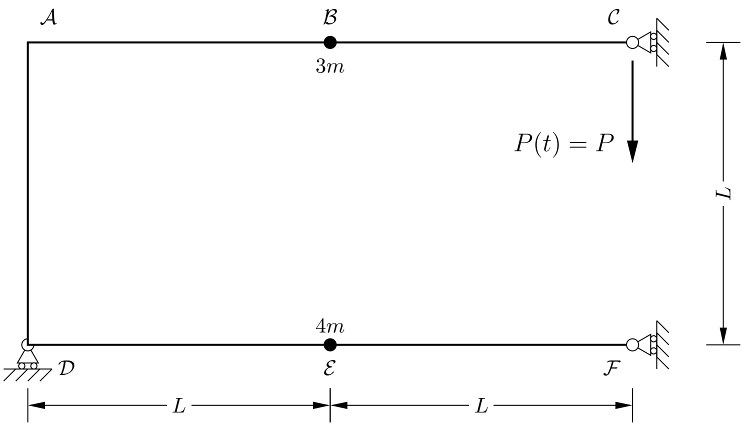
2DoF

Giacomo Boffi

2 DOF System



The system in figure is composed of a single uniform beam supporting two *different* lumped masses.

Neglecting the beam mass and its axial deformability, the system has two dynamic degrees of freedom, x_1 the vertical displacement in $\mathcal{B}$ and x_2, the vertical displacement in $\mathcal{E}$.

The stiffness matrix being

$$\boldsymbol{K} = \frac{2}{5}\frac{EJ}{L^3}\begin{bmatrix}4 & 1\\1 & 4\end{bmatrix}$$

determine the eigenvalues (normalized with respect to $\omega_0^2 = EJ/L^3$) and the eigenvectors of the system (Note: it may be convenient to operate with non-normalied eigenvectors).

The system is at rest when, at time $t = 0$, it is loaded by a constant vertical force P, applied in $\mathcal{C}$.

Introducing an additional degree of freedom x_3, the vertical displacement of $\mathcal{C}$, the augmented stiffness matrix is

$$\underline{\boldsymbol{K}} = \frac{3}{28}\frac{EJ}{L^3}\begin{bmatrix}92 & 6 & -34\\6 & 15 & -1\\-34 & -1 & 15\end{bmatrix}$$

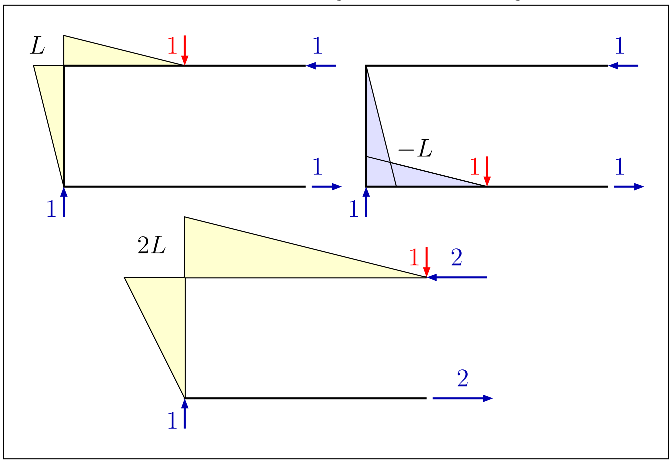
and it is possible to write the dynamic equations of equilibrium in terms of equivalent nodal loads.

1. Write the equations of dynamic equilibrium in matrix form but detailing the values of the equivalent nodal loads.
2. Write the two equations of equilibrium in modal coordinates.
3. Write the expressions of the modal responses.

Solution

Structural matrices

You have the stiffness matrices, initially I have not... here I compute the flexibility matrix for the 3 DOF system starting with the diagrams of the bending moments



next I write the polynomials that represent the bending moments and, using the principle of virtual works, I get the flexibility.

```
Ms = [[ p(0, 0), p(1, 0), p(-1, 1), p(0,  0), p(0, 0)],
      [ p(0, 0), p(0, 0), p(-1, 0), p(1, -1), p(0, 0)],
      [ p(1, 0), p(1, 1), p(-2, 2), p(0,  0), p(0, 0)]]
Ls = [1,1,1,1,1]
F = np.array([[sum(i(*mcl) for mcl in zip(M, C, Ls))
                   for M in Ms] for C in Ms])

dl(r'\boldsymbol F = \frac{L^3}{6EJ}'+pmat(6*F))
```

$$\boldsymbol F = \frac{L^3}{6EJ}\begin{bmatrix} 4.000 & -1.000 & 9.000 \\ -1.000 & 4.000 & -2.000 \\ 9.000 & -2.000 & 24.000 \end{bmatrix}$$

The stiffness associated with the non-dynamic degree of freedom is obviously the inverse of the 3x3 flexibility

```
K = inv(F)
dl(r'\overline{\boldsymbol K} = \frac{3EJ}{28L^3}'+pmat(28*K/3))
```

$$\overline{\boldsymbol K} = \frac{3EJ}{28L^3}\begin{bmatrix} 92.000 & 6.000 & -34.000 \\ 6.000 & 15.000 & -1.000 \\ -34.000 & -1.000 & 15.000 \end{bmatrix}$$

On the other hand, the stiffness associated with the dynamic degrees of freedom (that could be computed using the static condensation procedure) is computed (in a simpler way) as the inverse of the corresponding partition of the flexibility.

The mass matrix is easy...

```
F22 = F[:2,:2]
K22 = inv(F22)
M = np.array(((3,0),(0,4)))
# = np.array(((1,0),(0,1)))

dl(r'\boldsymbol K = \frac{2EJ}{5L^3}\,' + pmat(K22*5/2))
dl(r'\boldsymbol M = m\,' + pmat(M))
```

$$\boldsymbol K = \frac{2EJ}{5L^3}\begin{bmatrix}4.000 & 1.000\\1.000 & 4.000\end{bmatrix}$$

$$\boldsymbol M = m\begin{bmatrix}3.000 & 0.000\\0.000 & 4.000\end{bmatrix}$$

Eigenproblem and modal masses

Here we solve the eigenproblem, put the eigenvalues in a 2x2 matrix, de-normalize the eigenvectors (because we want to avoid irrational numbers) and compute the modal mass matrix.

```
evals, evecs = eigh(K22, M)
Lambda = np.diag(evals)
evecs[:,0] /= evecs[0,0]
evecs[:,1] /= evecs[1,1]
Mstar = evecs.T@M@evecs

dl(r'\boldsymbol \Lambda = '+ pmat(Lambda) +
   r',\qquad\boldsymbol \Omega^2 = \omega_0^2\,\boldsymbol\Lambda')
dl(r'\boldsymbol \Psi = ' + pmat(evecs))
dl(r'\boldsymbol M^\star = m\,' + pmat(Mstar))
```

$$\boldsymbol \Lambda = \begin{bmatrix}0.333 & 0.000\\0.000 & 0.600\end{bmatrix},\qquad\boldsymbol \Omega^2 = \omega_0^2\,\boldsymbol\Lambda$$

$$\boldsymbol \Psi = \begin{bmatrix}1.000 & 2.000\\-1.500 & 1.000\end{bmatrix}$$

$$\boldsymbol M^\star = m\begin{bmatrix}12.000 & 0.000\\0.000 & 16.000\end{bmatrix}$$

Equation of Motion

The efficace load, using the static condensation procedure with the index d denoting dynamic DOF's and the index s denoting static DOF's is

$$\boldsymbol p_{\text{eff}} = \boldsymbol p_d - \boldsymbol K_{ds}\boldsymbol K_{ss}^{-1}\boldsymbol p_s.$$

In our case $\boldsymbol p_d = 0$, $\boldsymbol K_{ss}$ is a scalar and $\boldsymbol p_s = P$ is a scalar as well, so we can write

```
peff = 0 - K[:2,2]/K[2,2]
dl(r'\boldsymbol p_\text{ eff} = ' + pmat(15*peff[:,None]) + r'\,\frac{P}{15}')
```

$$\boldsymbol{p}_{\text{eff}} = \begin{bmatrix} 34.000 \\ 1.000 \end{bmatrix} \frac{P}{15}$$

and the equation of motion is

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \begin{Bmatrix} 34 \\ 1 \end{Bmatrix} \frac{P}{15}.$$

If we write the EoM in modal coordinates

$$\boldsymbol{M}^\star\ddot{\boldsymbol{q}} + \omega_0^2\boldsymbol{M}^\star\boldsymbol{\Lambda}\boldsymbol{q} = \boldsymbol{\Psi}^T\boldsymbol{p}_{\text{eff}} = \boldsymbol{p}^\star$$

```
pstar = evecs.T @ peff
dl(r'''\boldsymbol M^\star\,\ddot{\boldsymbol q} +
       \omega_0^2\boldsymbol M^\star\boldsymbol\Lambda\boldsymbol q = ''' +
   pmat(30*pstar[:,None]) + r'\,\frac{P}{30}')
```

$$\boldsymbol{M}^\star\ddot{\boldsymbol{q}} + \omega_0^2\boldsymbol{M}^\star\boldsymbol{\Lambda}\boldsymbol{q} = \begin{bmatrix} 65.000 \\ 138.000 \end{bmatrix} \frac{P}{30}$$

Eventually we premultiply every term by $\boldsymbol{M}^{\star-1}$ to have the equation of motion written in terms of accelerations

```
Gamma = inv(Mstar) @ pstar
dl(r'''\ddot{\boldsymbol q} + \omega_0^2\boldsymbol\Lambda\boldsymbol q
       = \boldsymbol\Gamma =''' + pmat(720*Gamma[:,None]) + r'\,\frac{P}{720\,m}')
```

$$\ddot{\boldsymbol{q}} + \omega_0^2\boldsymbol{\Lambda}\boldsymbol{q} = \boldsymbol{\Gamma} = \begin{bmatrix} 130.000 \\ 207.000 \end{bmatrix} \frac{P}{720m}$$

The particular integrals

In our case the particular integral is simply a constant term, $\xi_i = \gamma_i/(\lambda_i\omega_0)^2$ and, taking into account that $m\omega_0^2 = k$, we can make the position $\delta = P/k$ and write

```
C = inv(Lambda) @ Gamma
dl(r'\boldsymbol \Xi = ' + pmat(48*C[:,None]) + r'\,\frac{\delta}{48}')
```

$$\boldsymbol{\Xi} = \begin{bmatrix} 26.000 \\ 23.000 \end{bmatrix} \frac{\delta}{48}$$

The general integrals

The general integral being $q_i = A_i\cos(\lambda_i\omega_0 t) + B_i\sin(\lambda_i\omega_0 t) + C_i$ by imposing $\dot{q}_i(0) = 0$ we have $B_i = 0$ and by imposing $q_i(0) = 0$ we have $A_i = -C_i$, hence

$$q_i(t) = C_i(1 - \cos(\lambda_i\omega_0 t))$$

and, substituting the numerical values we have

```python
for i in range(2):
    j = i+1
    dl(r'q_{%d}(t) = %+8.6f\,\delta\,(1-\cos(%f\,\omega_0t))'%(j,C[i],np.sqrt(Lambda[i,i
```

$$q_1(t) = +0.541667\delta(1-\cos(0.577350\omega_0 t))$$

$$q_2(t) = +0.479167\delta(1-\cos(0.774597\omega_0 t))$$

Initialization Cell

```python
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import eigh, inv
from IPython.display import Latex, HTML

def p(*l):
    'wrapper around poly1d class to modify call syntax'
    return np.poly1d(l)

def i(p,q,l):
'''computes definite integral of p*q from 0 to l
p and q are instances of poly1d class'''

    pqi = (p*q).integ()
    return pqi(l)-pqi(0)

def pmat(mat, fmt='%.3f'):
    'returns a LaTeX string representing a 2D array'
    inside =  r'\\'.join('&'.join(fmt%x for x in row) for row in mat)
    return r'\begin{bmatrix}'+inside+r'\end{bmatrix}'

def dl(s):
    'rich display a LaTeX string (surrounded by "$$")'
    display(Latex('$$'+s+'$$'))
```