CasADi tutorial

```
#
#
#
#
#
#
#
```

```
from numpy import *
import numpy
from casadi import *
from pylab import *
```

ODE integration

Let's construct a simple Van der Pol oscillator.

```
u = SX.sym("u")
x = SX.sym("x")
y = SX.sym("y")
ode = vertcat((1-y*y)*x-y+u,x)
```

DAE problem formulation as expected by CasADi's integrators:

```
dae  = {'x':vertcat(x,y), 'p':u, 'ode':ode}
```

The whole series of sundials options are available for the user

```
opts = {}
opts["fsens_err_con"] = True
opts["quad_err_con"] = True
opts["abstol"] = 1e-6
opts["reltol"] = 1e-6
tend=10
opts["t0"] = 0
opts["tf"] = tend
```

Create the Integrator

```
F = integrator("F", "cvodes", dae, opts)
print "%d -> %d" % (F.n_in(),F.n_out())
```

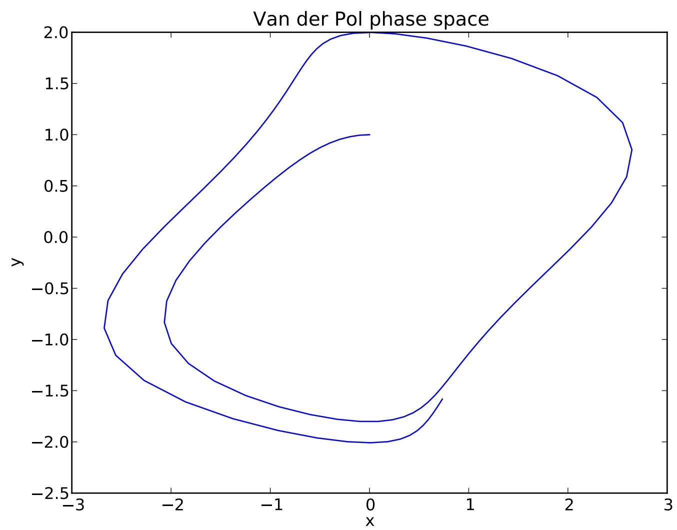
```
6 -> 6
```

Setup the Integrator to integrate from 0 to t=tend, starting at [x0,y0] The output of Integrator is the state at the end of integration. To obtain the whole trajectory of states, use Simulator:

```
ts=numpy.linspace(0,tend,100)
x0 = 0; y0 = 1
opts = {}
opts["fsens_err_con"] = True
opts["quad_err_con"] = True
opts["abstol"] = 1e-6
opts["reltol"] = 1e-6
opts["grid"] = ts
opts["output_t0"] = True
sim = integrator("sim", "cvodes", dae, opts)
sol = sim(x0=[x0,y0], p=0)

sol = sol['xf'].full().T
```

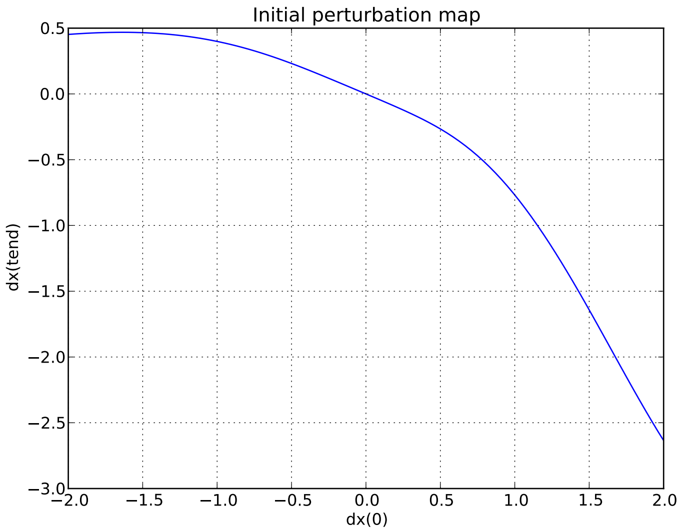
Plot the trajectory

```
figure()
plot(sol[:,0],sol[:,1])
title('Van der Pol phase space')
xlabel('x')
ylabel('y')
show()
```

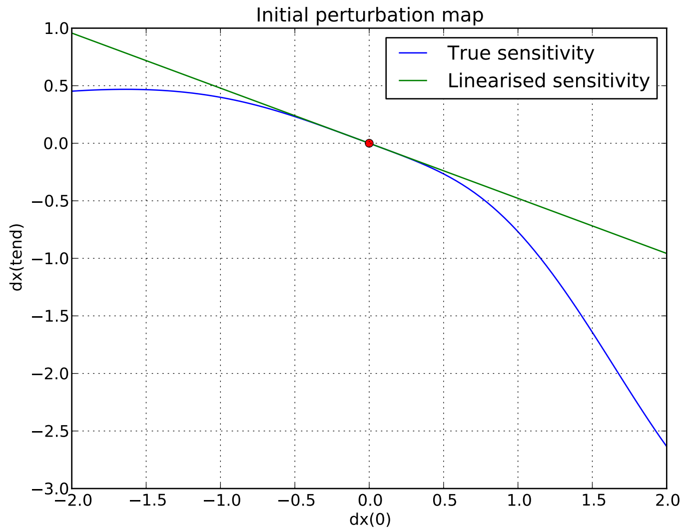


Sensitivity for initial conditions

```

def out(dx0):
        res = F(x0=[x0+dx0,y0])
        return res["xf"].full()
dx0=numpy.linspace(-2,2,100)
out = array([out(dx) for dx in dx0]).squeeze()
dxtend=out[:,0]-sol[-1,0]
figure()
plot(dx0, dxtend)
grid()
title('Initial perturbation map')
xlabel('dx(0)')
ylabel('dx(tend)')
show()
```

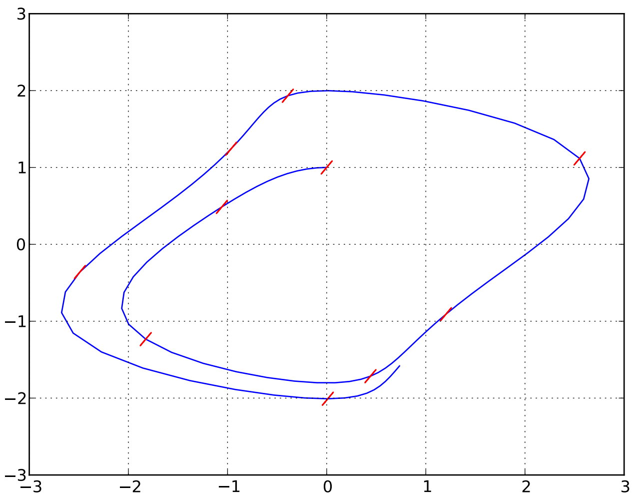



```
#

dintegrator = F.derivative(1,0)
res = dintegrator(der_x0=[x0,y0], fwd0_x0=[1,0])
A = res["fwd0_xf"][0]
A = float(A) # FIXME


plot(dx0,A*dx0)
legend(('True sensitivity','Linearised sensitivity'))
plot(0,0,'o')
show()
```



The interpetation is that a small initial circular patch of phase space evolves into ellipsoid patches at later stages.

```

def out(t):
        res = dintegrator(der_x0=[x0,y0], fwd0_x0=[1,0])
        A=res["fwd0_xf"].full()
        res = dintegrator(der_x0=[x0,y0], fwd0_x0=[0,1])
        B=res["fwd0_xf"].full()
        return array([A,B]).squeeze().T

circle = array([[sin(x),cos(x)] for x in numpy.linspace(0,2*pi,100)]).T

figure()
plot(sol[:,0],sol[:,1])
grid()
for i in range(10):
        J=out(ts[10*i])
        e=0.1*numpy.dot(J,circle).T+sol[10*i,:]
        plot(e[:,0],e[:,1],color='red')

show()
```



The figure reveals that perturbations perpendicular to the phase space trajectory shrink.

Symbolic intergator results

Since Integrator is just another Function, the usual CasADi rules for symbolic evaluation are active.

We create an MX 'w' that contains the result of a time integration with: - a fixed integration start time , t=0s - a fixed integration end time, t=10s - a fixed initial condition (1,0) - a free symbolic input, held constant during integration interval

```
u=MX.sym("u")
w = F(x0=MX([1,0]),p=u)["xf"]
```

We construct an MXfunction and a python help function 'out'

```
f=Function('f', [u],[w])

def out(u):
        w0 = f(u)
        return w0.full()

print out(0)
```

```
[[-2.54395395]
 [-0.43932676]]
```

```
print out(1)
```

```
[[-0.25397819]
 [ 1.39637624]]
```

Let's plot the results

```
uv=numpy.linspace(-1,1,100)

out = array([out(i) for i in uv]).squeeze()
figure()
plot(uv,out)
grid()
title('Dependence of final state on input u')
xlabel('u')
ylabel('state')
legend(('x','y'))
show()
```