01_SDOF_Response

May 23, 2014

1 Input

The data for the system is

```
m =    1200.0      ; mass = m
k = 800000.0
T = 0.250000
```

1.1 Determination of the damping ratio

We don't know the damping ratio, but it is

$$\omega_D^2 = \frac{2\pi}{T} = \omega_n^2 (1 - \zeta^2)$$
$$\zeta = \sqrt{1 - \frac{\omega_D^2}{\omega_n^2}}$$

When ζ is known, it is possible to compute `damp`, the damping constant c.

```
wd              =     2*pi/T
wd2             =     wd*wd
wn2             =     k/m
wn              =     sqrt(wn2)
z               =     sqrt(1-wd2/wn2)
is_it_T         =     2*pi/(wn*sqrt(1.0-z*z))

print "z = %6.3f%%"%(z*100,)
print "T =", is_it_T

damp            =     2*z*sqrt(k*m)
```

```
z = 22.917%
T = 0.25
```

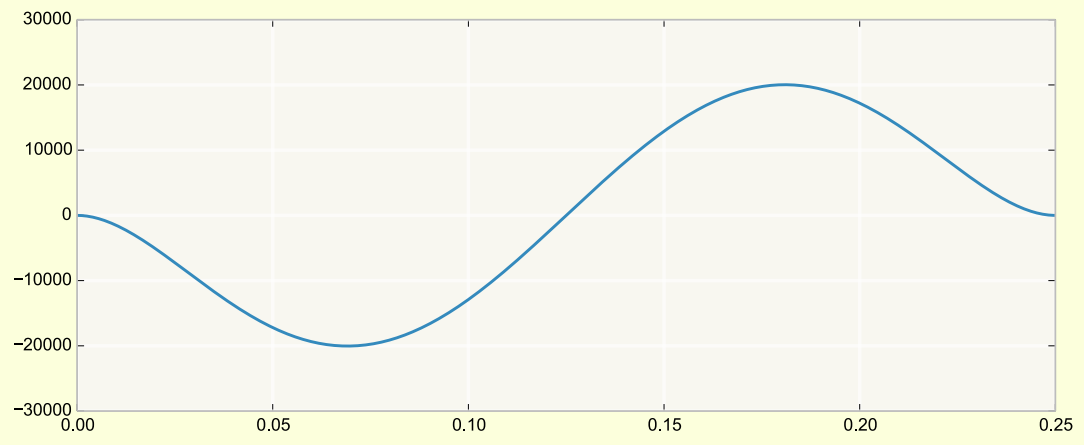
1.2 External load

The external load is assigned as a function of $\tau = t/T$, substituting $T = 0.25$s it is $\tau = 4t$, substituting in the polynomial and multiplying by p_o we have (nb, the different coefficients have different dimensionalities)

```
def p(t): return (((2293760000*t - 1433600000)*t + 286720000)*t - 17920000)*t*t
```

To get more confidence in the above definition, let's plot our `p(t)` over is interval of definition

```
t = linspace(0,.25,1001)
pl.plot(t,p(t));
```



OK, it looks fine...

2 Forced Response

First, the particular integral, then the initial conditions and finally the plots

2.1 Particular Integral

Now, its time to find a particular integral, first we name all the coefficients in the polynomial load, including also the coefficients that are equal to zero

```
ap =    2293760000.
bp = -  1433600000.
cp =     286720000.
dp = -    17920000.
ep =              0.
fp =              0.
```

Now, we start with a generic 5th degree polynomial representation of our $\xi(t)$

$$\begin{aligned}\xi(t) &= at^5 + bt^4 + ct^3 + dt^2 + et + f,\\ \dot{\xi}(t) &= 5at^4 + 4bt^3 + 3ct^2 + 2dt + e,\\ \ddot{\xi}(t) &= 20t^3 + 12t^2 + 6ct + 2d.\end{aligned}$$

Using M, C and K for the system parameters, substituting in the equation of motion and equating the polynomial coefficients we have

$$\begin{aligned}(Ka)t^5 = a_p t^5 \quad &\rightarrow \quad a = a_p/K,\\ (Kb+5Ca)t^4 = b_p t^4 \quad &\rightarrow \quad b = (b_p - 5Ca)/K,\\ (Kc+4Cb+20Ma)t^3 = c_p t^3 \quad &\rightarrow \quad c = (c_p - 4Cb - 20Ma)/K,\\ (Kd+3Cc+12Mb)t^2 = d_p t^2 \quad &\rightarrow \quad d = (d_p - 3Cc - 12Mb)/K,\\ (Ke+2Cd+6Mc)t^1 = e_p t^1 \quad &\rightarrow \quad e = (e_p - 2Cd - 6Mc)/K,\\ (Kf+Ce+2Md)t^0 = f_p t^0 \quad &\rightarrow \quad f = (f_p - Ce - 2Md)/K.\end{aligned}$$

Here it is a straightforward code translation of the above equations...

```
a = (ap)/k
b = (bp - 5*damp*a)/k
c = (cp - 4*damp*b - 20*mass*a)/k
```

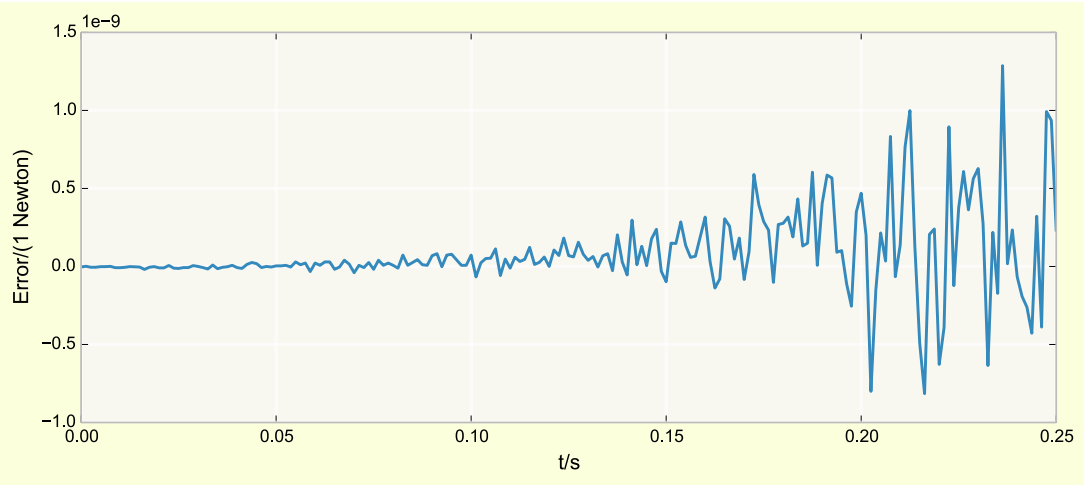
```
d = (dp - 3*damp*c - 12*mass*b)/k
e = (ep - 2*damp*d -  6*mass*c)/k
f = (fp - 1*damp*e -  2*mass*d)/k
```

2.1.1 Verification of the particular integral

Here there is a bit of verification, first we print the particular integral, then we plot the difference $m\ddot{\xi} + c\dot{\xi} + k\xi - p(t)$; for this purpose we have to define the corresponding functions (here `r0`, `r1` and `r2`).

```
lt = ('$$\\xi(t)='+'%+10.4f\\,t^{%d}'*6+'$$') % (a,5,b,4,c,3,d,2,e,1,f,0)
display(Latex(lt))

def r0(t): return a*t**5 + b*t**4 + c*t**3 + d*t**2 + e*t + f
def r1(t): return 5*a*t**4 + 4*b*t**3 + 3*c*t**2 + 2*d*t + e
def r2(t): return 20*a*t**3 + 12*b*t**2 + 6*c*t + 2*d

t = linspace(0.0, 0.25, 201)
pl.plot(t, k*r0(t) + damp*r1(t) + mass*r2(t) - p(t))
pl.xlabel('t/s')
pl.ylabel('Error/(1 Newton)');
```

$$\xi(t) = +2867.2000\,t^5 - 2046.4823\,t^4 + 417.6947\,t^3 - 7.8072\,t^2 - 3.4821\,t^1 + 0.0852\,t^0$$



As you can see, there is just a bit of numerical noise...

2.2 Initial conditions

```
print r0(0)
print r1(0)
```

```
0.0852328238184
-3.48207812204
```

The following block code does two things...

1. it finds the integration constants that respect the initial rest conditions;
2. define the displacement and velocity response functions.

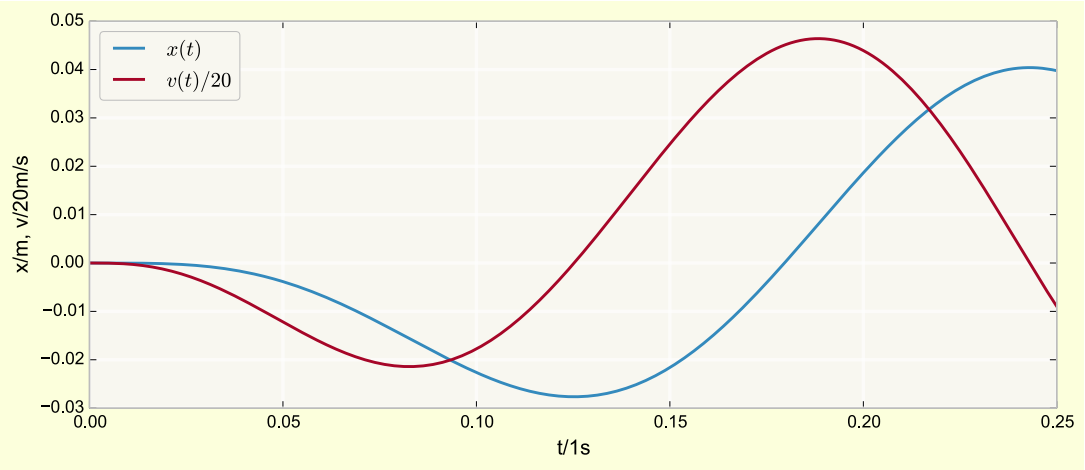
```
B = - r0(0)
A = (z*wn*B-r1(0))/wd
print A, B
def x(t, A=A, B=B):
    return exp(-z*wn*t)*(A*sin(wd*t)+B*cos(wd*t))+r0(t)
def v(t, A=A, B=B):
    vel = cos(wd*t)*(wd*A-z*wn*B) - sin(wd*t)*(z*wn*A+wd*B)
    return vel*exp(-z*wn*t) + r1(t)
    # - z*wn*exp(-z*wn*t)*(A*sin(wd*t)+B*cos(wd*t))
    # + wd*exp(-z*wn*t)*(-B*sin(wd*t)+A*cos(wd*t)) + r1(t)
```

```
0.118480812845 -0.0852328238184
```

2.3 Plotting Displacements and velocities

```
pl.plot(t, x(t),    label = r'$x(t)$')
pl.plot(t, v(t)/20, label = r'$v(t)/20$')
pl.xlabel(r't/1s')
pl.ylabel('x/m, v/20m/s')
pl.legend(loc=0)
print v(0)
```

```
0.0
```



The forcing function was *similar* to a sine, with the same period as the damped system, and you see that the displacement response (blue curve) is in quadrature with the excitation, as you could have expected... hadn't you?

3 Free response

```
print x(.25), v(.25)
```

```
0.039757530281 -0.17981859338
```

Denoting with x_f and v_f the free response, for $t > T$, we must impose the continuity of dsplacements and velocities for $t = T$:

$$x_f(T) = \exp(-\zeta\omega_n T)(A_f \sin(\omega_D T) + B_f \cos(\omega_D T)) = x(T),$$
$$v_f(T) = \exp(-\zeta\omega_n T)[-\zeta\omega_n(A_f \sin(\omega_D T) + B_f \cos(\omega_D T)) + \omega_D(A_f \cos(\omega_D T) - B_f \sin(\omega_D T))] = v(T).$$

Collecting the constants of integration we can put the above in matrix format:

$$\exp(-\zeta\omega_n t)\begin{bmatrix}\sin\omega_D T & \cos\omega_D T\\ -\zeta\omega_n\sin\omega_D T+\omega_D\cos\omega_D T & -\zeta\omega_n\cos\omega_D T-\omega_D\sin\omega_D T\end{bmatrix}\begin{Bmatrix}A_f\\ B_f\end{Bmatrix}=\begin{Bmatrix}x(T)\\ v(T)\end{Bmatrix}.$$

The following code defines the matrix of coefficients and the column vector of the initial state, then solves the linear system and extracts the constant of integration from the vector in which they were computed.

```
sT = sin(wd*T); cT = cos(wd*T); eT = exp(-z*wn*T)
M = matrix(( (          sT,              cT       ),
             ( -z*wn*sT+wd*cT, -wd*sT-z*wn*cT)))*eT
xvT = matrix(((x(T),),
              (v(T),)))
Af, Bf =  ravel(M.I*xvT)
print Af, Bf
```

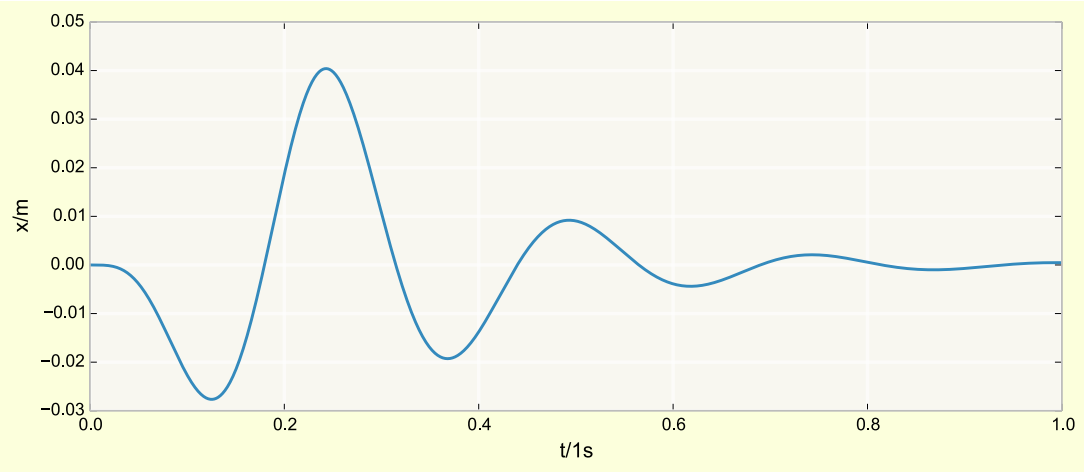
```
0.00968162561069 0.174525771352
```

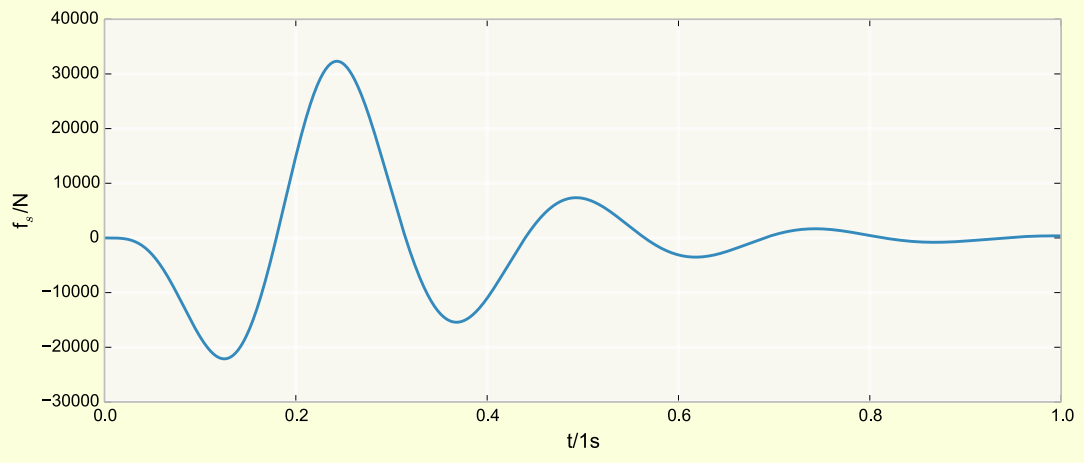
Using the constants of integration that we have just derived, we define the displacement function during free response and an auxiliary function that returns the correct response for the different time intervals, so that we can plot the response, both in terms of displacement and in terms of elastic force.

```
def xf(t):
    return exp(-z*wn*t)*(Af*sin(wd*t)+Bf*cos(wd*t))
def x12(t):
    return where(t<T, x(t), xf(t))
```

3.1 Plotting

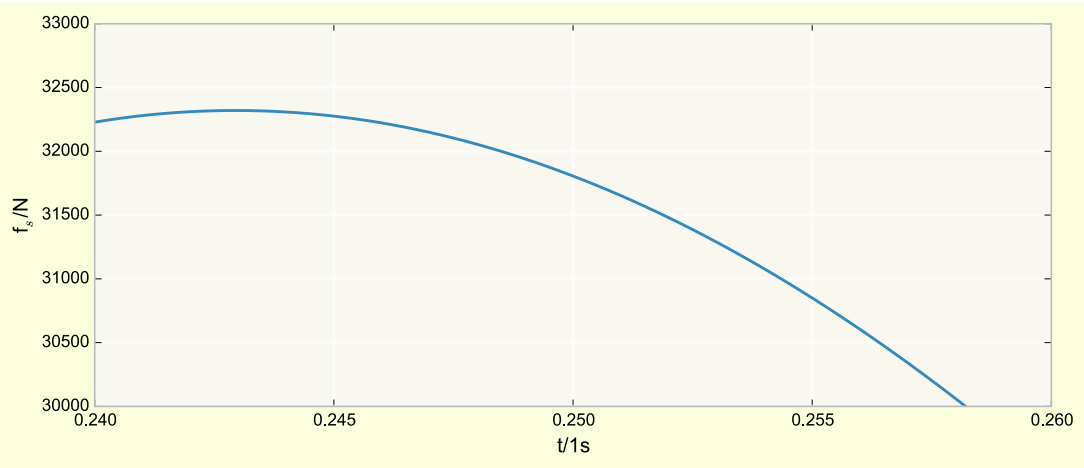
```
t1 = linspace(0.0, 1.0, 4001)
pl.plot(t1, x12(t1));
pl.xlabel(r't/1s')
pl.ylabel('x/m')
pl.show()
pl.plot(t1, k*x12(t1))
pl.xlabel(r't/1s')
pl.ylabel('f${}_s$/N');
```





We know, from the second problem, that $f_y = 32500$N. Is the spring force always below the yielding force?

```
pl.plot(t1, k*x12(t1)); pl.xlim(0.24,0.26); pl.ylim((30000,33000));
pl.xlabel(r't/1s')
pl.ylabel('f${}_s$/N');
```



Hmm, it seems OK.

4 Verification

We compute numerically the response using the linear acceleration algorithm

```
def P(t):
    return where(t<0.25, p(t), 0.0)

h  = 0.0001
h2 = h*h

stop = 1.0+h/2

k_ =        k + 3*damp/h + 6*mass/h2
a0_coeff = 3*mass + damp*h/2
v0_coeff = 6*mass/h + 3*damp
```

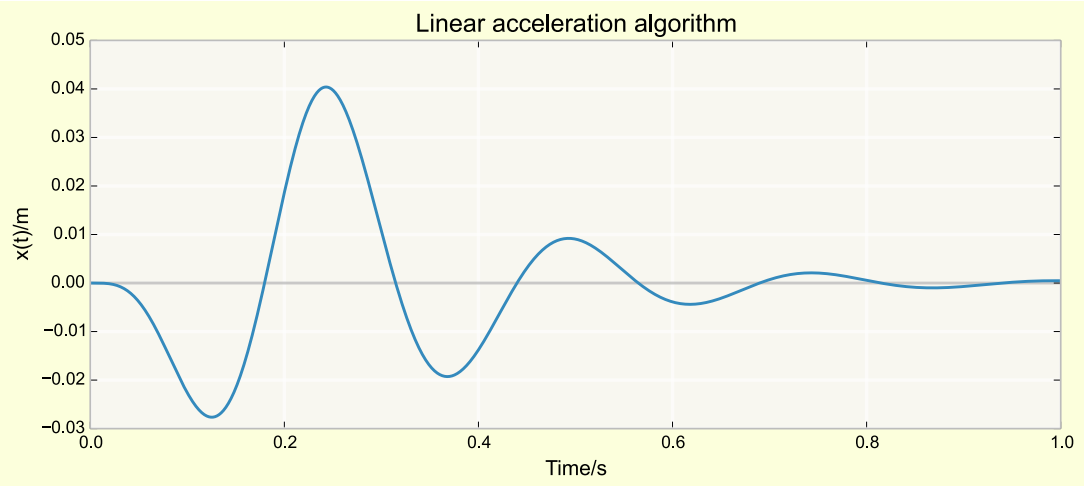
```
dx2dv = 3.0/h ; v2dv = 3.0 ; a2dv = h/2.0

t0 =0 ; x0 = 0 ; v0 = 0 ; p0 = 0 ; a0 = 0

t= [] ; X = [] ; V = [] ; A = []

while t0<stop:
    t.append(t0) ; X.append(x0) ; V.append(v0) ; A.append(a0)
    t1 = t0+h
    p1 = P(t1)
    dp = p1 - p0 + a0_coeff*a0 + v0_coeff*v0
    dx = dp/k_
    dv = dx*dx2dv - v0*v2dv - a0*a2dv
    t0 = t1
    p0 = p1
    x0 = x0+dx
    v0 = v0+dv
    a0 = (p0 - k*x0 - damp*v0)/mass
```

Put the numerical response in X and the analytical one in E, for exact...

```
t = array(t)
X = array(X)
E = x12(t)
```

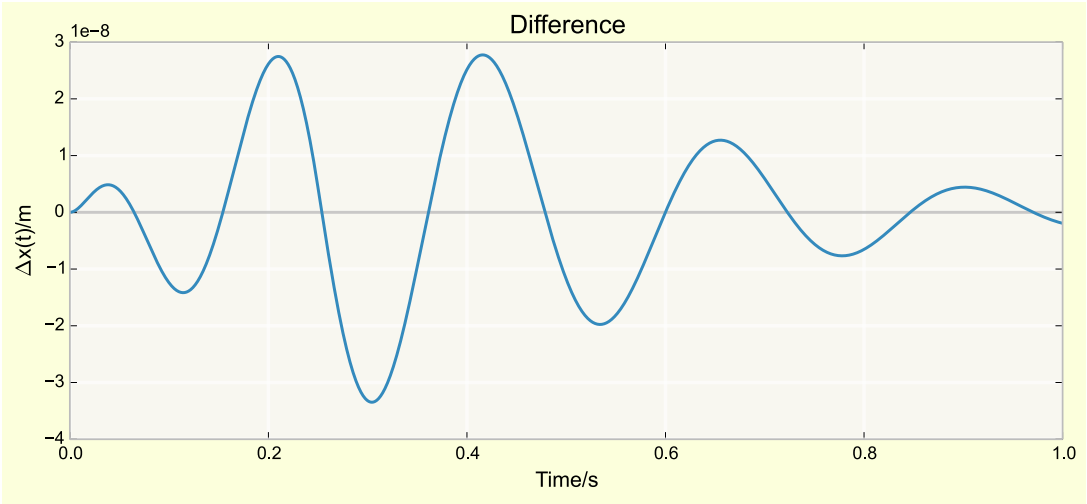
Plot first the numerical result

```
pl.plot(t,X)
pl.xlabel('Time/s')
pl.ylabel('x(t)/m')
pl.hlines(0,0,1,alpha=0.2)
pl.xlim(0,1.0)
pl.title('Linear acceleration algorithm')
pl.show()
```



and the difference between exact and numerical.

```
pl.plot(t, E-X)
pl.xlabel('Time/s')
pl.ylabel('$\\Delta$x(t)/m')
pl.hlines(0,0,1,alpha=0.2)
pl.xlim(0,1.0)
pl.title('Difference');
```


Difference
1e−8
3
2
1
0
−1
−2
−3
−4
Δx(t)/m
0.0
0.2
0.4
0.6
0.8
1.0
Time/s