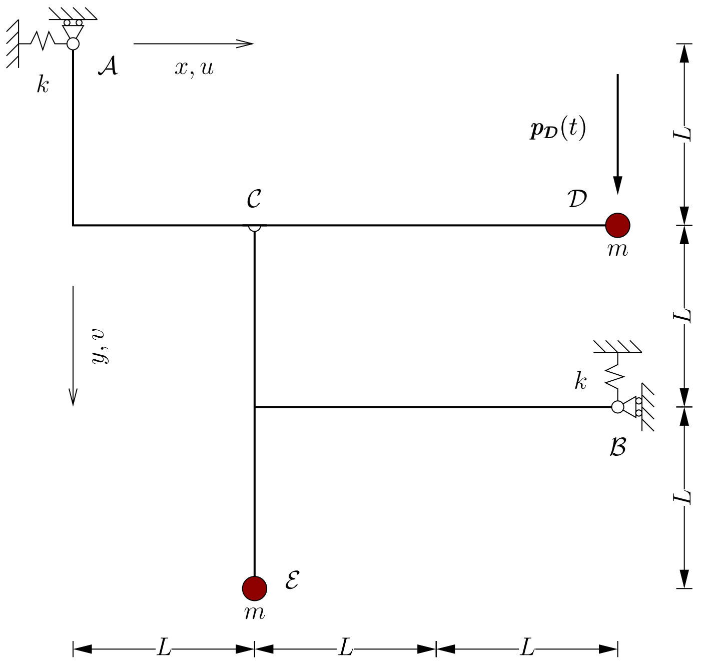
2 DOF System

G. Boffi

1 2 DOF system

All the helper functions are defined at the end of this file.



Using the function `material_point(xy, body)` the relevant points in our mechanism can be specified as follows

```
A = material_point(xy=(0,0), body=0)
B = material_point(xy=(3,2), body=1)
C = material_point(xy=(1,1), body=0)
D = material_point(xy=(3,1), body=0)
E = material_point(xy=(1,3), body=1)
```

The relevant individual points (that is, the ones associated with a force) are then collected in an array and the corresponding spring stiffnesses (in the orthogonal directions), masses and applied external forces are collected in parallel in three other arrays.

```
points = array([A,          D,          B,          E])
masses = array((mass(0),    mass(1),    mass(0),    mass(1)))
stiffs = array((stif(1, 0), stif(0, 0), stif(0, 1), stif(0, 0)))
forces = array((forc(0, 0), forc(0, 1), forc(0, 0), forc(0, 0)))
```

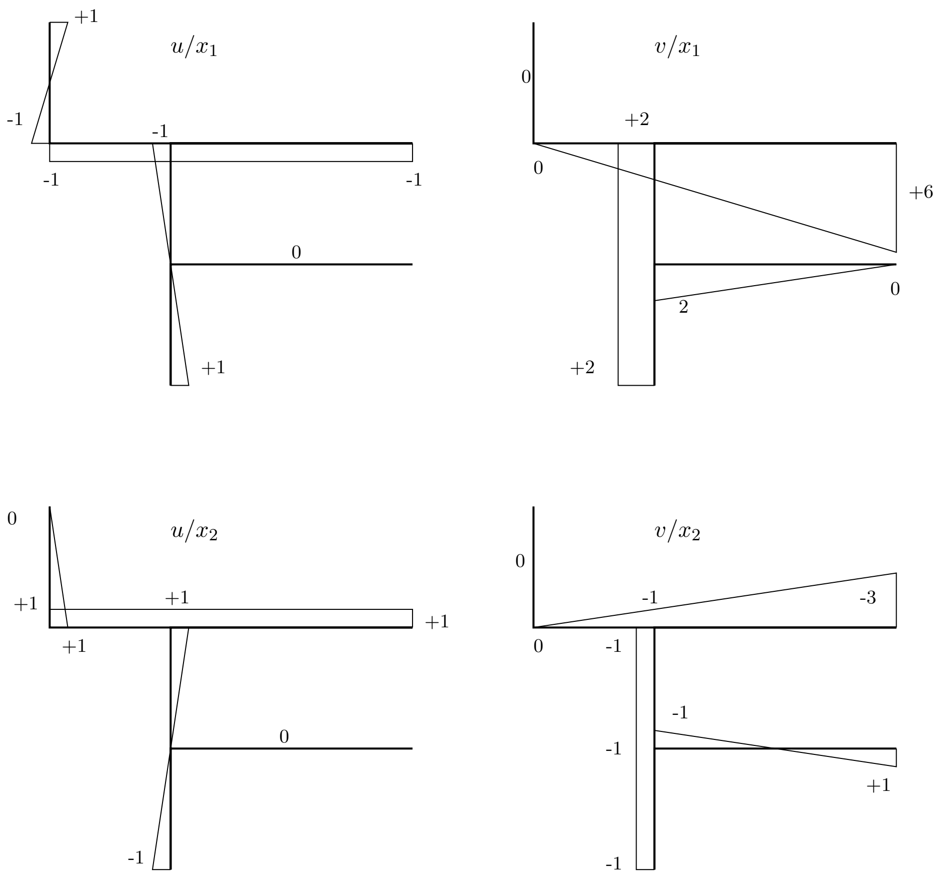
Our system has 2 DOF, for each DOF we have to determine the centre of instant rotation and the amplitude of the rotation for each of the (two) rigid bodies, so that one DOF is equal to 1 and all the other DOFs (here the other one) are (is) equal to 0.

This part of the procedure was performed on squared paper, with a pencil and a ruler — these are the results

```
#---------------body 0-------------body 1------
O_r_s = [[O_dr((0, 0.5), +2), O_dr((3, 2), -1)], # x_1 = 1, x_2 = 0
         [O_dr((0, 0.0), -1), O_dr((2, 2), +1)]] # x_1 = 0, x_2 = 1
#.............CIR.amp..............CIR...amp
```

For every DOF x_j we unfold from `O_r_s` the list of the CIRs and the rotation amplitudes (`for oas in O_r_s` below) for every body in the system, next for every relevant point we compute the displacement for $x_j = 1, x_{i\neq j} = 0$ using the utility function `disp()` defined below.

```
uv = array([[disp(P, oas) for P in points] for oas in O_r_s])
for dd, label in zip(uv, ('x1=1,x2=0', 'x1=0,x2=1')):
    print(label, end='->␣')
    print(';␣␣'.join('%s:u=%+4.1f,v=%+4.1f'%(p,*v)for p, v in zip('ADBE',dd)))
```



```
x1=1,x2=0-> A:u=+1.0,v=+0.0;  D:u=-1.0,v=+6.0;  B:u=+0.0,v=+0.0;  E:u=+1.0,v=+2.0
x1=0,x2=1-> A:u=+0.0,v=-0.0;  D:u=+1.0,v=-3.0;  B:u=+0.0,v=+1.0;  E:u=-1.0,v=-1.0
```

Eventually we use the PVD to compute the coefficients of the mass and the stiffness matrices, taking into account the individual masses and stiffnesses of the individual points.

```
M = np.array([[(masses*acc*vd).sum()  for acc  in uv] for vd in uv])
K = np.array([[(stiffs*disp*vd).sum() for disp in uv] for vd in uv])
p = np.array([(forces*vd).sum() for vd in uv])
```

$$\boldsymbol{M} = m\begin{bmatrix} 42.00000 & -22.00000 \\ -22.00000 & 12.00000 \end{bmatrix}$$

$$\boldsymbol{K} = k\begin{bmatrix} 1.00000 & 0.00000 \\ 0.00000 & 1.00000 \end{bmatrix}$$

$$\boldsymbol{p} = \delta k\begin{bmatrix} 6.00000 \\ -3.00000 \end{bmatrix}\sin 2\pi t/T$$

The equations of motion:

$$\begin{cases} +42m\ddot{x}_1 - 22\ddot{x}_2 + kx_1 = +6p(t) \\ -22m\ddot{x}_1 + 12\ddot{x}_2 + kx_2 = -3p(t) \end{cases}$$

We solve the problem of free vibrations using the `eigh()` library function

```
evals, evecs = eigh(K, M)
evecs[:,0] *= -1.0
pstar = evecs.T@p
```

```
Eigenvalues:
```

$$\mathbf{\Lambda} = \begin{bmatrix} 0.01865 & 0.00000 \\ 0.00000 & 2.68135 \end{bmatrix}$$

```
Eigenvectors:
```

$$\mathbf{\Psi} = \begin{bmatrix} 0.12073 & 0.76513 \\ -0.06381 & 1.44773 \end{bmatrix}$$

```
The modal equations of motion:
```

$$\begin{cases} m\ddot{q}_1 + 0.018647kq_1 = 0.915807\delta k \sin 2\pi \frac{t}{T} \\ m\ddot{q}_2 + 2.681353kq_2 = 0.247584\delta k \sin 2\pi \frac{t}{T} \end{cases}$$

Dividing each modal equation by `m` we have

$$\ddot{q}_i + \lambda_i^2 \omega_0^2 q + i = p_i^\star \delta \omega_0^2 \sin \bar{\lambda} \omega_0 t$$

with $\xi_i = C_i \sin \bar{\lambda} \omega_0 t$, substituting and removing the sine we have

$$(\lambda_i^2 - \bar{\lambda}^2) C_i \omega_0^2 = p_i^\star \delta \omega_0^2$$

hence

$$C_i = \delta \frac{p_i^\star}{\lambda_i^2 - \bar{\lambda}^2}$$

and, taking into account that the response starts from rest conditions,

$$q_i = \delta \frac{p_i^\star}{\lambda_i^2 - \bar{\lambda}^2} \left(\sin \bar{\lambda} \omega_0 t - \frac{\bar{\lambda}}{\lambda_i} \sin \lambda_i \omega_0 t \right).$$

Which is the value of $\bar{\lambda}$? We have $2\pi t/T = \bar{\lambda} \omega_0 t$ or

$$\bar{\lambda} = \frac{2\pi}{\omega_0 T} = \frac{2\pi}{\sqrt{14}}$$

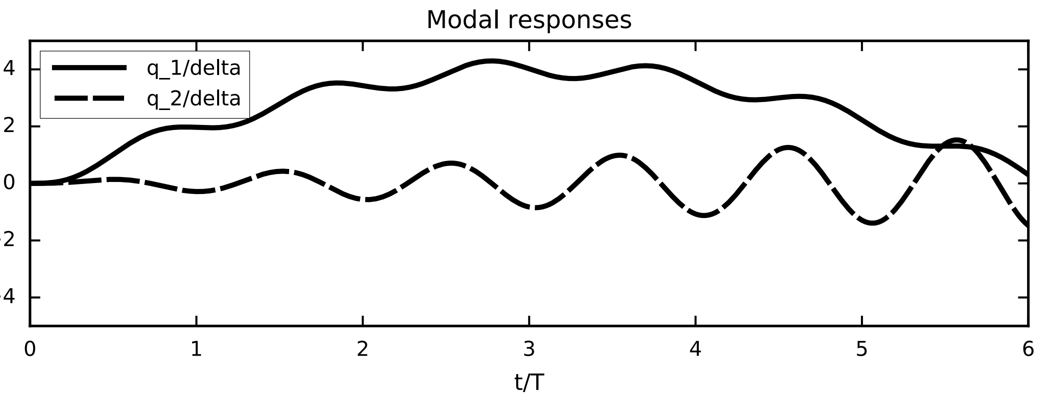
because in the text of the problem we have $(\omega_0 T)^2 = 14$.

```
blambda=2*pi/sqrt(14.0)
den = 1/(evals-blambda**2)
C = pstar*den
beta = blambda/sqrt(evals)
```

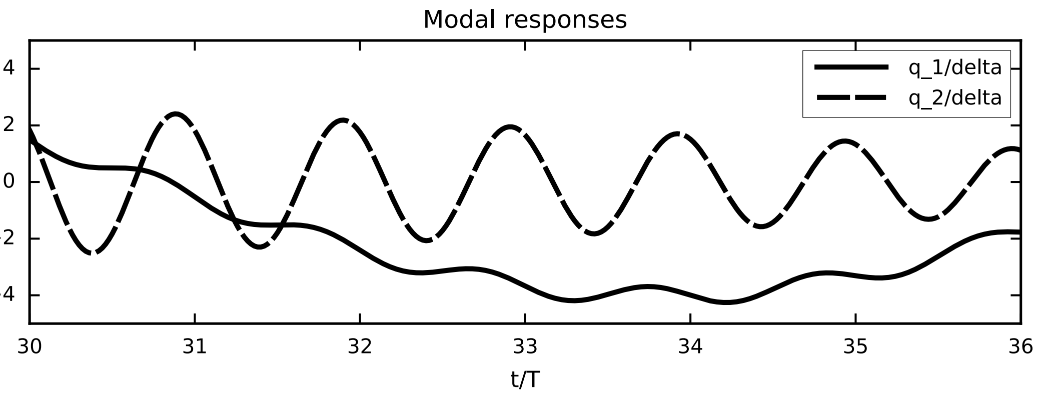
```
q_1 = -0.326929 d (sin(1.679252omega_0·t)-12.297245sin(0.136555omega_0·t))
q_2 = -1.787164 d (sin(1.679252omega_0·t)-1.025508sin(1.637484omega_0·t))
```

$$\begin{cases} q_1 = -0.326929 \left(\sin 2\pi \frac{t}{T} - 12.297245 \sin 0.0813192\pi \frac{t}{T} \right) \\ q_2 = -1.787164 \left(\sin 2\pi \frac{t}{T} - 1.025508 \sin 0.9751272\pi \frac{t}{T} \right) \end{cases}$$

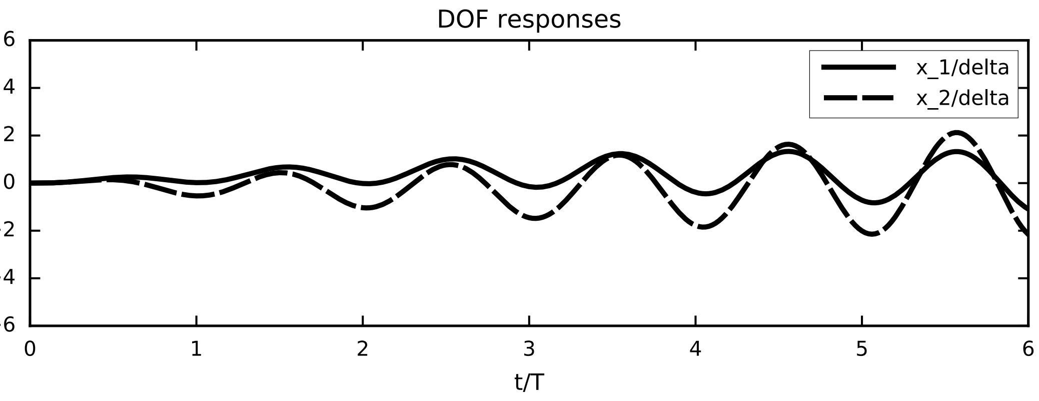
```
#plt.plot(t,-0.326929*(sin(2*pi*t)-12.297245*sin(2*pi*t/beta[0])))
#plt.plot(t,-1.787164*(sin(2*pi*t)- 1.025508*sin(2*pi*t/beta[1])))

t = linspace(0, 36, 1801)[:,None]
q = array(C*(sin(2*pi*t)-beta*sin(2*pi*t/beta)))
x = q@evecs.T
vd = x@array((6,-3))

show(t,  q, title='Modal_responses', legends=['q_1/delta', 'q_2/delta'])
show(t,  x, title='DOF_responses',   legends=['x_1/delta', 'x_2/delta'])
show(t, vd, title='Vertical_displacement_of_D',         legends=['v_D/delta'])
```



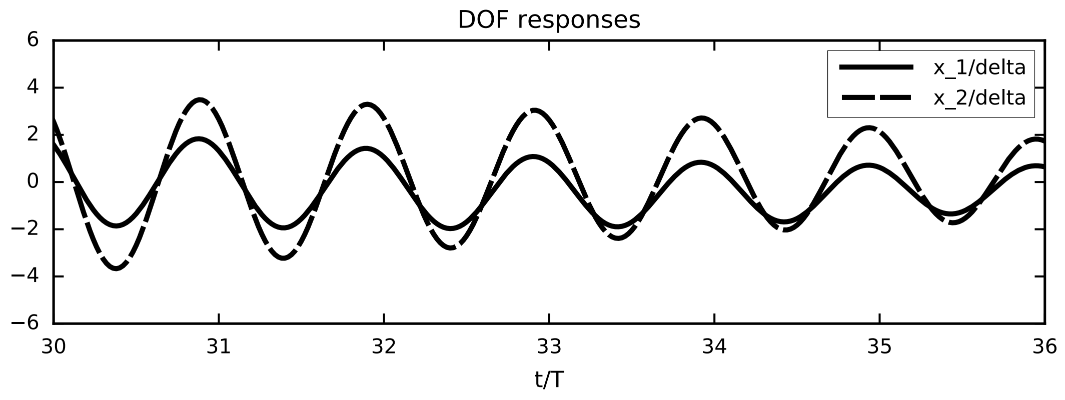
Modal responses
q_1/delta
q_2/delta
t/T

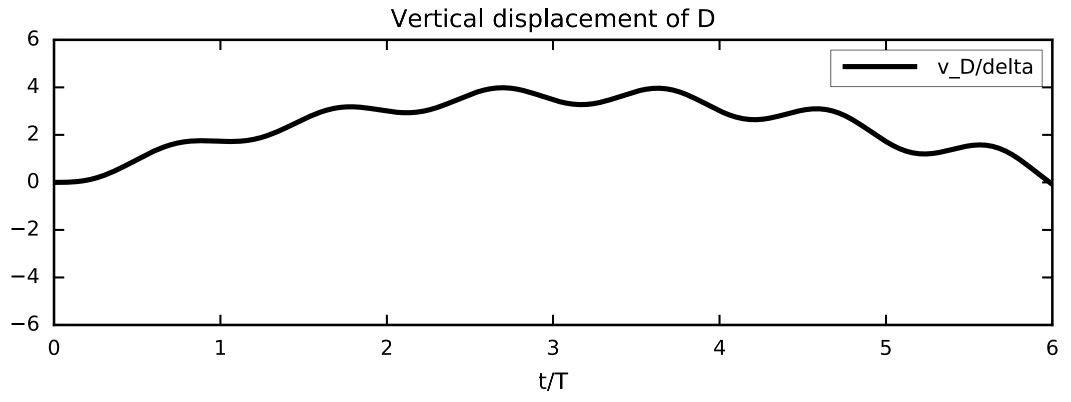


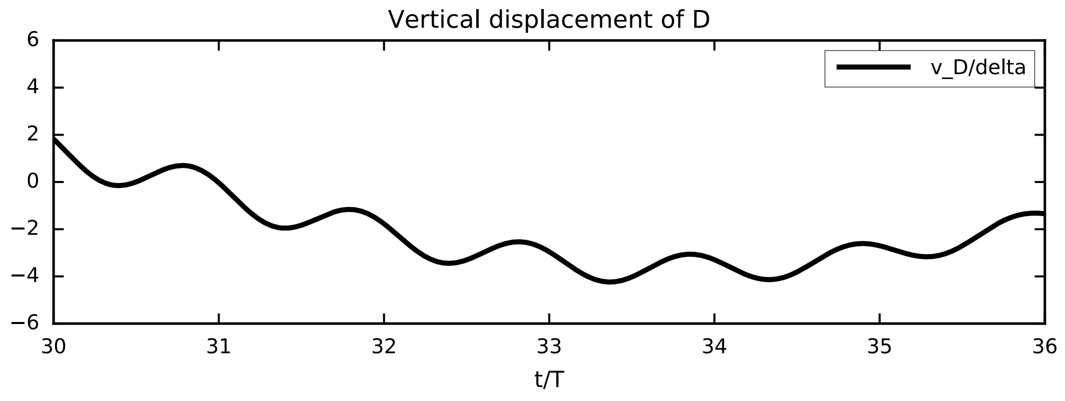
Modal responses
q_1/delta
q_2/delta
t/T



DOF responses
x_1/delta
x_2/delta
t/T







1.1 Initialization Cells

1.1.1 Numpy and Scipy

```
import numpy as np
from numpy import array, cos, linspace, pi, sin, sqrt
from scipy.linalg import eigh
```

1.1.2 Matplotlib

```
%matplotlib inline
%config InlineBackend.figure_formats=['svg']
import matplotlib.pyplot as plt
from cycler import cycler
plt.style.use([
        'seaborn-paper', {
        'axes.grid': True,
        'figure.figsize': (7,2),
        'legend.handlelength': 4.0,
        'axes.prop_cycle': (
                cycler('color', ['k'])*
                cycler('linewidth', [2,1])*
                cycler('dashes', [[],[13,2],[8,3,1,3]])
                ),
        'text.usetex': False}])
```

1.1.3 Utility Functions

The function `material_point()` defines a point as a 2D array, describing its position in the configuration of equilibrium, and a rigid body it belongs to.

The function `O_dr()` defines a small rotation or, as a particular case, a small translation of a rigid body, using a 2D array that gives the CIR or the translation components and a scalar that gives the otation amplitude or it's zero to represent a translation.

The function `disp(p, l_of_rots)` uses a `material_point` structure `p`, containing a position and a body number, and `l_of_Odr`, a list of `O_dr` data structures, one for each body. The displacement of the point is computed using the rotation/translation obtained by indexing `l_of_Odr` by the body number of the material point.

The functions `mass()`, `stif()` and `forc()` build arrays of masses, spring stiffnesses and forces.

```
def material_point(xy=(None, None), body=None):
    return array(xy), body

def O_dr(O, dr):
    return array(O), dr

def disp(mat_point, l_Odr):
    P, body = mat_point
    O, amp = l_Odr[body]
    if amp == 0 : return O
    dx, dy = P-O
    return array((-dy, dx))*amp

def mass(m): return array((m, m))
def stif(kx, ky): return array((kx, ky))
def forc(fx, fy): return array((fx, fy))

def sf(n, fmt="%d"):
    return "+␣" + fmt%n if n>=0 else "-␣" + fmt%(-n)
```

A function to conveniently plot what is required by the text of the problem.

```
def show(x, y, title=None, legends=None):
    for interval in ((0,6), (30,36)):
        lines = plt.plot(x, y)
        plt.xlim(interval)
        plt.xticks(linspace(*interval, 7))
        plt.xlabel('t/T')
        if title:
            plt.title(title)
        if legends:
            for l, legend in zip(lines, legends):
                l.set_label(legend)
            plt.legend(loc='best')
        plt.grid()
        plt.show()
```