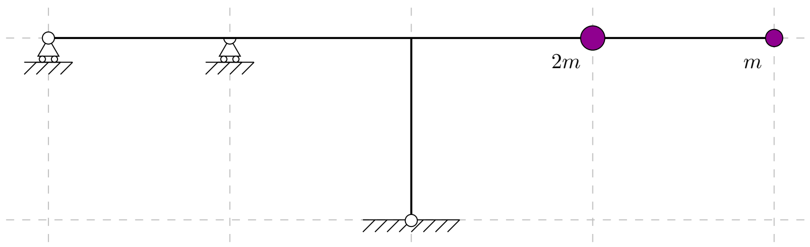
3 DOF System

Giacomo Boffi

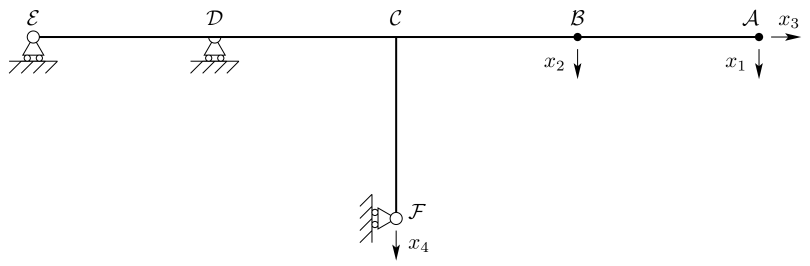
1 3 DOF System



The 3 DOF Dynamic System

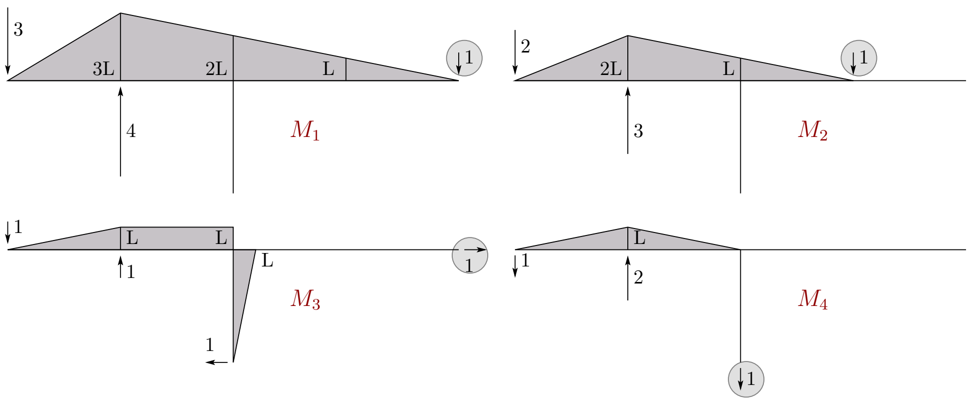
The structure above is statically overdetermined, to compute the stiffness matrix and, later, the influence matrix for the impressed displacement we have to introduce an additional degree of freedom, so that the system becomes statically determined.

Because we need to compute the influence matrix for the vertical displacement of the bottom hinge, we sobstitute the hinge with a roller allowing a vertical displacement and compute the flexibility matrix for the 4 DOF system (the 3 dynamic DOF's + the support displacement) using the PVD. In the figure below you can see the numeration of the DOF's and their positive direction.



The 4 DOF Dynamic System

The first step for computing the flexibilities is to load the structure with an external unit force applied to each of the DOF's and compute (a) the reactions of the rollers and (b) the bending moments:



Bending Moments for Different Load Conditions

We have 5 different fields, listed with the correct orientation:

1. $\overline{\mathcal{AB}}$, $0 \le x \le L$
2. $\overline{\mathcal{BC}}$, $0 \le x \le L$

3. $\overline{\mathcal{CD}}$, $0 \le x \le L$
4. $\overline{\mathcal{ED}}$, $0 \le x \le L$
5. $\overline{\mathcal{FC}}$, $0 \le x \le L$

and for every field we write down, in `L`, the respective length and in each row of `M` (one row for each DOF) the 5 expression for the bending moment on each of the 5 fields, using the univariate polynomial class that is offered by `numpy`, e.g., `p((2,0))` means `2*x**1 + 0*x**0`:

```
L = [1, 1, 1, 1, 1]
BM = [[p((1,0)), p((1,1)), p((1,2)), p((3,0)), p((0,0))],
      [p((0,0)), p((1,0)), p((1,1)), p((2,0)), p((0,0))],
      [p((0,0)), p((0,0)), p((0,1)), p((1,0)), p((1,0))],
      [p((0,0)), p((0,0)), p((1,0)), p((1,0)), p((0,0))]]
```

In the following `M` plays a double role, in one loop it represents the bending moments and in the other loop the curvatures; in the inner loop, for a fixed pair of moments and curvatures, we iterate over the different fields of integration, sum the diverse contributions and eventually we have one coefficient of the flexibility matrix.

That done, we invert the flexibility to have the stiffness matrix for the 4DOF system.

```
F = array([[sum(poly_int(μ, χ, λ) for μ, χ, λ in zip(μ_s, χ_s, L))
            for χ_s in BM] for μ_s in BM])
K = np.linalg.inv(F)

platex('\\overline{\\boldsymbol␣F}␣=␣\\frac{L^3}{6EJ}\\,',
       mat2lat(F*6,     fmt='%6.2f'), ',')
platex('\\overline{\\boldsymbol␣K}␣=␣\\frac{6\,EJ}{161\,L^3}\\,',
       mat2lat(K*161/6, fmt='%6.1f'), '.')
```

$$\overline{\boldsymbol{F}} = \frac{L^3}{6EJ}\begin{bmatrix} 72.00 & 40.00 & 21.00 & 14.00 \\ 40.00 & 24.00 & 13.00 & 9.00 \\ 21.00 & 13.00 & 10.00 & 5.00 \\ 14.00 & 9.00 & 5.00 & 4.00 \end{bmatrix},$$

$$\overline{\boldsymbol{K}} = \frac{6EJ}{161L^3}\begin{bmatrix} 44.0 & -103.0 & 7.0 & 69.0 \\ -103.0 & 296.0 & -42.0 & -253.0 \\ 7.0 & -42.0 & 56.0 & -0.0 \\ 69.0 & -253.0 & -0.0 & 368.0 \end{bmatrix}.$$

From the stiffness matrix, using partitioning, we have the structural stiffness matrix $\boldsymbol{K}_{ss}$ etc; the structural mass matrix is diagonal, for the vertical DOFs the diagonal terms are equal to the supported masses, respectively m and $2m$ while for the horizontal DOF the diagonal term is equal to the sum of the supported masses, namely $3m$.

```
Mss = array(((1.0, 0.0, 0.0), (0.0, 2.0, 0.0), (0.0, 0.0, 3.0)))
platex('{\\boldsymbol␣M}_{ss}␣=␣␣m\\,', mat2lat(Mss, fmt='%8.2f'), '.')
Kss, Ksg = K[:-1,:-1], K[:-1,-1:]
Kgs, Kgg = K[-1:,:-1], K[-1:,-1:]
platex('{\\boldsymbol␣K}_{ss}␣=␣\\frac{6EJ}{161L^3}\\,',
       mat2lat(Kss*161/6,fmt='%8.2f'), ',\\qquad',
       '{\\boldsymbol␣K}_{sg}␣=␣\\frac{6EJ}{161L^3}\\,',
       mat2lat(Ksg*161/6,fmt='%8.2f'), ',')
platex('{\\boldsymbol␣K}_{gs}␣=␣\\frac{6EJ}{161L^3}\\,',
       mat2lat(Kgs*161/6,fmt='%9.4f'), ',\\qquad',
       '{\\boldsymbol␣K}_{gg}␣=␣\\frac{6EJ}{161L^3}\\,',
       mat2lat(Kgg*161/6,fmt='%8.2f'), ',')
```

$$\boldsymbol{M}_{ss} = m\begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.00 & 2.00 & 0.00 \\ 0.00 & 0.00 & 3.00 \end{bmatrix}.$$

$$\boldsymbol{K}_{ss} = \frac{6EJ}{161L^3}\begin{bmatrix} 44.00 & -103.00 & 7.00 \\ -103.00 & 296.00 & -42.00 \\ 7.00 & -42.00 & 56.00 \end{bmatrix}, \qquad \boldsymbol{K}_{sg} = \frac{6EJ}{161L^3}\begin{bmatrix} 69.00 \\ -253.00 \\ -0.00 \end{bmatrix},$$

$$\boldsymbol{K}_{gs} = \frac{6EJ}{161L^3}\begin{bmatrix} 69.0000 & -253.0000 & -0.0000 \end{bmatrix}, \qquad \boldsymbol{K}_{gg} = \frac{6EJ}{161L^3}\begin{bmatrix} 368.00 \end{bmatrix},$$

eventually we compute the eigenvalues and the eigenvectors of the dynamic system, print them

```
om2, Psi = eigh(Kss, Mss)

platex('{\\boldsymbol\\Lambda}␣=␣\\omega_0^2\\,', mat2lat(om2[None,:]))
platex('{\\boldsymbol\\Psi}␣=␣', mat2lat(Psi))
```

$$\Lambda=\omega_0^2\begin{bmatrix}0.18659 & 0.68702 & 6.97733\end{bmatrix}$$

$$\Psi=\begin{bmatrix}0.82348 & -0.34127 & -0.45322\\ 0.32482 & -0.04792 & 0.62626\\ 0.19224 & 0.54128 & -0.05829\end{bmatrix}$$

as well as a check of the ortogonality of the eigenvectors:

```
platex('{\\boldsymbol\\Psi}^T␣{\\boldsymbol␣M}_{ss}␣{\\boldsymbol\\Psi}␣=␣m\\,',
       mat2lat(Psi.T @ Mss @ Psi),',')
platex('{\\boldsymbol\\Psi}^T␣{\\boldsymbol␣K}_{ss}␣{\\boldsymbol\\Psi}␣=␣\\frac{EJ}{L^3}\\,',
       mat2lat(Psi.T @ Kss @ Psi),'.')
```

$$\Psi^T\boldsymbol{M}_{ss}\Psi=m\begin{bmatrix}1.00000 & -0.00000 & -0.00000\\ 0.00000 & 1.00000 & 0.00000\\ -0.00000 & 0.00000 & 1.00000\end{bmatrix},$$

$$\Psi^T\boldsymbol{K}_{ss}\Psi=\frac{EJ}{L^3}\begin{bmatrix}0.18659 & -0.00000 & -0.00000\\ -0.00000 & 0.68702 & 0.00000\\ 0.00000 & 0.00000 & 6.97733\end{bmatrix}.$$

2 Modal Response for EQ Excitation

The equation of motion, writing it in terms of adimensional matrices, is

$$m\boldsymbol{M}\ddot{x}(t)+\frac{EJ}{L^3}\boldsymbol{K}\boldsymbol{x}=-m\boldsymbol{M}\boldsymbol{e}\,a_0f(t).$$

If we introduce an adimensional time coordinate, defined in terms of the structural characteristics parameter $\omega_0^2=EJ/mL^3$,

$$\tau=\frac{t}{t_0}\text{ with }\omega_0t_0=\alpha$$

substituting into the equation of motion and taking into account that

$$\frac{\mathrm{d}^2x(\tau)}{\mathrm{d}t^2}=\frac{1}{t_0^2}\frac{\mathrm{d}^2x(\tau)}{\mathrm{d}\tau^2}$$

we have, by successive manipulations

$$m\frac{1}{t_0^2}\boldsymbol{M}\ddot{x}(\tau)+\frac{EJ}{L^3}\boldsymbol{K}\boldsymbol{x}(\tau)=-m\boldsymbol{M}\boldsymbol{e}\frac{d_0}{t_0^2}f(\tau)$$

$$\boldsymbol{M}\ddot{x}(\tau)+\frac{EJ}{mL^3}t_0^2\leq\boldsymbol{K}\boldsymbol{x}(\tau)=-\boldsymbol{M}\boldsymbol{e}\,d_0f(\tau)$$

$$\boldsymbol{M}\ddot{x}(\tau)+\alpha^2\boldsymbol{K}\boldsymbol{x}(\tau)=-\boldsymbol{M}\boldsymbol{e}\,d_0f(\tau).$$

If we focus on the associated free vibration problem, the eigenvectors do not change with respect to the problem formulated in t while the eigenvalues are scaled by α^2/ω_0^2.

Using modal coordinates with the understanding that the eigenvectors are mass-matrix normalized ($\Psi^T\boldsymbol{M}\Psi=\boldsymbol{I}$)

$$\ddot{\boldsymbol{q}}+\alpha^2\Lambda\boldsymbol{q}=-\Psi^T\boldsymbol{M}\boldsymbol{e}\,d_0f(t)=\Gamma\,d_0f(t)$$

and the individual modal responses can be written

$$\frac{q_i}{d_0}=A_i\sin\alpha\lambda_i\tau+B_i\cos\alpha\lambda_i\tau+\xi_i(\tau)$$

with ξ a particular integral that satisfies

$$\ddot{\xi}_i+\alpha^2\lambda_i^2\xi=g_if(\tau).$$

For

$$f(\tau)=\sum_0^n f_j\tau^j$$

it is

$$\xi_i(\tau)=\sum_0^n x_{i,j}\tau^j,\qquad\ddot{\xi}_i(\tau)=\sum_0^{n-2}(j+1)(j+2)x_{i,j+2}\tau^j,$$

Substituting in the equation of motion and equating the coefficients for t^j we have

$$\alpha^2\lambda_i^2x_{i,j}+(j+1)(j+2)x_{i,j+2}=g_if_j.$$

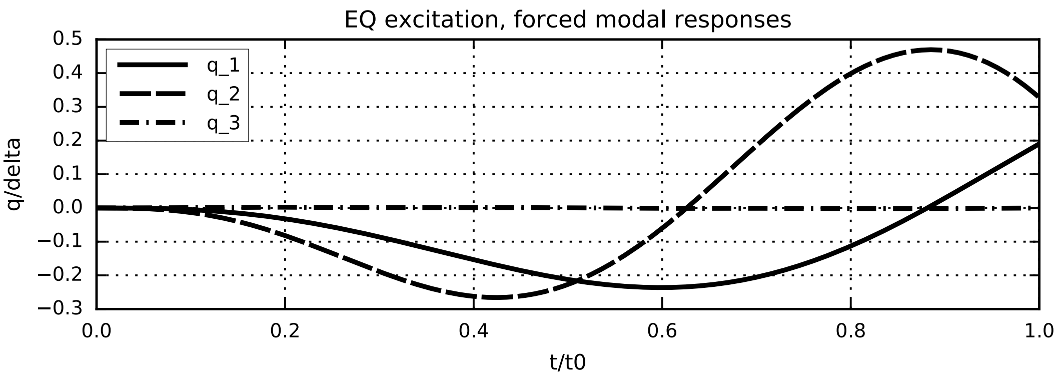
Because $x_{i,n+1} = x_{i,n+2} = 0$ the coefficients $x_{i,j}$ can be easily computed starting from $j = n$:

$$x_{i,n} = g_i f_n / (\alpha^2 \lambda_i^2)$$
$$x_{i,n-1} = g_i f_{n-1} / (\alpha^2 \lambda_i^2)$$
$$x_{i,n-2} = (g_i f_{n-2} - n(n-1) x_{i,n}) / (\alpha^2 \lambda_i^2)$$
$$\vdots$$

Considering a system that starts from rest conditions, the constants of integrations are simply

$$A_i = -\frac{x_{i,1}}{\alpha \lambda_i}, \quad B_i = -x_{i,0}.$$

```
a = 3*pi
a2 = a**2

M, K = Mss, a2*Kss

e = array((0.0, 0.0, 1.0))
Gamma = -Psi.T @ M @ e
lambda_i = sqrt(om2)
P0 = p((120.0, -180.0, 60.0, 0.0))

kappa_i = a2 * lambda_i**2
coefs = [particular_poly(P0*g, 1.0, 0.0, k) for k, g in zip(kappa_i, Gamma)]
B_i = [-coeff(0) for coeff in coefs]
A_i = [-coeff.deriv(1)(0)/(a*l) for coeff, l in zip(coefs, lambda_i)]

tau01 = linspace(0.0, 1.0, 201)
s01, c01 = sin(a*lambda_i*tau01[:,None]), cos(a*lambda_i*tau01[:,None])

q01 = A_i*s01 + B_i*c01 + array([coeff(tau01) for coeff in coefs]).T
v01 = a*lambda_i*(A_i*c01-B_i*s01)
v01 = v01 + array([coeff.deriv(1)(tau01) for coeff in coefs]).T
lines = plt.plot(tau01, q01)
for i, l in enumerate(lines, 1): l.set_label('q_%d'%i)
plt.xlabel('t/t0')
plt.ylabel('q/delta')
plt.title('EQ␣excitation,␣forced␣modal␣responses')
plt.legend(loc='best');
```



This solution works for $0 \le \tau \le 1$, in presence of the input acceleration, for $1 \le \tau$ the system freely vibrates, with

$$\frac{q_i^\star(\tau)}{\delta} = C_i \sin n\pi\lambda_i\tau + D_i \cos n\pi\lambda_i\tau$$

we have

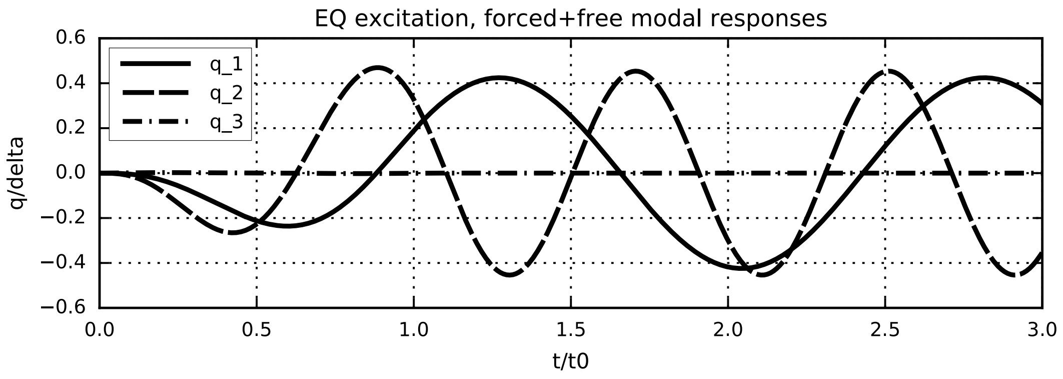
$$\begin{bmatrix} \mathcal{S}_i & \mathcal{C}_i \\ \mathcal{C}_i & -\mathcal{S}_i \end{bmatrix} \begin{Bmatrix} C_i \\ D_i \end{Bmatrix} = \frac{1}{\delta} \begin{Bmatrix} q_i(1) \\ \frac{\dot{q}_i(1)}{n\pi\lambda_i} \end{Bmatrix}$$

where $\mathcal{S}_i = \sin n\pi\lambda_i\tau|_{\tau=1} = \sin n\pi\lambda_i$ and $\mathcal{C}_i = \cos n\pi\lambda_i$.

The cofficient matrix is orthogonal and symmetric, hence

$$\begin{Bmatrix}C_i\\D_i\end{Bmatrix}=\frac{1}{\delta}\begin{bmatrix}\mathcal{S}_i & \mathcal{C}_i\\ \mathcal{C}_i & -\mathcal{S}_i\end{bmatrix}\begin{Bmatrix}q_i(1)\\ \frac{\dot q_i(1)}{n\pi\lambda_i}\end{Bmatrix}$$

```
q1, v1 = q01[-1,:], v01[-1,:]
s1, c1 = s01[-1,:], c01[-1,:]

C_i = s1*q1 + c1*v1/(a*lambda_i)
D_i = c1*q1 - s1*v1/(a*lambda_i)

tau13 = linspace(1.0, 3.0, 401)
s13, c13 = sin(a*lambda_i*tau13[:,None]), cos(a*lambda_i*tau13[:,None])
q13, v13 = C_i*s13 + D_i*c13, a*lambda_i*(C_i*c13-D_i*s13)

tau03 = np.hstack((tau01, tau13[1:]))
q03 = np.vstack((q01, q13[1:]))
v03 = np.vstack((v01, v13[1:]))

lines = plt.plot(tau03, q03)
for i, l in enumerate(lines, 1): l.set_label('q_%d'%i)
plt.xlabel('t/t0')
plt.ylabel('q/delta')
plt.title('EQ␣excitation,␣forced+free␣modal␣responses')
plt.legend(loc='best');
```



2.1 Analytic espressions of the q_i

We can compute the analytical expressions of the modal responses:

```
for i in (0, 1, 2):
    l = a*lambda_i[i]
    A, B, C, D = A_i[i], B_i[i], C_i[i], D_i[i]
    qqq = r'\small{q_%d(\tau)␣=␣'%(i+1)
    beg = r'\begin{cases}'
    eq1 = r'%+6.2g\sin(%5.2f\tau)%+6.2f\cos(%5.2f\tau)'%(A, l, B, l)
    eq1 = eq1 + ''.join('%+5.2f\\,\\tau^{%d}'%(c,3-j) if j<2 else ('%+5.2f\\,\\tau' if j==2 else '%+5.2f')%c f
    in1 = r'0\le\tau\le1\\'
    eq2 = r'%+f\sin(%f\tau)%+f\cos(%f\tau)&'%(C, l, D, l)
    in2 = r'1\le\tau.'
    end = r'\end{cases}}'
    platex(qqq,beg,eq1,in1,eq2,in2,end)
    print (',␣'.join('%g'%c for c in [A, B] + list(coefs[i])))
```

$$q_1(\tau)=\begin{cases}+0.14\sin(4.07\tau)+0.76\cos(4.07\tau)-4.18\tau^3+6.26\tau^2-0.58\tau-0.76 & 0\le\tau\le1\\ -0.379464\sin(4.071096\tau)+0.190256\cos(4.071096\tau) & 1\le\tau.\end{cases}$$

```
0.141525, 0.755817, -4.17559, 6.26339, -0.576162, -0.755817
```

$$q_2(\tau) = \begin{cases} +0.16\sin(7.81\tau)+0.16\cos(7.81\tau)-3.19\tau^3+4.79\tau^2-1.28\tau-0.16 & 0\le\tau\le 1 \\ +0.314107\sin(7.811864\tau)+0.327623\cos(7.811864\tau) & 1\le\tau. \end{cases}$$

```
0.164187, 0.156973, -3.1931, 4.78965, -1.2826, -0.156973
```

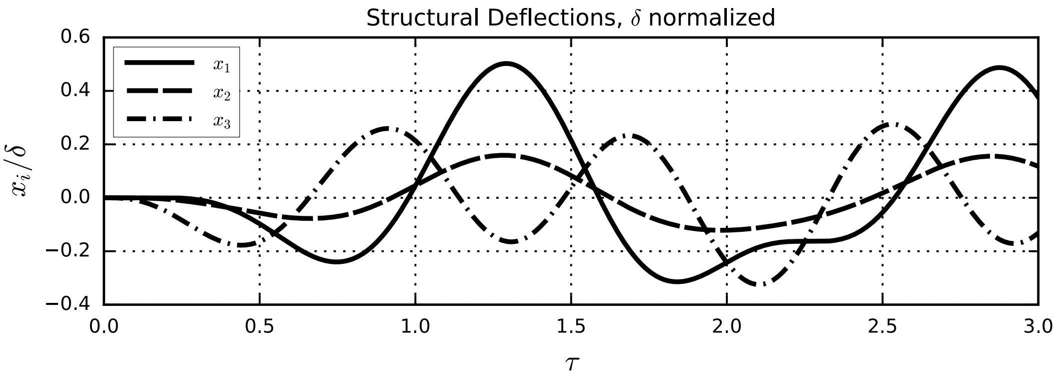
$$q_3(\tau) = \begin{cases} -0.00067\sin(24.90\tau)-0.00\cos(24.90\tau)+0.03\tau^3-0.05\tau^2+0.02\tau+0.00 & 0\le\tau\le 1 \\ +0.000020\sin(24.895206\tau)-0.000166\cos(24.895206\tau) & 1\le\tau. \end{cases}$$

```
-0.000666841, -0.000163889, 0.0338578, -0.0507867, 0.0166011, 0.000163889
```

2.2 Plotting the x_i

This is easy as the line of code `x = (Psi@q)` ...

```
x = (Psi@q03.T)
lines = plt.plot(tau03, x.T)
for i, l in enumerate(lines): l.set_label('$x_%d$'%(i+1))
plt.title('Structural␣Deflections,␣$\\delta$␣normalized')
plt.xlabel(r'$\tau$', fontsize='large')
plt.ylabel(r'$x_i/\delta$', fontsize='large')
plt.legend(loc='best');
```



3 Numerical Integration

Using the *Constant Acceleration Algorithm*, we use the same time step used in the time array `t03` to compute all the (matricial) constants, etc.

```
h = 1/200 # time step
dK = K+ 4*M/h/h # the modified stiffness
dF = np.linalg.inv(dK) # ... and its inverse
# these are the v0, a0 coefficients to correct dp, the incremental load
dC, dM = 4*M/h, 2*M
mass_v = -M @ e
Minv = np.linalg.inv(M)
a_g = np.where(tau03<1, P0(tau03), 0.0) # the horizontal ground acceleration
```

Let's initialize the different vectors, as well as the containers with the output

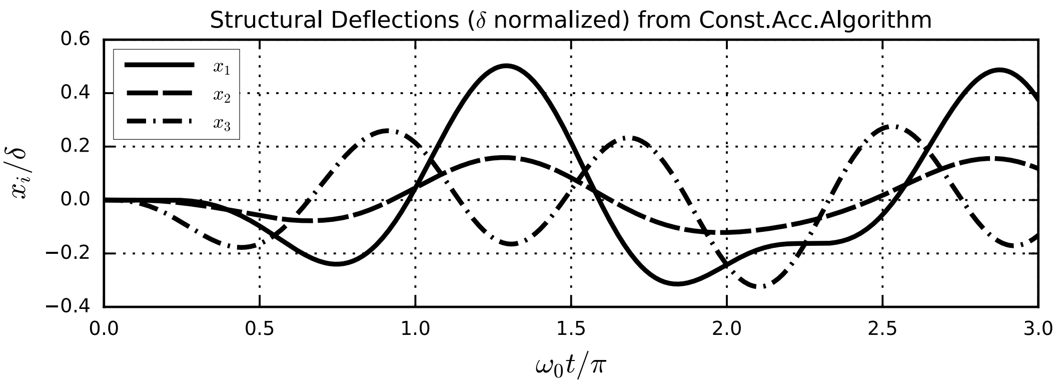
```
# initial values
x0, v0, ag0 = np.zeros(3), np.zeros(3), a_g[0]
a0 = Minv @ (mass_v*ag0 - K @ x0)

# initialize the deflections, velocities and accelerations
X, V, A = [x0], [v0], [a0]
```

Finally, the integration proper

```
for ag0, ag1 in zip(a_g[:-1], a_g[+1:]):
    dx = dF@(mass_v*(ag1-ag0) + dC @ v0 + dM @ a0)
    x0 = dx + x0
    v0 = 2*dx/h - v0
    a0 = Minv@(mass_v*ag1 - K @ x0)
    X.append(x0), V.append(v0), A.append(a0)
```

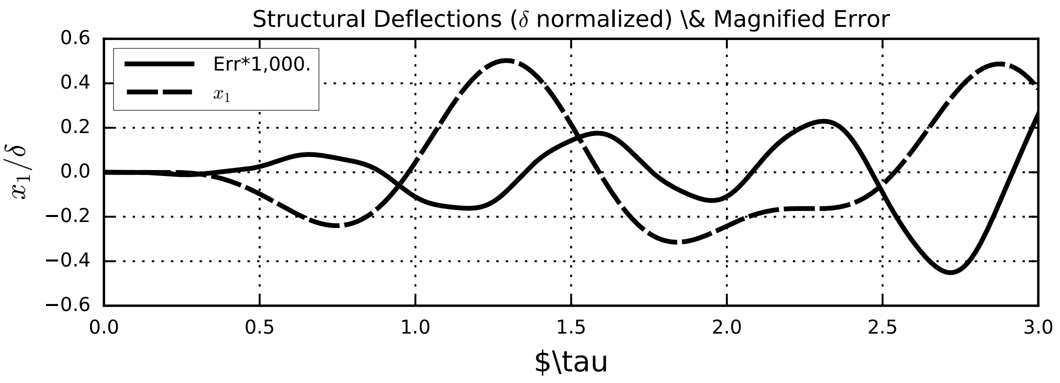
Now, we have just to plot our results, looking at them and comparing with the analytic solution it seems that we have a good agreement,

```
X = array(X)

lines = plt.plot(tau03, X)
for i, l in enumerate(lines): l.set_label('$x_%d$'%(i+1))
plt.title('Structural␣Deflections␣($\\delta$␣normalized)␣from␣Const.Acc.Algorithm')
plt.xlabel(r'$\omega_0t/\pi$', fontsize='large')
plt.ylabel(r'$x_i/\delta$', fontsize='large')
plt.legend(loc='best');
```

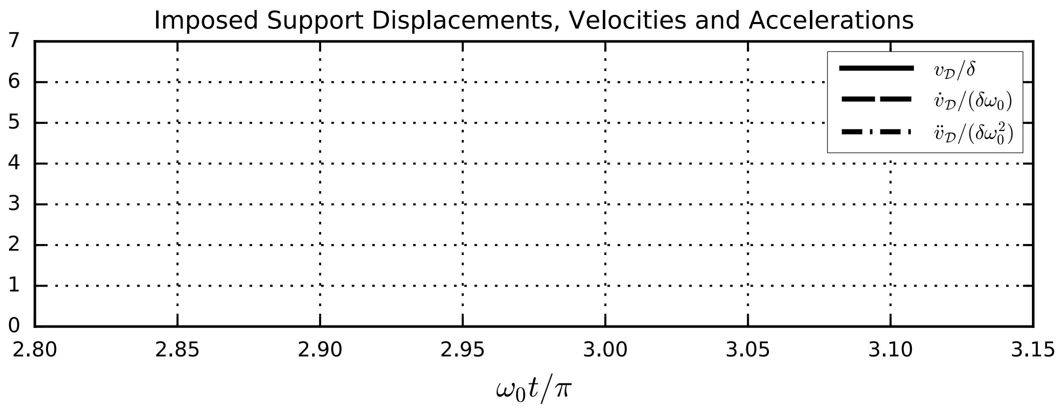


but we can also plot just `x_1` and the difference between the analytic solution and the numerical one to appreciate how good is the numerical solution.

```
plt.plot(tau03, 1000*(X[:,0]-x[0]), label='Err*1,000.')
plt.plot(tau03, x[0], label='$x_1$')
plt.title('Structural␣Deflections␣($\\delta$␣normalized)␣\\&␣Magnified␣Error')
plt.xlabel(r'$\tau', fontsize='large')
plt.ylabel(r'$x_1/\delta$', fontsize='large')
plt.legend(loc='best');
```



```
plt.plot(a/pi, np.where(a<2*pi, a-sin(a), 2*pi), label=r'$v_{\cal_D}/\delta$')
plt.plot(a/pi, np.where(a<2*pi, 1-cos(a),    0), label=r'$\dot{v}_{\cal_D}/(\delta\omega_0)$')
plt.plot(a/pi, np.where(a<2*pi,   sin(a),    0), label=r'$\ddot{v}_{\cal_D}/(\delta\omega_0^2)$')
plt.title('Imposed␣Support␣Displacements,␣Velocities␣and␣Accelerations')
plt.xlabel(r'$\omega_0t/\pi$', fontsize='large')
plt.legend(loc='best');
```



4 Support Motion

The structure is at rest when it is subjected to a vertical motion of the bottom hinge $v+\mathcal{F}$,

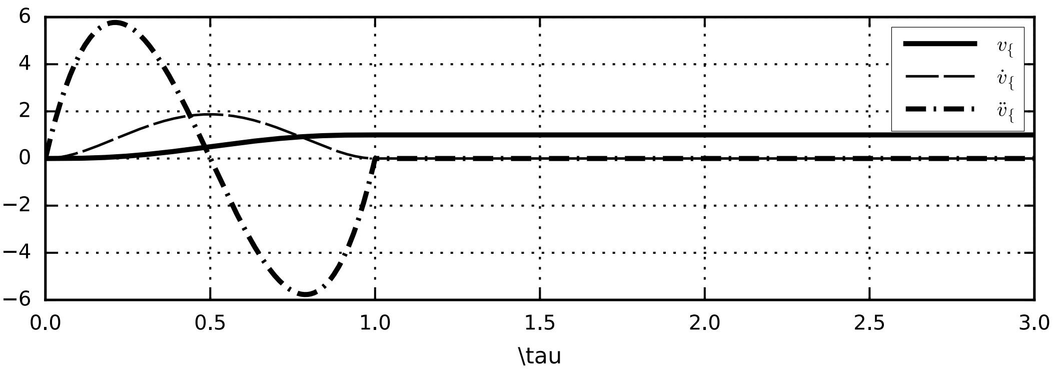
$$v_{\mathcal{F}}(t)=\delta\begin{cases}0 & \tau<0,\\ 6\tau^5-15\tau^4+10\tau^3 & 0\le\tau\le 1,\\ 1 & 1<\tau.\end{cases}$$

1. Plot $v_{\mathcal{F}}(t)$, $\dot{v}_{\mathcal{F}}(t)$ and $\ddot{v}_{\mathcal{F}}(t)$ in the interval $0\le\tau\le 3$.
2. Compute and plot the total vertical displacement of the smaller mass in the same time interval, using the modal superposition procedure.

4.1 Plot of the displacements, velocities and accelerations.

```
df = p((6.0, -15.0, 10.0, 0.0, 0.0, 0.0), variable='τ')
vf = p(df.deriv(), variable='τ')
af = p(vf.deriv(), variable='τ')

df03 = np.where(tau03<=1, df(tau03), 1.0)
vf03 = np.where(tau03<=1, vf(tau03), 0.0)
af03 = np.where(tau03<=1, af(tau03), 0.0)

plt.plot(tau03, df03, label=r'$v_{\cal_f}$')
plt.plot(tau03, vf03, label=r'$\dotv_{\cal_f}$', lw=1)
plt.plot(tau03, af03, label=r'$\ddotv_{\cal_f}$')
plt.legend(loc='best')
plt.xlabel('\\tau')
print(df, '␣␣␣␣␣displacement')
print(vf, '␣␣␣␣velocity')
print(af, '␣␣acceleration')
```

```
   5      4      3
6 Ï - 15 Ï + 10 Ï      displacement
    4      3      2
30 Ï - 60 Ï + 30 Ï     velocity
     3       2
120 Ï - 180 Ï + 60 Ï   acceleration
```



4.2 Modal Response

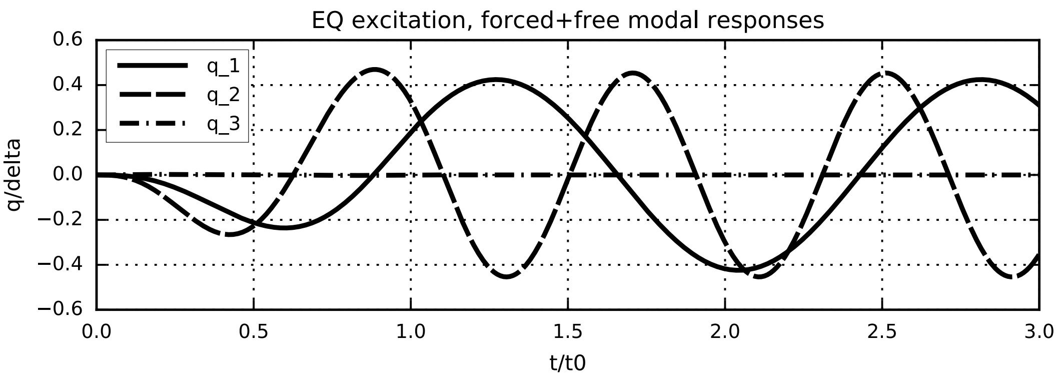
Previously we had $\boldsymbol{e}=\{0\,0\,1\}^T$, now it is $\boldsymbol{e}=\{14/4\,9/4\,5/4\}^T$ (as can be seen from the last column of the 4×4 flexibility matrix) *and this is the only difference between this part of the problem and the previous one.*

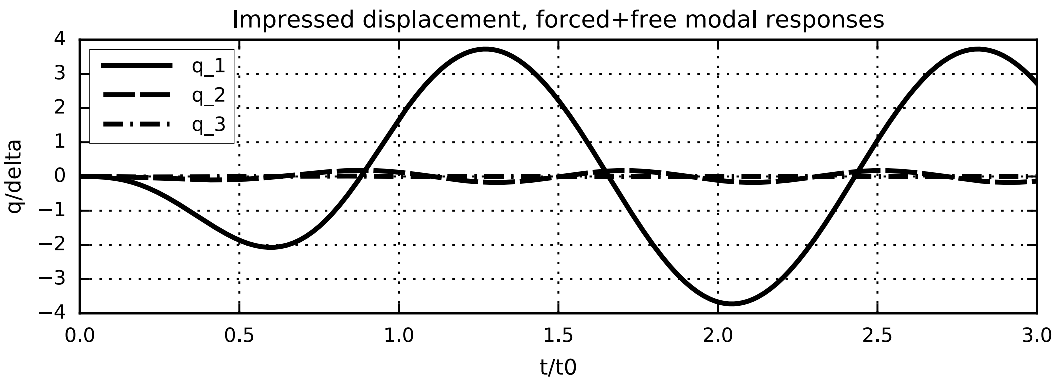
```
e = F[:-1,-1]/F[-1,-1]
print('e_=', e*4, '/_4')
```

```
e = [ 14.   9.   5.] / 4
```

```
Gammad = -Psi.T @ M @ ed
print(Gamma, Gammad)
coefsd = [particular_poly(P0*g, 1.0, 0.0, k) for k, g in zip(kappa_i, Gammad)]
Bd_i = [-coeff(0) for coeff in coefsd]
Ad_i = [-coeff.deriv(1)(0)/(a*l) for coeff, l in zip(coefsd, lambda_i)]
qd01 = Ad_i*s01 + Bd_i*c01 + array([coeff(tau01) for coeff in coefsd]).T
vd01 = a*lambda_i*(Ad_i*c01-Bd_i*s01)
vd01 = vd01 + array([coeff.deriv(1)(tau01) for coeff in coefsd]).T

qd1, vd1 = qd01[-1,:], vd01[-1,:]

Cd_i = s1*qd1 + c1*vd1/(a*lambda_i)
Dd_i = c1*qd1 - s1*vd1/(a*lambda_i)

qd13, vd13 = Cd_i*s13 + Dd_i*c13, a*lambda_i*(Cd_i*c13-Dd_i*s13)

qd03 = np.vstack((qd01, qd13[1:]))
vd03 = np.vstack((vd01, vd13[1:]))

lines = plt.plot(tau03, q03)
for i, l in enumerate(lines, 1): l.set_label('q_%d'%i)
plt.xlabel('t/t0')
plt.ylabel('q/delta')
plt.title('EQ_excitation,_forced+free_modal_responses')
plt.legend(loc='best');
plt.show()

lines = plt.plot(tau03, qd03)
for i, l in enumerate(lines, 1): l.set_label('q_%d'%i)
plt.xlabel('t/t0')
plt.ylabel('q/delta')
plt.title('Impressed_displacement,_forced+free_modal_responses')
plt.legend(loc='best');
```

```
[-0.57671262 -1.62383001  0.17486753] [-5.06474972 -0.61970069 -1.01330217]
```





4.2.1 The modal responses, analytic expressions

```
for i in (0, 1, 2):
    l = a*lambda_i[i]
    A, B, C, D = Ad_i[i], Bd_i[i], Cd_i[i], Dd_i[i]
    qqq = r'\small{q_%d(\tau)␣=␣'%(i+1)
    beg = r'\begin{cases}'
    eq1 = r'%+6.2g\sin(%5.2f\tau)%+6.2f\cos(%5.2f\tau)'%(A, l, B, l)
    eq1 = eq1 + ''.join('%+5.2f\\,\\tau^{%d}'%(c,3-j) if j<2 else ('%+5.2f\\,\\tau' if j==2 else '%+5.2f')%c
    in1 = r'0\le\tau\le1\\'
    eq2 = r'%+f\sin(%f\tau)%+f\cos(%f\tau)&'%(C, l, D, l)
    in2 = r'1\le\tau.'
    end = r'\end{cases}}'
    platex(qqq,beg,eq1,in1,eq2,in2,end)
    print (',␣'.join('%g'%c for c in [A, B] + list(coefsd[i])))
```

$$q_1(\tau)=\begin{cases}+1.2\sin(4.07\tau)+6.64\cos(4.07\tau)-36.67\tau^3+55.01\tau^2-5.06\tau-6.64 & 0\le\tau\le1\\ -3.332488\sin(4.071096\tau)+1.670845\cos(4.071096\tau) & 1\le\tau.\end{cases}$$

```
1.24289, 6.63766, -36.6705, 55.0057, -5.05991, -6.63766
```

$$q_2(\tau)=\begin{cases}+0.063\sin(7.81\tau)+0.06\cos(7.81\tau)-1.22\tau^3+1.83\tau^2-0.49\tau-0.06 & 0\le\tau\le1\\ +0.119872\sin(7.811864\tau)+0.125031\cos(7.811864\tau) & 1\le\tau.\end{cases}$$

```
0.0626584, 0.0599054, -1.21858, 1.82787, -0.489479, -0.0599054
```

$$q_3(\tau) = \begin{cases} +0.0039\sin(24.90\tau)+0.00\cos(24.90\tau)-0.20\tau^3+0.29\tau^2-0.10\tau-0.00 & 0\le\tau\le1 \\ -0.000115\sin(24.895206\tau)+0.000963\cos(24.895206\tau) & 1\le\tau. \end{cases}$$

```
0.00386413, 0.000949683, -0.196195, 0.294293, -0.0961983, -0.000949683
```

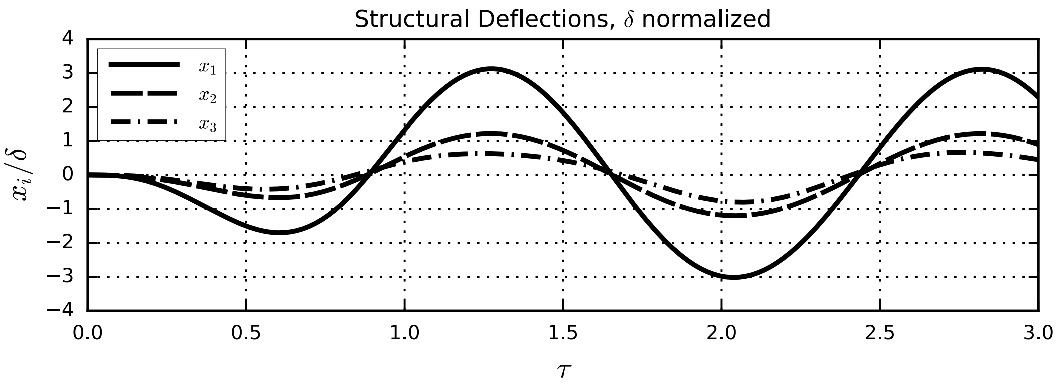
4.3 Structural response

We have to perform the following computations

1. from the modal response compute the dynamic component of structural displacements,
2. using the influence matrix, compute the static component,
3. summing the dynamic and the static components, compute the total displacements.

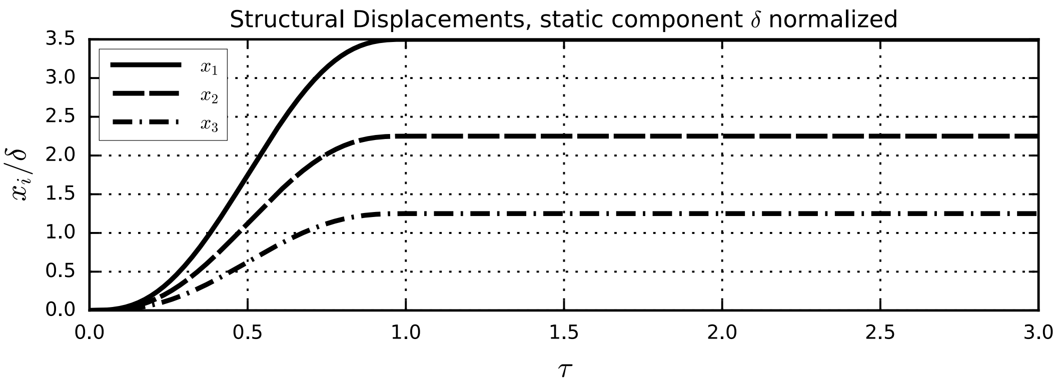
4.3.1 The dynamic component

```
xd = (Psi@qd03.T)
lines = plt.plot(tau03, xd.T)
for i, l in enumerate(lines): l.set_label('$x_%d$'%(i+1))
plt.title('Structural␣Deflections,␣$\\delta$␣normalized')
plt.xlabel(r'$\tau$', fontsize='large')
plt.ylabel(r'$x_i/\delta$', fontsize='large')
plt.legend(loc='best');
```



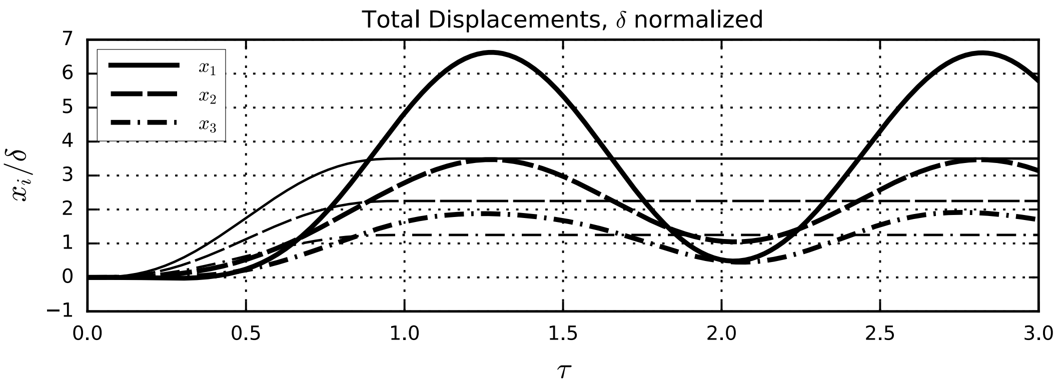
4.3.2 The static component

```
xs = e[:,None] * np.where(tau03<=1, df(tau03), 1)
lines = plt.plot(tau03, xs.T)
for i, l in enumerate(lines): l.set_label('$x_%d$'%(i+1))
plt.title('Structural␣Displacements,␣static␣component␣$\\delta$␣normalized')
plt.xlabel(r'$\tau$', fontsize='large')
plt.ylabel(r'$x_i/\delta$', fontsize='large')
plt.legend(loc='best');
```



4.3.3 Total displacements

```
xt = xd+xs
lines = plt.plot(tau03, xt.T)
for i, l in enumerate(lines): l.set_label('$x_%d$'%(i+1))
plt.plot(tau03, xs.T)
plt.title('Total␣Displacements,␣$\\delta$␣normalized')
plt.xlabel(r'$\tau$', fontsize='large')
plt.ylabel(r'$x_i/\delta$', fontsize='large')
plt.legend(loc='best');
```


Total Displacements, δ normalized
x_1
x_2
x_3
x_i/δ
τ