Exact Integration for an EP SDOF System

We want to compute the response, using the constant acceleration algorithm plus MNR, of an Elasto Plastic (EP) system... but how we can confirm or reject our results?

It turns out that computing the exact response of an EP system with a single degree of freedom is relatively simple.

Here we discuss a program that computes the analytical solution of our problem.

The main building blocks of the program will be two functions that compute, for the elastic phase and for the plastic phase, the analytical functions that give the displacement and the velocity as functions of time.

Elastic response

We are defininining a function that, for a linear dynamic system, returns not the displacement or the velocity at a given time, but rather a couple of functions of time that we can use afterwards to compute displacements and velecities at any time of interest.

The response depends on the parameters of the dynamic system m, c, k, on the initial conditions x_0, v_0, and on the characteristics of the external load.

Here the external load is limited to a linear combination of a cosine modulated, a sine modulated (both with the same frequency ω) and a constant force,

$$P(t) = c_C \cos \omega t + c_S \sin \omega t + F,$$

but that's all that is needed for the present problem.

The particular integral being

$$\xi(t) = S \cos \omega t + R \sin \omega t + D,$$

substituting in the equation of motion and equating all the corresponding terms gives the undetermined coefficients in $\xi(t)$, then evaluation of the general integral and its time derivative for $t = 0$ permits to find the constants in the homogeneous part of the integral.

The final step is to define the displacement and the velocity function, according to the constants we have determined, and to return these two function to the caller

In [267]:
```
def resp_elas(m,c,k, cC,cS,w, F, x0,v0):
    wn2 = k/m ; wn = sqrt(wn2) ; beta = w/wn
    z = c/(2*m*wn)
    wd = wn*sqrt(1-z*z)
    # xi(t) = R sin(w t) + S cos(w t) + D
    det = (1.-beta**2)**2+(2*beta*z)**2
    R = ((1-beta**2)*cS + (2*beta*z)*cC)/det/k
    S = ((1-beta**2)*cC - (2*beta*z)*cS)/det/k
    D = F/k
    A = x0-S-D
    B = (v0+z*wn*A-w*R)/wd

    def x(t):
        return exp(-z*wn*t)*(A*cos(wd*t)+B*sin(wd*t))+R*sin(w*t)+S*cos(w*t)+D

    def v(t):
        return (-z*wn*exp(-z*wn*t)*(A*cos(wd*t)+B*sin(wd*t))
                  +wd*exp(-z*wn*t)*(B*cos(wd*t)-A*sin(wd*t))
                  +w*(R*cos(w*t)-S*sin(w*t)))
    return x,v
```

Plastic response

In this case the equation of motion is

$$m\ddot{x} + c\dot{x} = P(t),$$

the homogeneous response is

$$x(t) = A\exp(-\tfrac{c}{m}\,t) + B,$$

and the particular integral, for a load described as in the previous case, is again

$$\xi(t) = S\cos\omega t + R\sin\omega t + D.$$

Having computed R, S, and D from substituting ξ in the equation of motion, A and B by imposing the initial conditions,we can define the displacement and velocity functions and, finally, return these two functions to the caller.

In [268]:
```
def resp_yield(m,c, cC,cS,w, F, x0,v0):
    # csi(t) = R sin(w t) + S cos(w t) + Q t
    Q = F/c
    det = w**2*(c**2+w**2*m**2)
    R = (+w*c*cC-w*w*m*cS)/det
    S = (-w*c*cS-w*w*m*cC)/det
    # x(t) = A exp(-c t/m) + B + R sin(w t) + S cos(w t) + Q t
    # v(t) = - c A/m exp(-c t/m) + w R cos(w t) - w S sin(w t) + Q
    #
    # v(0) = -c A / m + w R + Q = v0
    A = m*(w*R + Q - v0)/c
    # x(0) = A + B + S = x0
    B = x0 - A - S

    def x(t):
        return A*exp(-c*t/m)+B+R*sin(w*t)+S*cos(w*t)+Q*t
    def v(t):
        return -c*A*exp(-c*t/m)/m+w*R*cos(w*t)-w*S*sin(w*t)+Q

    return x,v
```

An utility function

We need to find when

I. the spring yields
II. the velocity is zero

to individuate the three ranges of different behaviour

I. elastic
II. plastic
III. elastic, with permanent deformation.

We can use the simple and robust algorithm of *bisection* to find the roots for

$$x_{el}(t) = x_y \text{ and } \dot{x}_{ep}(t) = 0.$$

In [269]:
```
def bisect(f,val,x0,x1):
    h  = (x0+x1)/2.0
    fh = f(h)-val
    if abs(fh)<1e-8 : return h
    f0 = f(x0)-val
    if f0*fh > 0 :
        return bisect(f, val, h, x1)
    else:
        return bisect(f, val, x0, h)
```

The system parameters

In [270]:
```
mass = 1000.   # kg
k    = 40000.  # N/m
zeta = 0.03    # damping ratio
fy   = 2500.   # N
```

Derived quantities

The damping coefficient c and the first yielding displacement, x_y.

In [271]:
```
damp = 2*zeta*sqrt(k*mass)
xy = fy/k    # m
```

Load definition

Our load is a half-sine impulse

$$p(t) = \begin{cases} p_0 \sin(\frac{\pi t}{t_1}) & 0 \le t \le t_1, \\ 0 & \text{otherwise.} \end{cases}$$

In our exercise

In [272]:
```
t1 = 0.3     # s
w  = pi/t1   # rad/s
Po = 6000.   # N
```

The actual computations

Elastic, initial conditions, get system functions

In [273]:
```
x0=0.0     # m
v0=0.0     # m/s
x_next, v_next = resp_elas(mass,damp,k, 0.0,Po,w, 0.0, x0,v0)
```

Yielding time is

The time of yielding is found solving the equation $x_{\text{next}}(t) = x_y$

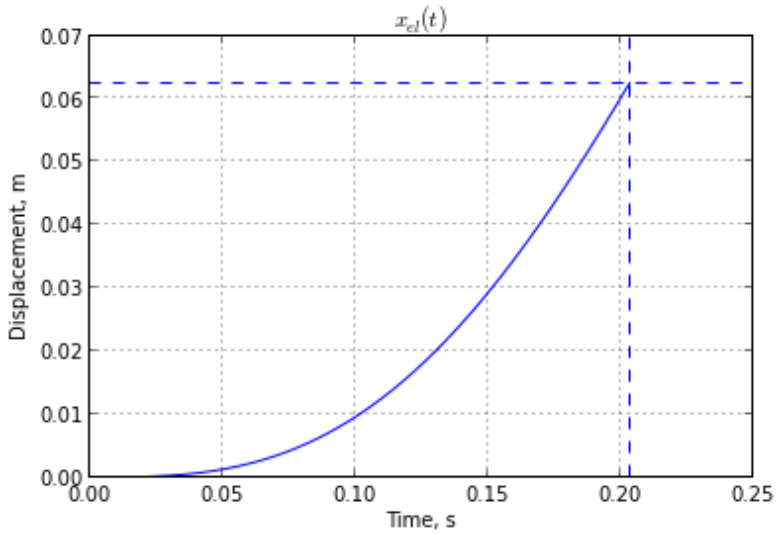
In [274]:
```
t_yield = bisect(x_next, xy, 0.0, t1)
print t_yield, x_next(t_yield)*k
```

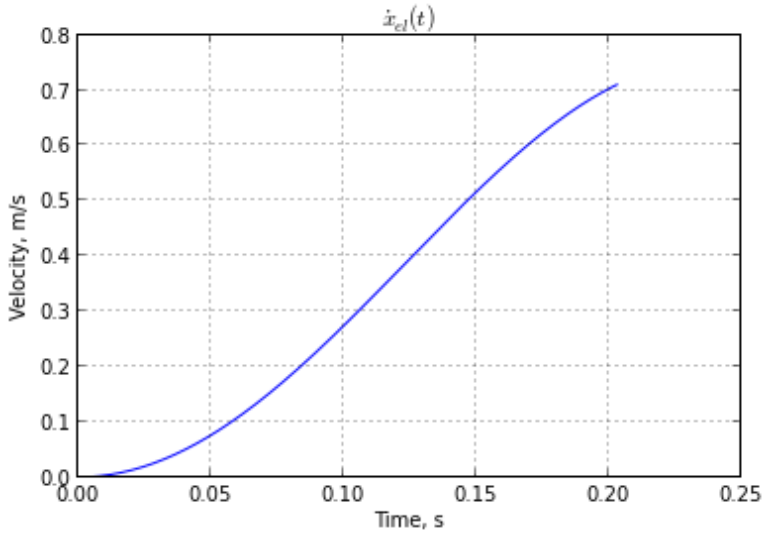
```
0.203265702724 2500.00009219
```

Type *Markdown* and LaTeX: α^2

In [275]:
```
t_el = linspace( 0.0, t_yield, 201)
x_el = vectorize(x_next)(t_el)
v_el = vectorize(v_next)(t_el)
# ------------------------------
figure(0);grid()
plot(t_el,x_el,
     (0,0.25),(xy,xy),'--b',
     (t_yield,t_yield),(0,0.0699),'--b')
title("$x_{el}(t)$")
xlabel("Time, s")
ylabel("Displacement, m")
# ------------------------------
figure(1);grid()
plot(t_el,v_el)
title("$\dot x_{el}(t)$")
xlabel("Time, s")
ylabel("Velocity, m/s")
```

Out[275]: <matplotlib.text.Text at 0x7fe764f2d210>





Preparing for EP response

First, the system state at t_y is the initial condition for the EP response

In [276]:
```
x0=x_next(t_yield)
v0=v_next(t_yield)
print x0, v0
```

```
0.0625000023047 0.709743249699
```

now, the load must be expressed in function of a *restarted time*,

$$\tau = t - t_y \;\rightarrow\; t = \tau + t_y \;\rightarrow\; \sin(\omega t) = \sin(\omega\tau + \omega t_y)$$

$$\rightarrow\; \sin(\omega t) = \sin(\omega\tau)\cos(\omega t_y) + \cos(\omega\tau)\sin(\omega t_y)$$

In [277]:
```
cS = Po*cos(w*t_yield)
cC = Po*sin(w*t_yield)

print Po*sin(w*0.55), cS*sin(w*(0.55-t_yield))+cC*cos(w*(0.55-t_yield))
```

```
-3000.0 -3000.0
```

Now we generate the displacement and velocity functions for the yielded phase, please note that the yielded spring still exerts a constant force f_y on the mass, and that this fact must be (and it is) taken into account.

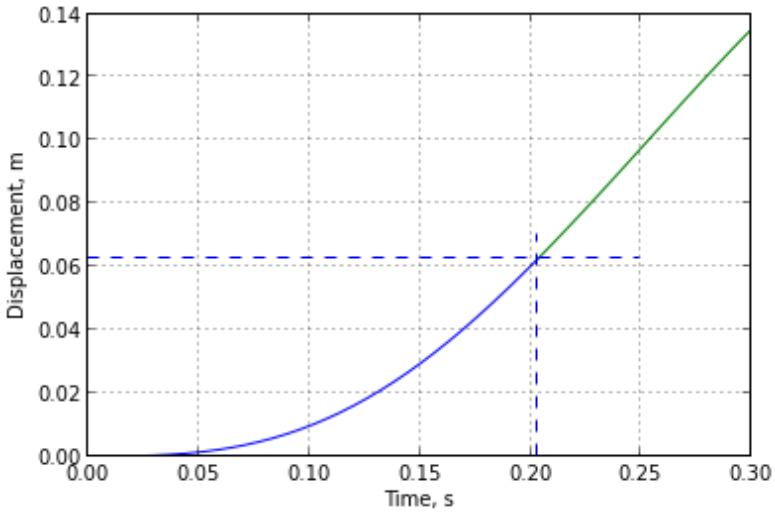
In [278]:
```
x_next, v_next = resp_yield(mass, damp,    cC,cS,w, -fy, x0,v0)
```

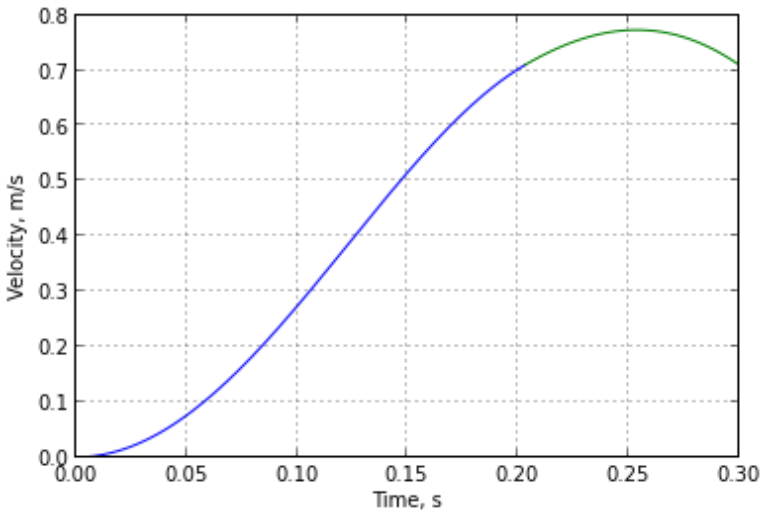
At this point I must confess that I have already peeked the numerical solution, hence I know that the velocity at $t = t_1$ is grater than 0 and I know that the current solution is valid in the interval $t_y \leq t \leq t_1$.

In [279]:
```
t_y1 = linspace(t_yield, t1, 101)
x_y1 = vectorize(x_next)(t_y1-t_yield)
v_y1 = vectorize(v_next)(t_y1-t_yield)
```

In [280]:
```
figure(3) ; grid()
plot(t_el,x_el, t_y1,x_y1,
     (0,0.25),(xy,xy),'--b',
     (t_yield,t_yield),(0,0.0699),'--b')
xlabel("Time, s")
ylabel("Displacement, m")
# ------------------------------
figure(4) ; grid()
plot(t_el, v_el,  t_y1, v_y1)
xlabel("Time, s")
ylabel("Velocity, m/s")
```

Out[280]: <matplotlib.text.Text at 0x7fe764f0dc50>





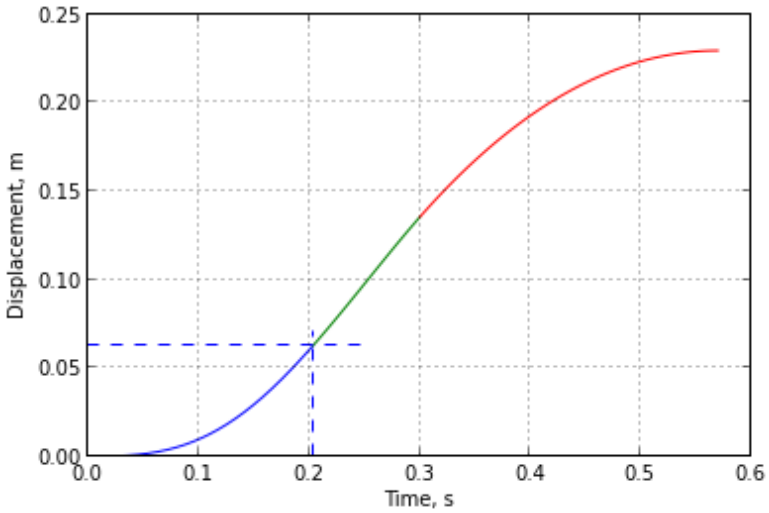
In the next phase, still it is $\dot{x} > 0$ so that the spring is still yielding, but now $p(t) = 0$, so we must compute two new state functions, starting as usual from the initial conditions (note that the yielding force is still applied)

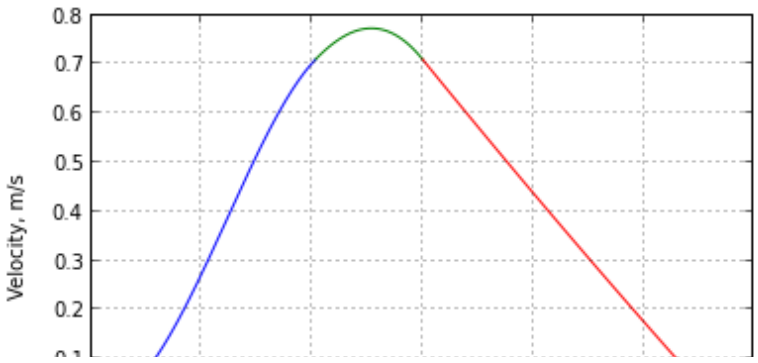
In [281]:
```
x0 = x_next(t1-t_yield)
v0 = v_next(t1-t_yield)
print x_0, v_0
x_next, v_next = resp_yield(mass, damp, 0, 0, w, -fy, x0, v0)

t2 = t1 + bisect( v_next, 0.0, 0, 0.3)
print t2
t_y2 = linspace( t1, t2, 101)
x_y2 = vectorize(x_next)(t_y2-t1)
v_y2 = vectorize(v_next)(t_y2-t1)
print x_next(t2-t1)
# -----------------------------
figure(5) ; grid()
plot(t_el,x_el, t_y1,x_y1, t_y2, x_y2,
     (0,0.25),(xy,xy),'--b',
     (t_yield,t_yield),(0,0.0699),'--b')
xlabel("Time, s")
ylabel("Displacement, m")
# -----------------------------
figure(6) ; grid()
plot(t_el, v_el,  t_y1, v_y1, t_y2, v_y2)
xlabel("Time, s")
ylabel("Velocity, m/s")
```

```
0.135209330223 0.709996878577
0.569713139534
0.229324078054
```

Out[281]: <matplotlib.text.Text at 0x7fe7655c4790>





Elastic unloading

The only point worth commenting is the constant force that we apply to our system.

The force-displacement relationship for an EP spring is

$$f_{\mathrm{E}} = k(x - x_{\mathrm{pl}}) = kx - k(x_{\max} - x_y)$$

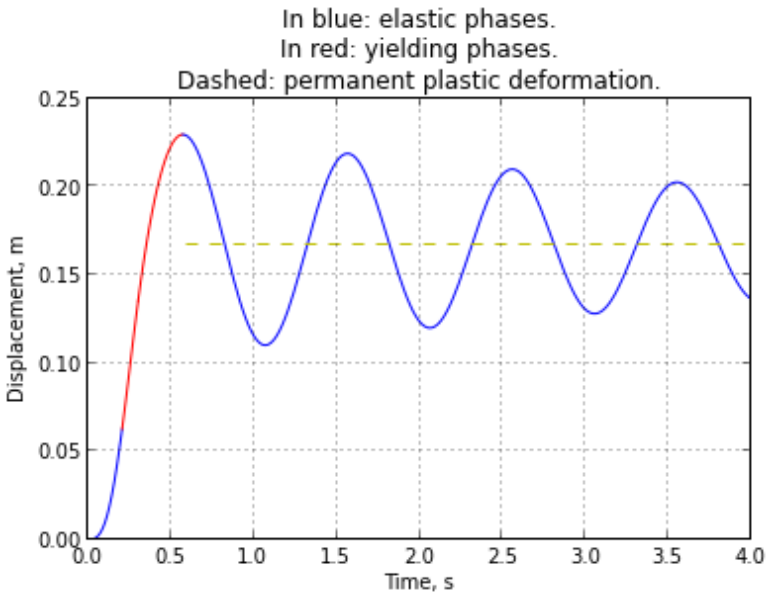
taking the negative, constant part of the last expression into the right member of the equation of equilibrium we have a constant term, as follows

In [282]:
```
x0 = x_next(t2-t1) ; v0 = 0.0
print x0, x_y2[-1]
x_next, v_next = resp_elas(mass,damp,k, 0.0,0.0,w, k*x0-fy, x0,v0)
t_e2 = linspace(t2,4.0,201)
x_e2 = vectorize(x_next)(t_e2-t2)
v_e2 = vectorize(v_next)(t_e2-t2)
```

```
0.229324078054 0.229324078054
```

In [283]:
```
# ------------------------------
figure(7) ; grid()
plot(t_el, x_el, '-b',
     t_y1, x_y1, '-r',
     t_y2, x_y2, '-r',
     t_e2, x_e2, '-b',
     (0.6, 4.0), (x0-xy, x0-xy), '--y')
title("In blue: elastic phases.\n"+
      "In red: yielding phases.\n"+
      "Dashed: permanent plastic deformation.")
xlabel("Time, s")
ylabel("Displacement, m")
```

Out[283]: <matplotlib.text.Text at 0x7fe764efc810>



Numerical solution

In [284]:
```
def p(t):
    if t<0.3:
        return 6000.0*sin(pi*t/0.3)
    return 0.0
```

Type *Markdown* and LaTeX: α^2

In [284]:
```

```