02_Numerical_Integration

May 23, 2014

Usual imports, plus the `%matplotlib inline` magic to have the graphics inlined and a small helper function.

```
%matplotlib inline
from scipy import *
import matplotlib.pylab as pl
from IPython.display import display, Latex, HTML
def pnv(n, v, u=""):
    "print name and value on a single line"

    print "%45s = %g %s" % (n, v, u)
```

1 System definition

The system parameters as in the problem statement.

```
m =    1200.     ; mass = m
k = 800000.
T = 0.2500
```

The other parameters are readily derived, with obvious meaning of the different names.

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

pnv('Period of the damped system', T, 's')
pnv("System mass", m, 'kg')
pnv("System damping", damp, 'N s/m')
pnv("System stiffness", k, 'N/m')
pnv("Damping ratio", z*100, '%')
pnv("Natural circular frequency", wn, 'rad/s')
pnv("Damped circular frequency", wd, 'rad/s')
```

```
Period of the damped system = 0.25 s
                                  System mass = 1200 kg
                               System damping = 14201 N s/m
                             System stiffness = 800000 N/m
                                Damping ratio = 22.9168 %
                   Natural circular frequency = 25.8199 rad/s
                    Damped circular frequency = 25.1327 rad/s
```

The characteristics of the loading are

```
T0 = 0.04
wf = pi/T0
po = 80000.0
```

2 Estimate of the max response

For an *undamped* system (our system is heavily damped) when the maximum response occurs in the free response, the peak response is given by

$$x_0 = \Delta_s \frac{2\beta}{\beta^2 - 1} \cos\left(\frac{\pi}{2\beta}\right)$$

```
beta = T/(2*T0)
pnv("Max response for a fictitious undamped system",
    (80000/k) * (2*beta)/(beta**2-1) * cos(pi/(2*beta)) *1000, 'mm')
```

```
Max response for a fictitious undamped system = 62.4818 mm
```

3 Analytical response

To compare our numerical solution with another solution of the same problem, we derive analytically the integral of the EOM.

3.1 Particular Integral

We compute the particular integral, $\xi = S \sin \omega_f t + C \cos \omega_f t$, determining the costants S and C

```
M = matrix(((wn**2-wf**2,  -2*z*wn*wf),
            ( +2*z*wn*wf, wn**2-wf**2)))
sc = matrix((wn*wn*po/k, 0.0)).T
S, C = ravel(M.I*sc)
```

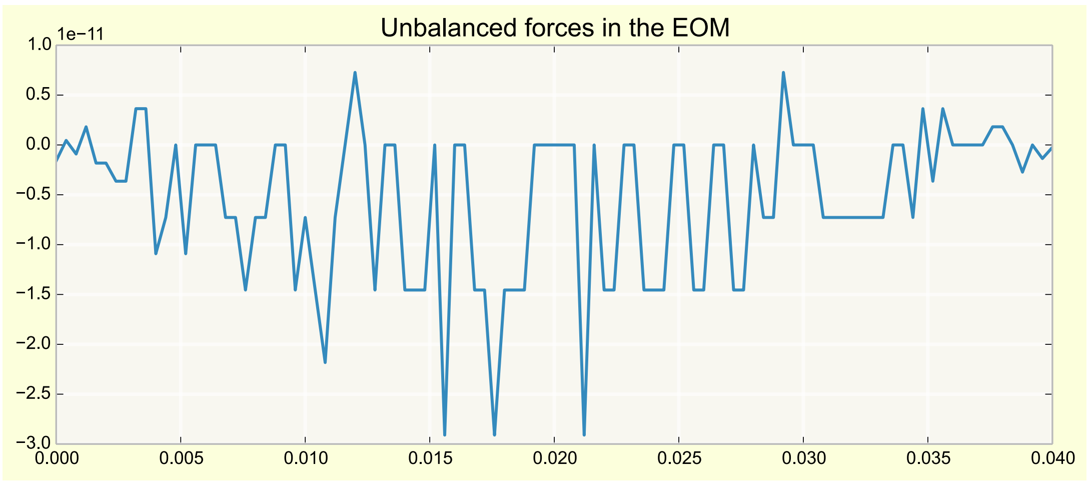
We define a set of functions that return the values of the particular integral and the values of its time derivatives

```
def r0(t, S=S, C=C): return S*sin(wf*t)+C*cos(wf*t)
def r1(t, S=S, C=C): return wf*(S*cos(wf*t)-C*sin(wf*t))
def r2(t): return -wf**2*r0(t)
```

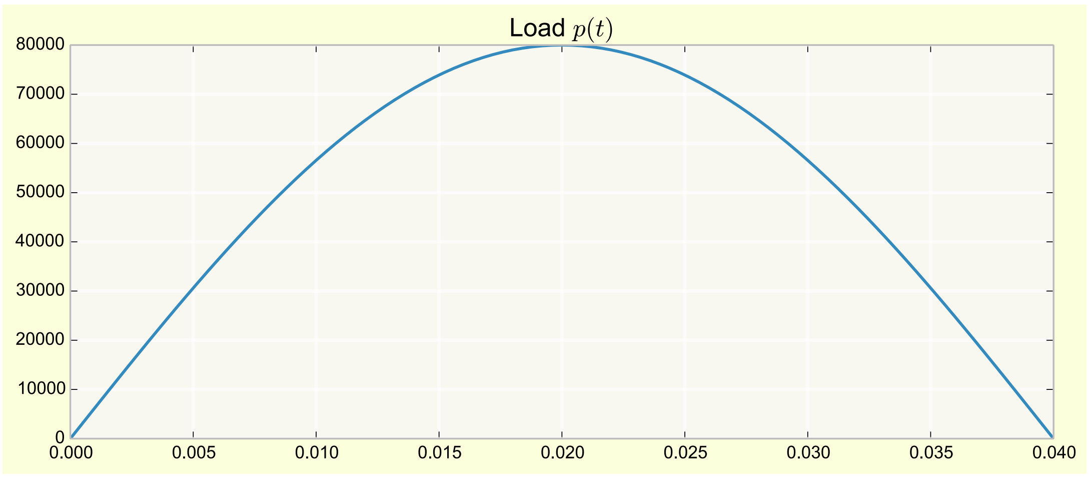
3.1.1 Verification

Just to be sure, we plot the difference between the load and the value that we obtain substituting the particular integral in the left member of the eq. of dynamic equilibrium.

```
def error(t): return mass*r2(t)+damp*r1(t)+k*r0(t)-p(t)
t = linspace(0,T0,101)
def p(t): return po*sin(wf*t)
pl.plot(t,error(t)); pl.xlim(0,T0);
pl.title('Unbalanced forces in the EOM')
pl.show()
pl.plot(t,p(t)); pl.xlim(0,T0)
pl.title('Load $p(t)$')
```



```
<matplotlib.text.Text at 0x7f7086065590>
```



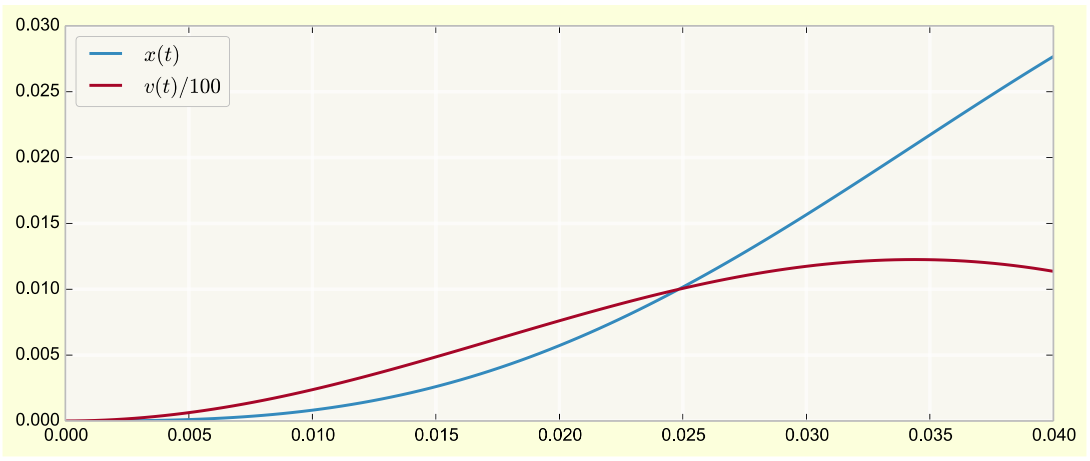
3.2 Forced response

Having $\xi(t)$ and $\dot{\xi}(t)$ we can determine the constants of integration in the homogeneous integral imposing the respect of the initial rest conditions.

```
B = - r0(0)
A = (z*wn*B-r1(0))/wd
def xforced(t,A=A,B=B):
    return exp(-z*wn*t)*(A*sin(wd*t)+B*cos(wd*t))+r0(t)
def vforced(t,A=A,B=B):
    s = sin(wd*t) ; c = cos(wd*t) ; e =exp(-z*wn*t)
    return e*(-z*wn*(A*s+B*c)+wd*(-B*s+A*c))+r1(t)
```

Let's plot the forced response in terms of displacements and velocities

```
pl.plot(t, xforced(t),    label = r'$x(t)$')
pl.plot(t, vforced(t)/100, label = r'$v(t)/100$')
pl.legend(loc=0)
pl.xlim(0,T0);
```



Let's see the values of the response at the beginning and at the end of the external loading

```
print xforced(0), xforced(T0)
print vforced(0), vforced(T0)
```

```
0.0 0.0276771363852
1.11022302463e-16 1.13622468627
```

3.3 Free response

Use a function to compute the constants of integration of the h. response for $t > t_0$

```
def coeff(T,x,v):
    s = sin(wd*T) ; c = cos(wd*T) ; e =exp(-z*wn*T)
    M = matrix(((       s,                c          ),
               ( -z*wn*s + wd*c, -z*wn*c - wd*s)))*e
    xvT = matrix(((x(T),),
                 (v(T),)))
    AB = M.I*xvT
    return AB[0,0], AB[1,0]
```

Use the function above to determine the constants of integration and define the free response and a (vector) function that returns either the forced or the free response according to the value of `t`

```
Af, Bf = coeff(T0,xforced,vforced)
def xfree(t, A=Af, B=Bf):
    return exp(-z*wn*t)*(A*sin(wd*t)+B*cos(wd*t))
def x12(t):
    return where(t<T0, xforced(t), xfree(t))
```

3.4 A graphic of the analytical response?

No, thanks. Please see the verification section.

4 Numerical Integration

4.1 Computation of the Response using Constant Acceleration

4.1.1 Initialisation

1. time step
2. upper limit to duration

3. coefficients for constant acceleration
4. load definition
5. initialise the state of the system
6. initialise the containers in which we'll store the response

```
# 1
h   = 0.001
h2 = h*h

# 2
stop = 0.500+h/2

# 3
k_ = k+2*damp/h+4*mass/h2
a0_coeff = 2*mass
v0_coeff = 4*mass/h+damp+damp

# 4
def p(t): return where(t<T0, 80000*sin(wf*t), 0)

# 5
t0 =0 ; x0 = 0 ; v0 = 0 ; p0 = 0 ; a0 = 0

# 6
t= [] ; X = [] ; V = [] ; A = []
```

4.1.2 Computation

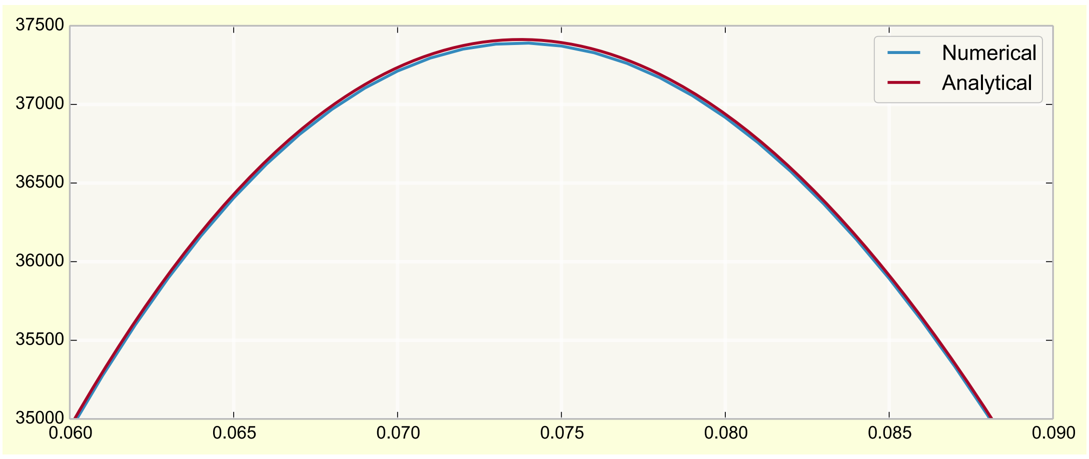
The actual loop that iterates the system state according to the constant acceleration algorithm

```
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 = 2*(dx/h-v0)
    t0 = t1
    p0 = p1
    x0 = x0+dx
    v0 = v0+dv
    a0 = (p0 - k*x0 - damp*v0)/mass
```

4.1.3 Plots of CA results

Lets plot our results near the maximum, in terms of spring force, against the exact response we previously found.

```
pl.plot(t,[k*d for d in X], label='Numerical')
time = linspace(0,0.5, 5001)
pl.plot(time, k*x12(time),label='Analytical')
pl.xlim(0,0.5)
pl.legend(loc=0)
if 1:
    pl.ylim(35000,37500)
    pl.xlim(0.06,0.09)
```



4.2 Computation of the response using Linear Acceleration

Repeat using the linear acceleration algorithm...

4.2.1 Initialisation

```
h   = 0.001
h2  = h*h

stop = 0.500+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

def p(t): return where(t<0.04, 80000*sin(pi*t/0.04), 0)

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

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

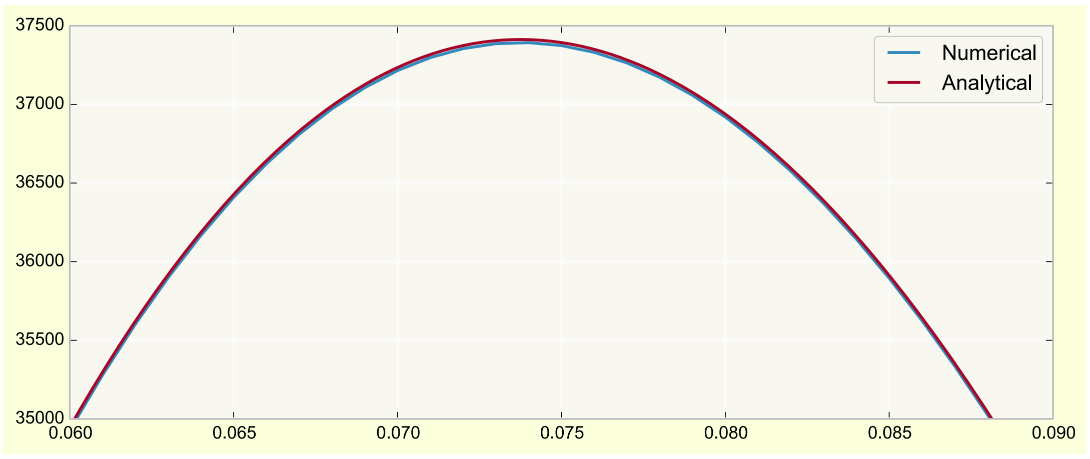
4.2.2 Computation

The loop is exactly the same as before, so I omit *printing* the code (it's still visible in the browser)

4.2.3 Plots of LA results and verifications

A comparison between the exact force response and numerical force response, in a neighborhood of the peak value of the response.

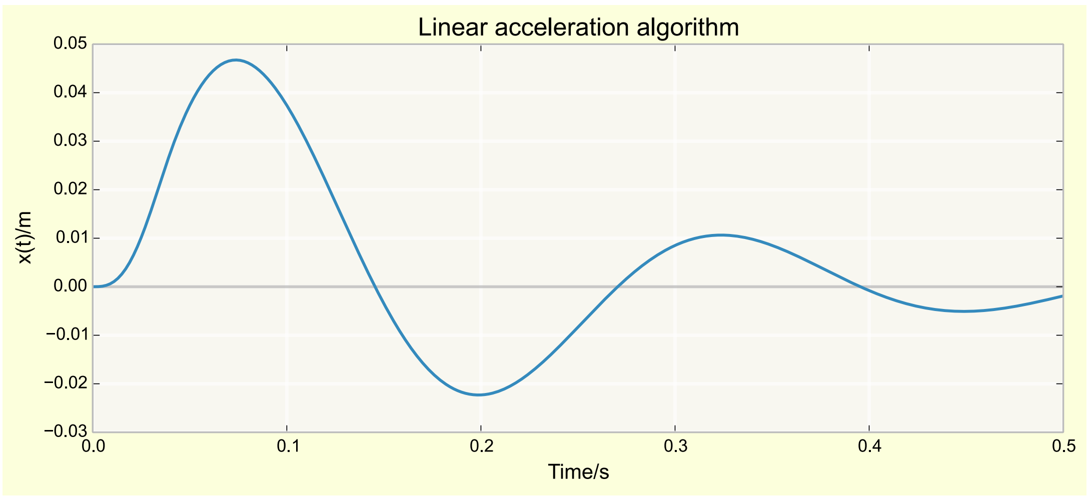
```
pl.plot(t,[k*d for d in X], label='Numerical')
time = linspace(0,0.5, 5001)
pl.plot(time, k*x12(time),label='Analytical')
pl.xlim(0,0.5)
pl.legend(loc=0)
if 1:
    pl.ylim(35000,37500)
    pl.xlim(0.06,0.09)
```

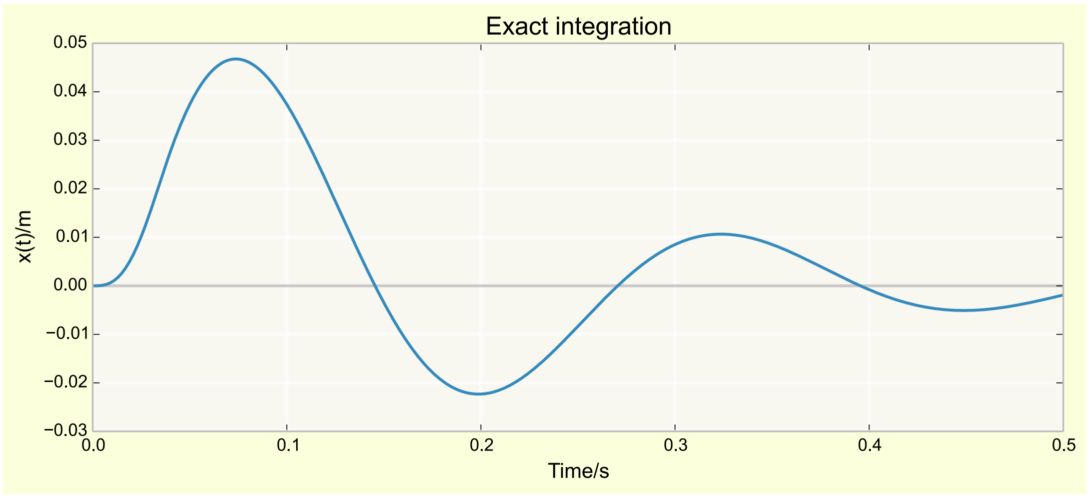


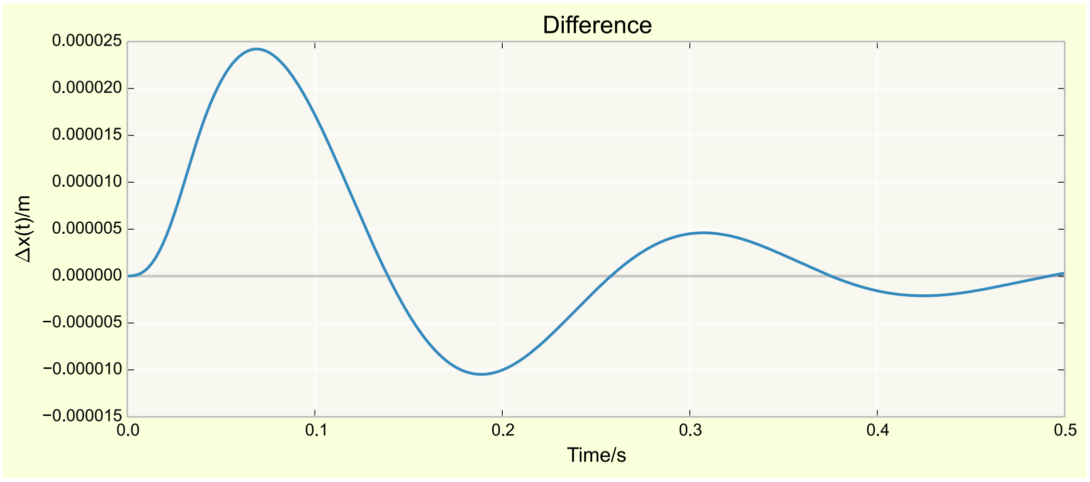
Finally, a plot for

1. the numerical response,
2. the exact response,
3. the difference, exact - numerical.

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

pl.plot(t,X)
pl.xlabel('Time/s')
pl.ylabel('x(t)/m')
pl.hlines(0,0,0.5,alpha=0.2)
pl.xlim(0,0.5)
pl.title('Linear acceleration algorithm')
pl.show()

pl.plot(t, x12(t))
pl.xlabel('Time/s')
pl.ylabel('x(t)/m')
pl.hlines(0,0,0.5,alpha=0.2)
pl.xlim(0,0.5)
pl.title('Exact integration')
pl.show()

pl.plot(t, E-X)
pl.xlabel('Time/s')
pl.ylabel('$\\Delta$x(t)/m')
pl.hlines(0,0,0.5,alpha=0.2)
pl.xlim(0,0.5)
pl.title('Difference');
```


Linear acceleration algorithm
x(t)/m
Time/s
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
0.0
0.1
0.2
0.3
0.4
0.5


Exact integration
x(t)/m
Time/s
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
0.0
0.1
0.2
0.3
0.4
0.5


Difference
Δx(t)/m
Time/s
0.000025
0.000020
0.000015
0.000010
0.000005
0.000000
−0.000005
−0.000010
−0.000015
0.0
0.1
0.2
0.3
0.4
0.5