06_MDOF_System

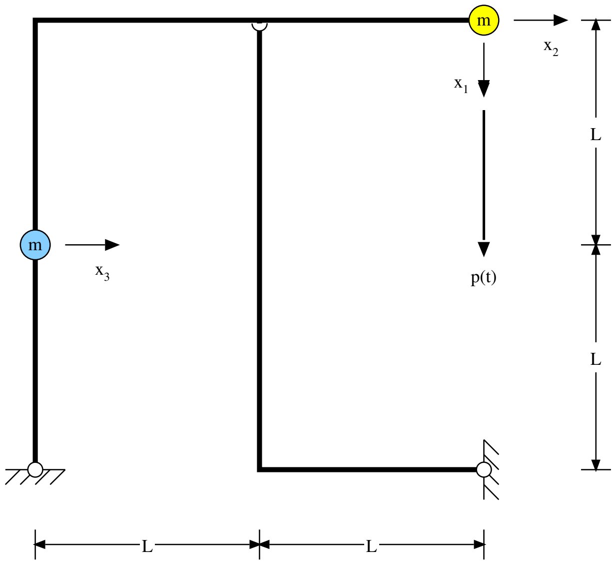
May 23, 2014

```
%matplotlib inline
from scipy import *
from IPython.display import SVG, display, Latex
import matplotlib
import matplotlib.pylab as pl
```

1 MDOF System, Modal Analysis

We have to study the structure in the figure below (btw, one mass is yellow and other is blue because they're M&M's).

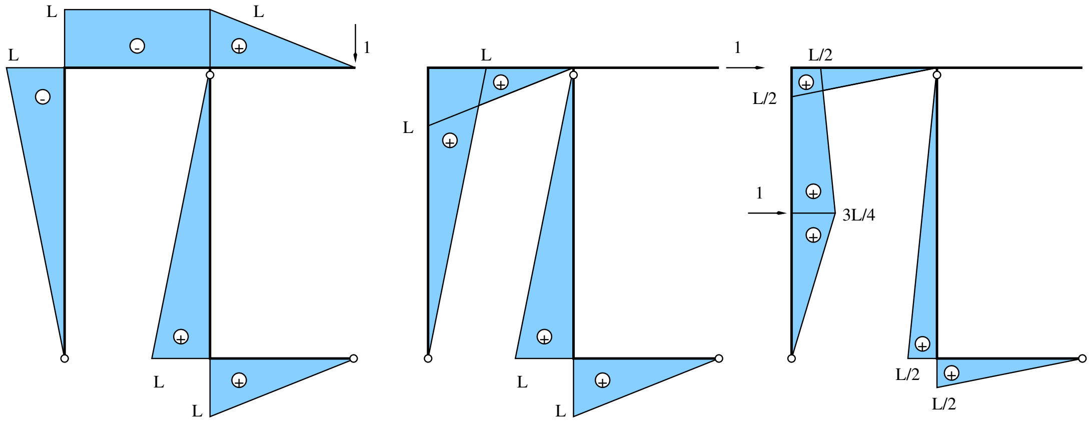
```
display(SVG(filename="mdof.svg"))
```



1.1 Flexibility matrix

We'll use the Principle of Virtual Displacements, and we start drawing the diagrams of bending moments for the three different load conditions we have loading the structure with a unit force in the location and direction of each one of the dynamic degrees of freedom.

```
display(SVG(filename="bending.svg"))
```



The bending moments on each stretch of beam can be represented using polynomials, so we give a short name to the polynomial object that is defined by the `scipy` library, and then fill a table `m` with `NDOF` rows and as many columns as the number of different intervals of definition of the bending moments (here 6).

Note that the *single* argument needed to create a polynomial object is a sequence, containing all the coefficients from the highest power to the lowest one — e.g., in `p((1,0))` the sequence `(1,0)` that is used to create the polynomial object has length 2, so the polynomial is `1*x**1 + 0*x**0`.

```
p = poly1d

m = [[p((1,0)),p((0.0,-1)),p((-0.50,0)),p((-0.5, -0.5)),p((0.50,0)),p((1.0,0))],
     [p((0,0)),p((1.0,+0)),p((+0.50,0)),p(( 0.5,  0.5)),p((0.50,0)),p((1.0,0))],
     [p((0,0)),p((0.5,+0)),p((+0.75,0)),p((-0.25,0.75)),p((0.25,0)),p((0.5,0))]]
```

Now that we have a data structure with a representation of the bending moments we

1. initialise a matrix,
2. create a sequence containing the normalised lengths of the beam stretches (only one has a length 2L)
3. compute the `i` and `j` terms of the flexibility matrix - `i` and `j` are the indices used to access the rows of `m`
   - zero out `s`
   - start a cycle over the columns - aka beam stretches - of `m`
   - for each stretch, compute $M_i(x)M_j(x)$ using the nice properties that the multiplication of two poly objects gives a new poly object
   - compute the indefinite integral, taking advantge of a library functions that does this for polynomial objects
   - compute the definite integral over the (normalized) length of the current stretch and increment `s` accordingly
   - assign `s` to the element `i,j` of the flexibility matrix.

```
F = zeros((3,3))
l = [1,1,1,1,2,1]

for i in range(3):
    for j in range(3):
        s = 0
        for n in range(6):
            integrand = m[i][n]*m[j][n]
            integral = polyint(integrand)
            s = s + integral(l[n]) - integral(0)
        F[i,j] = s

print F*12,"/ 12"
```

```
[[ 36.  -2.  -4.]
 [ -2.  24.  15.]
 [ -4.  15.  11.]] / 12
```

1.2 Eigenvalues and Eigenvectors

Now we have `F`, that is an `array` object in the speech of `scipy`, but it's easier to do what we want to do if we transform it into a `matrix` object, that has a lot of useful properties that directly mimics the properties of a matrix as we know them from algebra.

```
F = matrix(F)
```

The three most important properties are, the `.I` attribute that gives the matrix inverse, the `.T` attribute that gives the matrix transpose and last but not least the overloading of the `*` operator to have matrix product between two matrices.

So we compute `K` by inversion of `F`, we construct the mass matrix `M` directly and at his point we can import a library function and compute the eigenvalues' and eigenvectors' arrays.

```
K = F.I
print K*304/3

M = matrix("1 0 0;0 1 0;0 0 1")

from scipy.linalg import eigh

evals, evecs = eigh(K,M)
evecs = matrix(evecs)
evecs[:,0] *= -1
evecs[:,2] *= -1
```

```
[[  39.  -38.   66.]
 [ -38.  380. -532.]
 [  66. -532.  860.]]
```

Are the eignvalues correctly sorted? Are the eigenvectors orthonormal with respect to the structural matrices?

```
print evals
print
print evecs
print
print evecs.T*M*evecs
print
print evecs.T*K*evecs
```

```
[  0.30751138   0.38746035  11.9267388 ]

[[ 0.80015337  0.59627453  0.06489431]
 [-0.47377838  0.69467934 -0.54125287]
 [-0.36781604  0.40233978  0.83835199]]

[[  1.00000000e+00   5.55111512e-17  -5.55111512e-17]
 [  5.55111512e-17   1.00000000e+00  -1.11022302e-16]
 [ -5.55111512e-17  -1.11022302e-16   1.00000000e+00]]

[[  3.07511376e-01   1.15879528e-15  -5.13478149e-16]
 [  8.74300632e-16   3.87460352e-01   4.71844785e-16]
 [ -4.44089210e-16   0.00000000e+00   1.19267388e+01]]
```

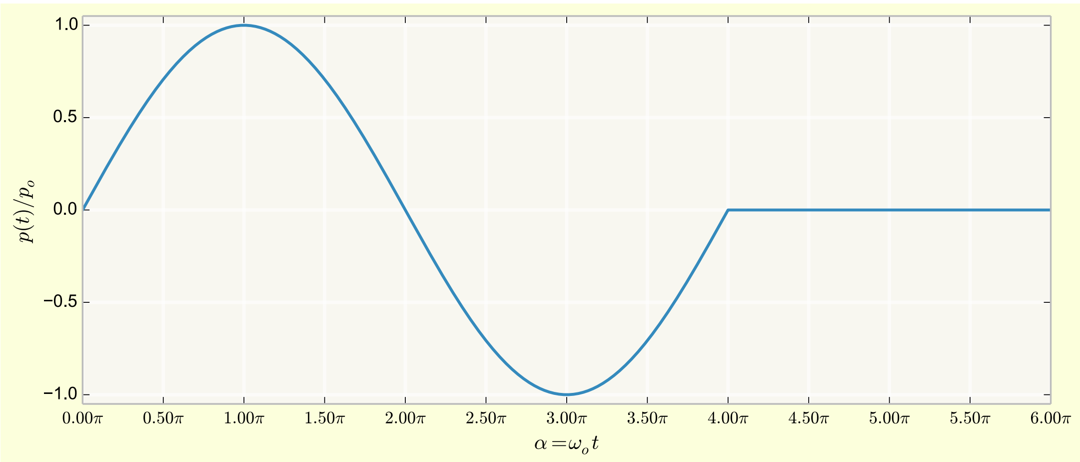
The largest absolute off-diagonal term in `evecs.T*M*evecs` is rather small: if `1` were an AU (iu.e., the mean Earth-Sun distance) then `1.39E-16*`would be about 21 micron.

1.3 Excitation

Our plan is to use an adimensional time, $\alpha = \omega_o t$, and an adimensional load, $\boldsymbol{\rho}(\alpha) = p(\alpha)/p_o \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}$.

```
def r(a):
    return where(a<4*pi, sin(0.5*a), 0.0)
```

```
a = linspace(0,6*pi, 2401)
pl.plot(a, r(a))
pl.xlabel(r'$\alpha = \omega_ot$')
pl.ylabel(r'$p(t)/p_o$')
n = 200
pl.xticks(a[::n], [r"$%5.2f\pi$"%(x/pi,) for x in a[::n]])
pl.xlim(0,6*pi)
pl.ylim(-1.05,1.05);
```



Note that the circular frq. of the excitation, $0.5\omega_o$, is quite close to the circular frequencies of the first two modes, especially the 1st one.

```
print "   modal circular frequencies:", sqrt(evals)
print "dynamic amplification factors:", 1/(1-0.25/evals)
```

```
modal circular frequencies: [ 0.55453708  0.62246313  3.45351108]
dynamic amplification factors: [ 5.34696608  2.81870624  1.02141009]
```

1.4 Modal equations of motion

With $M_i = m$, $K_i = \omega_i^2 m = \Lambda_i^2 \omega_o^2 m$ and $p_o = \delta_o \frac{EJ}{L^3}$, the modal eom is

$$m\ddot{q}_i(t) + \Lambda^2 \omega_o^2 m q_i(t) = \Gamma_i \frac{EJ}{L^3} \delta_o \sin(0.5\omega_o t)$$

dividing both members by m, as by definition it is $\omega = \frac{EJ}{mL^3}$ we have

$$\ddot{q}_i(t) + \Lambda_i^2 \omega_o^2 q_i(t) = \Gamma_i \omega_o^2 \delta_o \sin(0.5\omega_o t).$$

The coefficients Γ_i are given by

$$\boldsymbol{\Gamma} = \boldsymbol{\Psi}^T \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}.$$

```
Gamma = evecs.T*matrix('1;0;0')
print evecs
print
print Gamma
```

```
[[ 0.80015337  0.59627453  0.06489431]
 [-0.47377838  0.69467934 -0.54125287]
 [-0.36781604  0.40233978  0.83835199]]

[[ 0.80015337]
 [ 0.59627453]
 [ 0.06489431]]
```

For our undamped systems, subjected to a sine excitation not in resonance, with $\Lambda_o = 0.5$, the general integrals can be written

$$q_i(a) = A_i \sin(\Lambda_i a) + B_i \cos(\Lambda_i a) + C_i \sin(\Lambda_o a).$$

1.4.1 Particular Integral

With $\xi_i = C_i \sin(\Lambda_o a)$ and $\ddot{\xi}_i = -\omega_0^2 \Lambda_o^2 C_i \sin(\Lambda_o a)$, substituting in the eom gives

$$C_i\,\omega_o^2\left(\Lambda_i^2 - \Lambda_o^2\right)\sin(\Lambda_o a) = \Gamma_i \omega_o^2 \delta_o \sin(\Lambda_o a) \qquad \Rightarrow \qquad C_i = \frac{\Gamma_i}{\Lambda_i^2 - \Lambda_o^2}\delta_o.$$

```
L_o = 0.50
L   = sqrt(evals)
C = ravel(Gamma)/(evals-L_o**2)
print C
```

```
[  1.39129582e+01   4.33779285e+00   5.55757144e-03]
```

1.4.2 Forced response

$$q_i(a) = C_i\left(\sin(\Lambda_o a) - \frac{\Lambda_o}{\Lambda_i}\sin(\Lambda_i a)\right).$$

We still miss the coefficients $\beta_i = \frac{\Lambda_o}{\Lambda_i}$, so we compute them and immediately define two arrays of functions, the modal velocities and the modal velocities. Note that the modal displacements are normalized with respect to δ_o, while the velocities are normalized with respect to the unit velocity $\delta_o\omega_o$.

```
B = L_o/sqrt(evals)

q0_f = [lambda a,i=i: C[i] * (sin(L_o*a) - B[i]*sin(L[i]*a)) for i in range (3)]
q1_f = [lambda a,i=i: C[i] * L_o * (cos(L_o*a) - cos(L[i]*a)) for i in range (3)]
```

It's about time to display our results, first in a textual representation

```
format_q = r'$\displaystyle\frac{q_%d(a)}{\delta_o} = %+10G
\sin( 0.500 a) %+10G \sin(%9G a)$'
format_v = r'$\displaystyle\frac{\dot{q}_%d(a)}{\delta_o\omega_o} = %+10G \left(\cos( 0.500 a
for i in range(3):
    display(Latex(format_q % (i+1, C[i], -C[i]*B[i], L[i])))
    display(Latex(format_v % (i+1, C[i]*L_o, L[i])))
    print
```

$$\frac{q_1(a)}{\delta_o} = +13.913\sin(0.500a) - 12.5447\sin(0.554537a)$$

$$\frac{\dot{q}_1(a)}{\delta_o\omega_o} = +6.95648\left(\cos(0.500a) - \cos(0.554537a)\right)$$

$$\frac{q_2(a)}{\delta_o} = +4.33779\sin(0.500a) - 3.48438\sin(0.622463a)$$

$$\frac{\dot{q}_2(a)}{\delta_o\omega_o} = +2.1689\,(\cos(0.500a) - \cos(0.622463a))$$

$$\frac{q_3(a)}{\delta_o} = +0.00555757\sin(0.500a) - 0.000804626\sin(3.45351a)$$

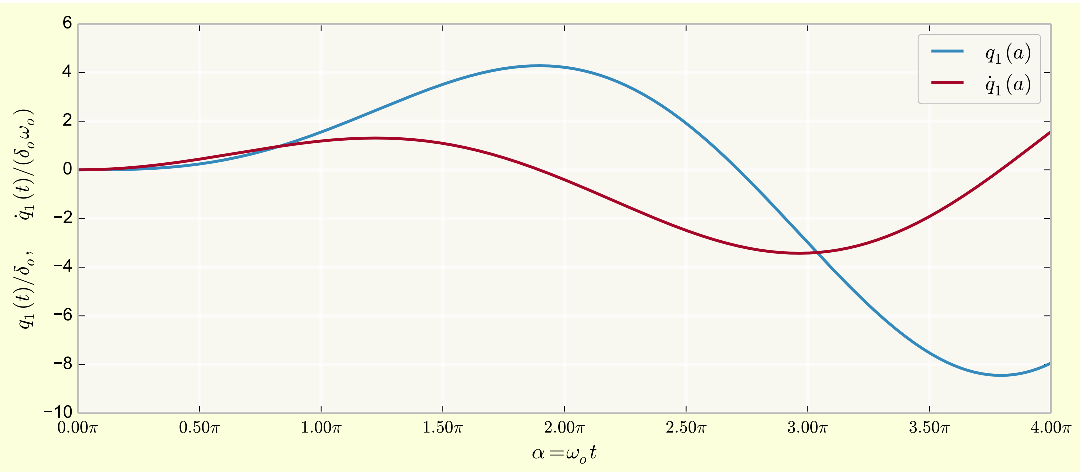
$$\frac{\dot{q}_3(a)}{\delta_o\omega_o} = +0.00277879\,(\cos(0.500a) - \cos(3.45351a))$$

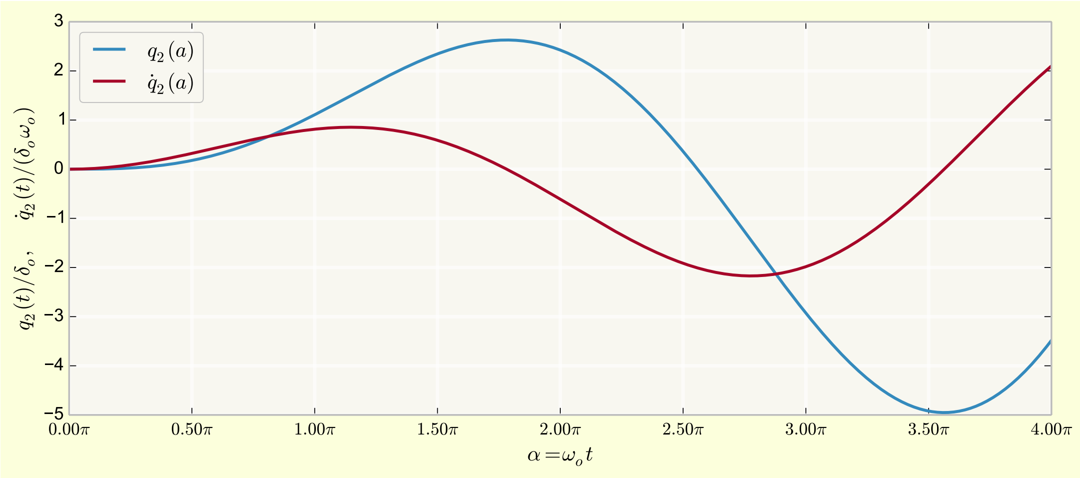
and eventually in a graphical rendering.

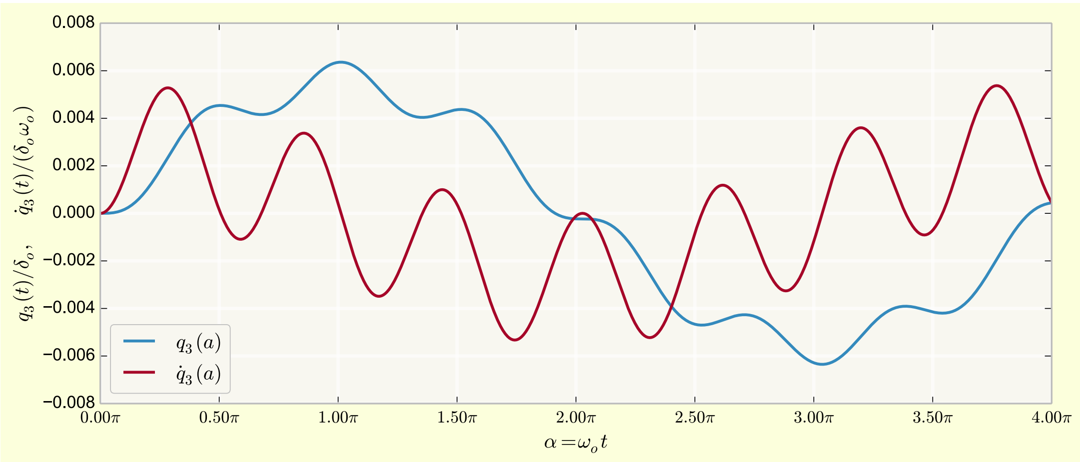
Here I have chosen to display separately, mode by mode, the modal displacement and the modal velocity, so the plotting commands are inside a loop, as well as a final `pl.show()` command that is necessary to obtain three separate plots (try to comment the last line and re-execute).

```
a = linspace(0,4*pi, 1601)
n = 200

for i in range(3):
    pl.plot(a,q0_f[i](a), label=r'$q_%d(a)$'%(i+1,))
    pl.plot(a,q1_f[i](a), label=r'$\dot{q}_%d(a)$'%(i+1,))
    pl.legend(loc=0)
    pl.xlabel(r'$\alpha = \omega_ot$')
    pl.ylabel(r'$q_%d(t)/\delta_o,\quad \dot q_%d(t)/(\delta_o\omega_o)}$'%(i+1,i+1))
    pl.xticks(a[::n], [r"$%5.2f\pi$"%(x/pi,) for x in a[::n]])
    pl.xlim(0,4*pi)
    pl.show();
```

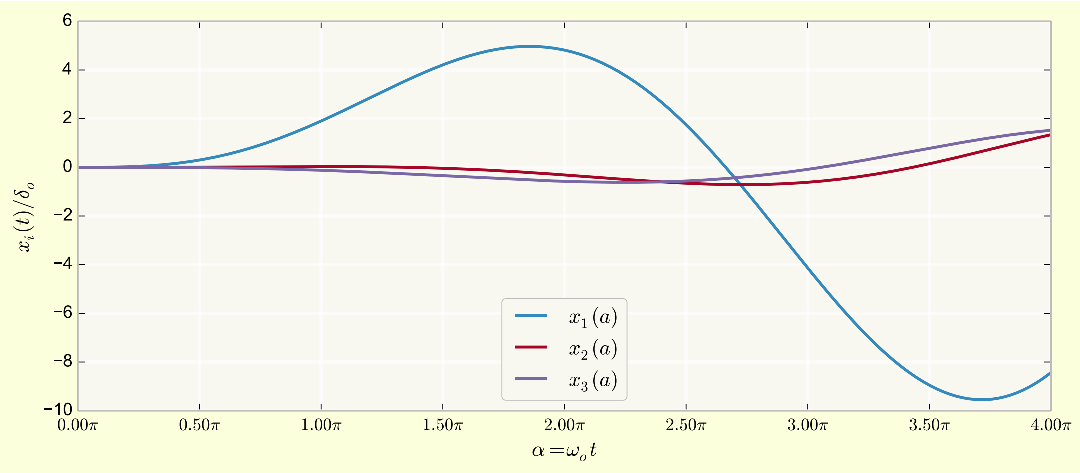






For plotting the DOF displacements, I put inside the loop only the `pl.plot` command, so that the three components can be appreciated side by side.

```
for i in range(3):
    pl.plot(a,
            evecs[i,0]*q0_f[0](a)+
            evecs[i,1]*q0_f[1](a)+
            evecs[i,2]*q0_f[2](a),
            label=r'$x_%d(a)$'%(i+1,))
pl.xlabel(r'$\alpha = \omega_ot$')
pl.ylabel(r'$x_i(t)/\delta_o$')
pl.xticks(a[::n], [r"$%5.2f\pi$"%(x/pi,) for x in a[::n]])
pl.xlim(0,4*pi)
pl.legend(loc=8);
```



1.4.3 Free Response

```
a_a = 4*pi # from a in forced range to a in free response range
for i in range(3):
    print "%d\t%+12G\t%+12G" % (i+1, q0_f[i](a_a), q1_f[i](a_a))
```

```
1            -7.9399            +1.57072
2           -3.48261            +2.09977
3       +0.000443775        +0.000460847
```

```
q  = [] ; qf = [] ; pp = []

for i in range(3):
    l = L[i]
    M = matrix((( sin(l*a_a),    + cos(l*a_a)),
                ( l*cos(l*a_a),  - l*sin(l*a_a))))
    xvT = matrix(( (q0_f[i](a_a),), (q1_f[i](a_a),)))
    Af, Bf = ravel(M.I*xvT)
    def qf0(a, i=i, l=l, Af=Af, Bf=Bf):
        return Af*sin(l*a)+Bf*cos(l*a)
    qf.append(qf0)
    qf1 = lambda a: l*Af*cos(l*a) - l*Bf*sin(l*a)
    print "%d\t%+12G\t%+12G" % (i+1, qf0(4*pi), qf1(4*pi))
    format_qf = r'\frac{q_%d(a)}{\delta_o} &= %+10G \sin(%10G a) %+10G \cos(%10G a)'
    pp.append(format_qf%(i+1, Af, L[i], Bf, L[i]))
    def qi(a, i=i):
        return where(a<a_a, q0_f[i](a), qf[i](a))
    q.append(qi)
print
display(Latex(r'\begin{align}'+r'\\'.join(pp)+r'\end{align}'))
```

```
1            -7.9399            +1.57072
2           -3.48261            +2.09977
3       +0.000443775        +0.000460847
```

$$\frac{q_1(a)}{\delta_o} = -2.83248 \sin(0.554537a) - 7.9399 \cos(0.554537a) \tag{1}$$

$$\frac{q_2(a)}{\delta_o} = -3.37332 \sin(0.622463a) - 3.48261 \cos(0.622463a) \tag{2}$$

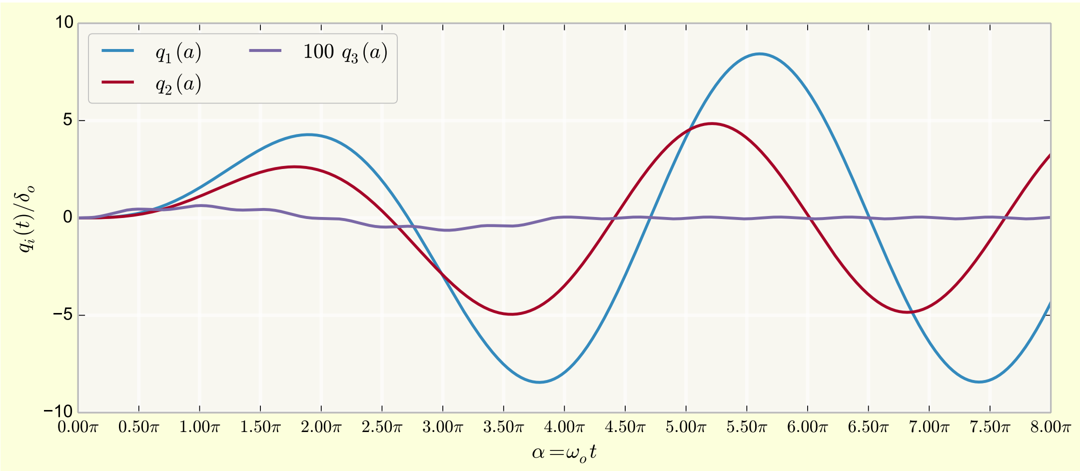
$$\frac{q_3(a)}{\delta_o} = -0.000133443 \sin(3.45351a) + 0.000443775 \cos(3.45351a) \tag{3}$$

1.4.4 Closing down

To close this soo loong exercise, here are the plots for the modal displacements (note the scaling factor applied to q_3, that would be otherwise represented as a horizontal line)

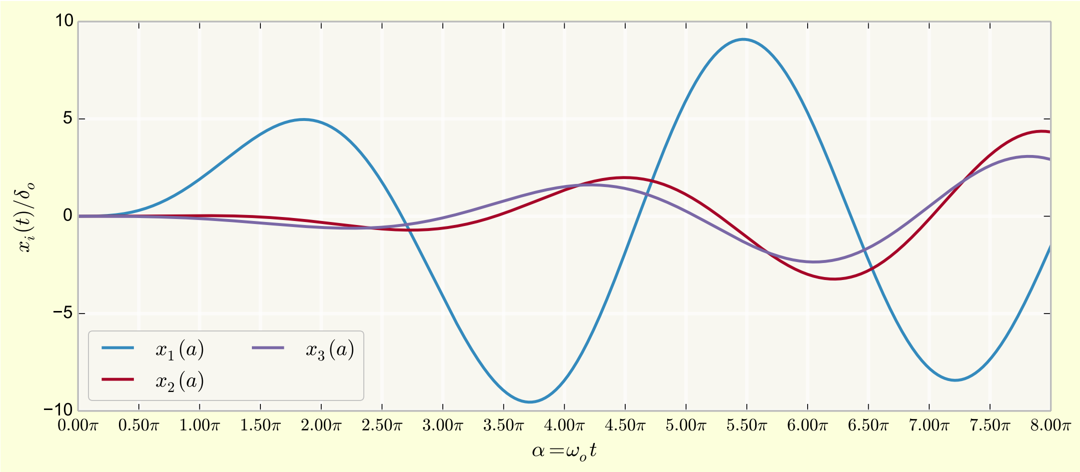
```
a = linspace(0,8*pi,3201)

for i in range(3):
    pl.plot(a,(1,1,100)[i]*q[i](a), label=r'$%sq_%d(a)$'%(('','','100\\,')[i],i+1))
pl.xlabel(r'$\alpha = \omega_ot$')
pl.ylabel(r'$q_i(t)/\delta_o$')
pl.xticks(a[::n], [r"$%5.2f\pi$"%(x/pi,) for x in a[::n]])
pl.xlim(0,8*pi)
pl.legend(loc=0,ncol=2);
```



and for the DOF displacements.

```
for i in range(3):
    pl.plot(a,evecs[i,0]*q[0](a)+evecs[i,1]*q[1](a)+evecs[i,2]*q[2](a),
            label=r'$x_%d(a)$'%(i+1,))
pl.xlabel(r'$\alpha = \omega_ot$')
pl.ylabel(r'$x_i(t)/\delta_o$')
pl.xticks(a[::n], [r"$%5.2f\pi$"%(x/pi,) for x in a[::n]])
pl.xlim(0,8*pi)
pl.legend(loc=3,ncol=2);
```



10
5
0
−5
−10
$x_i(t)/\delta_o$
$x_1(a)$
$x_2(a)$
$x_3(a)$
0.00π
0.50π
1.00π
1.50π
2.00π
2.50π
3.00π
3.50π
4.00π
4.50π
5.00π
5.50π
6.00π
6.50π
7.00π
7.50π
8.00π
$\alpha = \omega_o t$