In [1]:
```
from numpy import *
import matplotlib.pyplot as plt
%matplotlib inline
```

Numerical Integration with E-P Behaviour

Here we are going to find the analytical solution of the 3rd problem of the homework #1 and next the numerical solution, using the constant acceleration algorithm.

Problem Data

and some easily derived quantities.

In [2]:
```
mass = 400
wn = 2*pi*4
zeta = 0.03
p0, t0 = 8200.0, 0.040
fy = 3200.0

stif = mass*wn**2
damp = 2*zeta*wn*mass
Tn = 2*pi/wn
beta = Tn/2/t0

Dst, xy = p0/stif, fy/stif
```

Particular Integral

$\xi(t) = C\sin(\beta\omega_n t) + D\cos(\beta\omega_n t)$

In [3]:
```
det = (1-beta**2)**2 + (2*zeta*beta)**2
C = (1-beta**2)/det
D = -2*zeta*beta/det
```

Homogeneous Solution and Initial Conditions

$x(t) = \exp(-\zeta\omega_n t)(A\sin(\delta\omega_n t) + B\cos(\delta\omega_n t)) + \xi(t), \qquad \delta = \sqrt{1-\zeta^2}.$

$x(0) = B + D = 0 \ \rightarrow \ B = -D.$

$v(0) = \omega_n(-\zeta B + \delta A + \beta C) = \omega_n(\zeta D + \delta A + \beta C) = 0 \ \rightarrow \ A = -(\zeta D + \beta C).$
$/\delta$

In [4]:

```
d = sqrt(1-zeta**2)
B = -D
A = (-zeta*D-beta*C) / d
```

Compute the forced response

For simplicity we use the adimensional time variable $a = \omega_n t$.

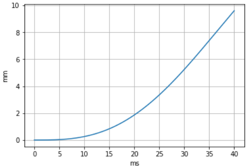
In [5]:
```
z, b = zeta, beta # shorter aliases

t = linspace(0, t0, 401)
a = wn*t

x = Dst*( exp(-z*a)*(A*sin(d*a)+B*cos(d*a)) + C*sin(b*a) + D*cos(b*a))
v = (exp(-z*a) * ( -z*(A*sin(d*a)+B*cos(d*a)) +d*(A*cos(d*a)-B*sin(d*a))))
v = Dst*wn*(v + b*(C*cos(b*a)-D*sin(b*a)))
```

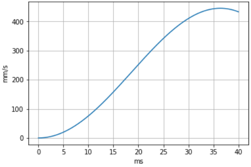
Plot of Displacements

In [6]:
```
plt.plot(1000*t, 1000*x); plt.grid()
plt.ylabel('mm') ; plt.xlabel('ms')
plt.show()
```



Plot of Velocities

In [7]:
```
plt.plot(1000*t, 1000*v); plt.grid()
plt.ylabel('mm/s') ; plt.xlabel('ms')
plt.show()
```



Free Elastic Response

$$
\begin{aligned}
x(\tau) &= \exp(-\zeta\omega_n\tau)(E\sin(\delta\omega_n\tau) + F\cos(\delta\omega_n\tau)), \quad \tau = t - t_0\\
x(0) &= F = x(t_0)\\
v(0) &= \omega_n(-\zeta F + \delta E) = v(t_0)
\end{aligned}
$$

In [8]:

```
x0, v0 = x[-1], v[-1] # the initial values of x, v are the last element
F = x0
E = (v0/wn+z*F)/d
```

Evaluate the Elastic Response

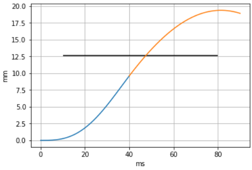
In [9]:
```
t2 = linspace(0,0.05,501)
a2 = wn*t2
x2 = exp(-z*a2)*(E*sin(d*a2)+F*cos(d*a2))
v2 = wn*exp(-z*a2)*( -z*(E*sin(d*a2)+F*cos(d*a2)) + d*(E*cos(d*a2)-F*sin(d*a2)))
```

Plot the Elastic Displacements

and an horizontal line indicating the value of x_y...

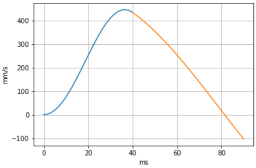
In [10]:
```
plt.plot(1000*t, 1000*x); plt.grid()
plt.plot(1000*(t2+t0), 1000*x2)
plt.ylabel('mm') ; plt.xlabel('ms')
plt.hlines(xy*1000, 10, 80)
print('Max elastic response is %f mm.'%(1000*max(x2)))
```

```
Max elastic response is 19.361712 mm.
```



Plot the Elastic Velocities

In [11]:
```
plt.plot(1000*t, 1000*v); plt.grid()
plt.plot(1000*(t2+t0), 1000*v2)
plt.ylabel('mm/s') ; plt.xlabel('ms');
```



Plastic Response

We need the elastic response, that we already obtained at discrete points of time, as functions of a time variable, so that we can find the instant of yielding (using a library function) and then, using the info on the initial velocity, find the integral of the plastic response.

Elastic Response as Funtions

In [12]:
```
f0 = lambda s: exp(-z*wn*s)*(E*sin(d*wn*s)+F*cos(d*wn*s))
f1 = lambda s: wn*exp(-z*wn*s)*( -z*(E*sin(d*wn*s)+F*cos(d*wn*s)) +
                                  d*(E*cos(d*wn*s)-F*sin(d*wn*s)))
```

Find t_y and the initial conditions for plastic branch

Now τ is $\tau = t - (t_0 + t_y)$...

In [13]:
```
from scipy.optimize import newton
ty = newton(lambda s: f0(s)-xy, 0.01)

xp = f0(ty)
vp = f1(ty)
```

Plastic Response

Particular Integral

The external force is $-f_y$, it is $\xi = S\tau$, substituting we have $cS = -f_y$ and $S = -f_y/c$

In [14]:
```
S = -fy/damp
```

Hom. solution and initial conditions

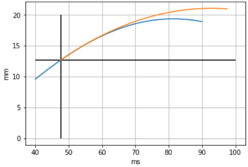
The general integral is $x=Q+R\exp(-2\zeta\omega_n\tau)+S\tau$,
$v=-2\zeta\omega_n R\exp(-2\zeta\omega_n\tau)+S$ and $x(0)=Q+R=x_p$,
$v(0)=-2\zeta\omega_n R+S=v_p$.

In [15]:
```
R = -(vp-S)/2/z/wn
Q = xp - R

Xp = lambda s: Q + R*exp(-2*z*wn*s) + S*s
Vp = lambda s: -2*z*wn*R*exp(-2*z*wn*s) + S
```

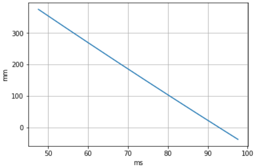
Plotting the displacements, plastic branch

In [16]:
```
plt.plot((t2+t0)*1000, 1000*f0(t2))      # elastic, blue
plt.plot((t2+ty+t0)*1000, 1000*Xp(t2))   # plastic, red
plt.hlines(xy*1000, 40, 100) ; plt.vlines((ty+t0)*1000, 0, 20)
plt.xlabel('ms') ; plt.ylabel('mm')
plt.grid();
```



Plotting the velocity in plastic range

In [17]:
```
plt.plot((t2+ty+t0)*1000, 1000*Vp(t2))
plt.xlabel('ms') ; plt.ylabel('mm') ; plt.grid();
```



As you can see, the final velocity is negative, this means that our solution is invalid (we are in an elastic unloading branch).

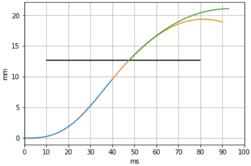
And the maximum displacement ($v = 0$) is...

In [18]:
```
tVp0 = newton(Vp, 0.05)
print('Time of max x: ', 1000*(tVp0+ty+t0), 'ms,\nMax x: ', 1000*Xp(tVp0),' mm.')
```

```
Time of max x:  92.87120039276594 ms,
Max x:  21.065084067663083  mm.
```

Plotting it all together, forced, free, plastic

In [19]:
```
tp = linspace(0, tVp0, 201)
plt.plot(1000*t, 1000*x) ; plt.plot(1000*(t2+t0), 1000*x2) # elastic
plt.plot(1000*(t0+ty+tp), 1000*Xp(tp))                       # plastic
plt.xlim((0,100)) ; plt.xticks(arange(0,101,10))
plt.ylabel('mm') ; plt.xlabel('ms')
plt.hlines(xy*1000, 10, 80); plt.grid();
```



Numerical Solution

Discretize Time and Force

In [20]:

```
Nsteps = 100
T = linspace(0, tVp0+ty+t0, Nsteps+1)
P = array([p0*sin(pi*t/t0) if t<t0 else 0 for t in T])
h = (tVp0+ty+t0)/Nsteps
```

Constants for Constant Acceleration Algorithm

In [21]:
```
DPm, DPc = 2*mass, 4*mass/h + 2*damp
km, kd = 4*mass/h/h, 2*damp/h
```

Initial Values, Initialize Storage to Save Our Results

In [22]:
```
x0, v0 = 0.0, 0.0 ; X = [x0]
```

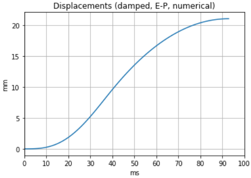
Stepping

Simplified algorithm, no Modified Newton-Raphson

In [23]:
```
for P0, DP in zip(P, P[1:]-P[:-1]):
    a0=(P0-damp*v0-(stif*x0 if x0<xy else fy))/mass
    DP_= DP + DPc*v0 + DPm*a0
    k_ = (stif if x0<xy else 0) + km + kd
    dx = DP_/k_
    # prepare for next step: update state variables, save X
    x0, v0 = x0+dx, 2*dx/h - v0 ;     X.append(x0)
X = array(X)
```

The never missing plot

In [24]:
```
plt.plot(1000*T, 1000*X); plt.grid()
plt.ylabel('mm') ; plt.xlabel('ms')
plt.xticks(range(0,101,10)) ; plt.xlim((0,100))
plt.title('Displacements (damped, E-P, numerical)')
plt.show()
```



Let's check the numerical result.

We print the numerical result, the exact result and the percentual of the difference.

In [25]:
```
xmxn = 1000*max(X)
xmxe = 1000*Xp(tVp0)
print('Max displacement, numerical:  ', xmxn, 'mm,')
print('Max displacement, exact:      ', xmxe, 'mm,')
print('(max_n-max_e)/max_e [%]:      ', '%f%%'%(100*(xmxn-xmxe)/xmxe))
```

```
Max displacement, numerical:   21.049417325626646 mm,
Max displacement, exact:       21.065084067663083 mm,
(max_n-max_e)/max_e [%]:       -0.074373%
```