(f'\t- Modo i = {j + 1} →  U{i + 1}{j + 1} = {U[i][j]:.3f} • sin({omega:.2f}t) cm\n')

    Uabs[i] = sum(U[i])
    print(f'\t\t∴ U{i + 1} = {Uabs[i]:.3f} • sin({omega:.2f}t) cm\n')
```

```
* Desplazamiento del nivel 1:

        - Modo i = 1 →  U11 = 0.756 • sin(12.57t) cm

        - Modo i = 2 →  U12 = 0.005 • sin(12.57t) cm

                ∴ U1 = 0.761 • sin(12.57t) cm

* Desplazamiento del nivel 2:

        - Modo i = 1 →  U21 = 1.653 • sin(12.57t) cm

        - Modo i = 2 →  U22 = -0.002 • sin(12.57t) cm

                ∴ U2 = 1.651 • sin(12.57t) cm
```

8.4 Desplazamientos Relativos

In [17]:

```python
for i in range(gdl):
    if i == 0:
        print(f'> δ{i + 1} = {Uabs[0]:.3f} • sin({omega:.2f}t) cm\n')
    else:
        print(f'> δ{i + 1} = {Uabs[i] - Uabs[i - 1]:.3f} • sin({omega:.2f}t) cm\n')
```

```
> δ1 = 0.761 • sin(12.57t) cm

> δ2 = 0.890 • sin(12.57t) cm
```

9. Cortante en la Base

- La fuerza cortante en cada entrepiso está dada por:

$$V_{base} = K_1 \cdot \delta_1$$

- Donde δ_1 es la notación del *desplazamiento relativo* del primer entrepiso

In [18]:

```python
V = Uabs[0] * k[0]
print(f'∴ Vbase = {V / 1000:.3f} • sin({omega:.2f}t) ton')
```

```
∴ Vbase = 118.370 • sin(12.57t) ton
```