Constant

April 9, 2015

In [1]:
```
%pylab inline
%config InlineBackend.figure_format = 'svg'
import json
s = json.load( open("mplrc.json") )
matplotlib.rcParams.update(s)
matplotlib.rcParams['figure.figsize'] = 9,4
black="#404060" # plots containing "real black" elements look artificial
from IPython.core.display import HTML
def css_styling():
    styles = open("this_custom.css", "r").read()
    return HTML(styles)
css_styling()
```

```
Populating the interactive namespace from numpy and matplotlib

/usr/lib/python2.7/dist-packages/matplotlib/__init__.py:857: UserWarning: svg.embed_char_paths is depreca
  warnings.warn(self.msg_depr % (key, alt_key))
```

Out[1]: `<IPython.core.display.HTML at 0x7fdb1050f290>`

0.0.1 Constant acceleration algorithm

We are, again and again, integrating the equation of motion for the same example, that is a damped SDOF with period $T = 0.6$
, s, with a triangular loading with a peak value of 40 kN.

The loading function

In [2]:
```
def p(t):
    if t < 1.00 : return 4E5 * t
    if t < 3.00 : return 2E5 * (3-t)
    return 0.00
```

The SDOF system parameters and some derived values

In [3]:
```
mass = 6E05
T_n  = 0.60
wn   = 2*pi/T_n
k    = mass*wn**2
zeta = 0.02
wd   = wn * sqrt(1.00-zeta**2)
damp = 2*zeta*mass*wn
```

Initialization of the CA algorithm

```
In [4]: # time step duration
        h = 0.025

        # we require the response from 0 to 6 s, it is convnient
        # to define a slightly modified duration
        duration = 6.00 + h/2

        # The constants used by the algorithm
        k_ = k + 2*damp/h + 4*mass/h/h
        cv = 2*damp + 4*mass/h
        ca = 2* mass

        # We'll use these three containers to store our results
        x = [] ; v = [] ; t = []
```

Initial conditions of the system

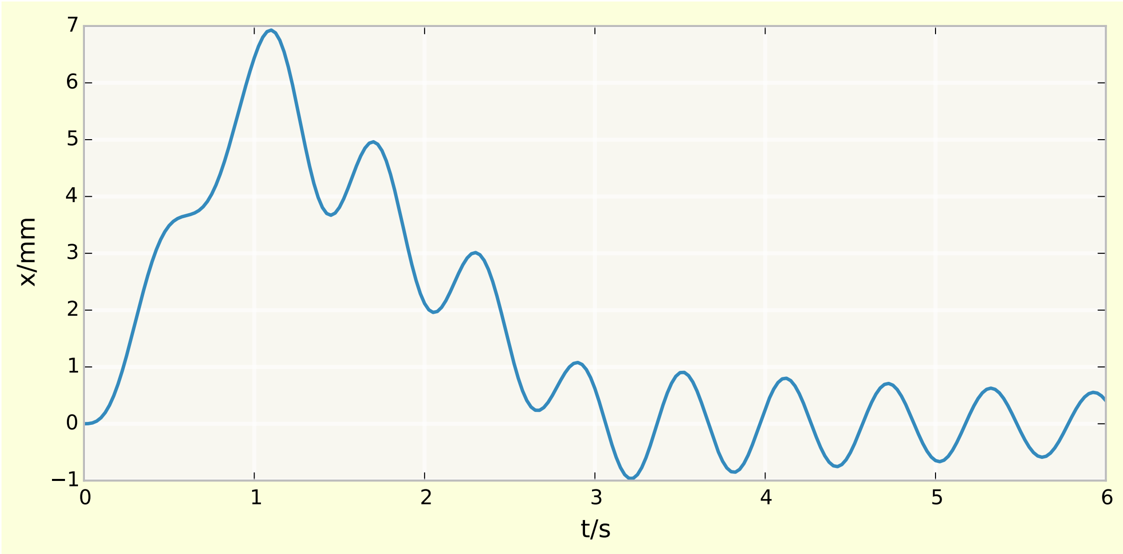
```
In [5]: T = 0.00
        X = 0.00
        V = 0.00
        P = p(T)
        A = (P - V*damp - X*k)/mass
```

Iteration of the elementary, step-wise incremental solution

```
In [6]: while T < duration:
            x.append(X) ; v.append(V) ; t.append(T)
            # print "%6.3f   %+12.10f %+12.10f" % (t, X, V)
            T = T+h
            Ph = p(T)
            dp_ = (Ph-P) + cv*V + ca*A
            dx  = dp_/k_
            dv  = 2*dx/h - 2*V
            X = X+dx ; V = V+dv
            P = Ph ; A = (P - damp*V - k*X)/mass
```

Plotting the resulting displacements

```
In [7]: plot(t, [1000*X for X in x]) ; xlim((0,6)) ; xlabel('t/s') ; ylabel('x/mm') ;
```



In [7]: