

test_modele_dynamique

Laurent Perrinet (INT, UMR7289)

March 12, 2014

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

1 setting up experimental protocol

```
In [2]: from parametres import sliders, VPs, volume, p, kinects_network_config, d_x, d_y, d_z,
from modele_dynamique import Scenario, xyz2azel
events = [0, 0, 0, 0, 0, 0, 0, 0] # 8 types d'événements
print p['N'], VPs
DEBUG parametres , position croix: [ 10.80000019  5.19999981
1.36000001]
32 [{cz': 1.36, 'cy': 5.425, 'cx': 10.38, 'pc_min': 0.001, 'address':
'10.42.0.55', 'y': 1.69, 'x': 0.55, 'pc_max': 1000000.0, 'z': 1.36,
'foc': 21.802535306060136}, {'cz': 1.36, 'cy': 5.425, 'cx': 10.38,
'pc_min': 0.001, 'address': '10.42.0.56', 'y': 5.19, 'x': 0.55,
'pc_max': 1000000.0, 'z': 1.36, 'foc': 21.802535306060136}, {'cz':
1.36, 'cy': 5.425, 'cx': 10.38, 'pc_min': 0.001, 'address':
'10.42.0.51', 'y': 8.69, 'x': 0.55, 'pc_max': 1000000.0, 'z': 1.36,
'foc': 21.802535306060136}]

def simulpos(s, t, players):
    positions = []
    for player in players:
        positions.append([player['center'][0] + player['amp'][0]*np.cos(2*np.pi*t/player['clock']),
                         player['center'][1] + player['amp'][1]*np.cos(2*np.pi*t/player['clock']),
                         player['center'][2] + player['amp'][2]*np.cos(2*np.pi*t/player['clock'])])
    return positions

In [3]: def simul(p, players, dt=.01, t_stop=40., s_VP = 1,
             display=True, polar=False, N_step=400, limit=True):
    time = np.arange(0., t_stop, dt)
    N_time = len(time)
    n_players = len(players)
    positions_ = np.zeros((3, n_players, N_time))
    particles = np.zeros((12, p['N'], N_time))
    s = Scenario(p['N'], scenario, volume, [VPs[s_VP]], p, calibration)
    for i_t, t in enumerate(time):
        positions = simulpos(s, t, players)
        positions_[:, :, i_t] = np.array(positions).T
        s.do_scenario(positions=positions, events=events, dt=dt)
        particles[:, :, i_t] = s.particles#[0:6, :]\#s_VP*s.N:(s_VP+1)*s.N
    if display:
        fig = plt.figure(figsize=(18,10))
        T_step = max(int(N_time / N_step), 1)
```

```

if polar:
    rae_PC = xyz2azel(positions_, np.array([VPs[s_VP]['x'],
                                             VPs[s_VP]['y'],
                                             VPs[s_VP]['z']]))
    rae_VC = xyz2azel(particles[:3, :, :], np.array([VPs[s_VP]['x'],
                                                       VPs[s_VP]['y'],
                                                       VPs[s_VP]['z']]))

    for i_ax, axe in zip(range(3), ['r', 'a', 'e']):
        ax = fig.add_subplot(3, 1, 1+ i_ax)
        ax.plot(time, rae_PC[i_ax, :, :].T, '--')
        ax.plot(time[::T_step], rae_VC[i_ax, :, ::T_step].T, alpha=.5)
        #ax.errorbar(time[::T_step], particles[i_ax, :, ::T_step].mean(axis=0)
        #ax.errorbar(time[::T_step], particles[i_ax + 3, :, ::T_step].mean(axis=0)
        ax.set_ylabel(axe)

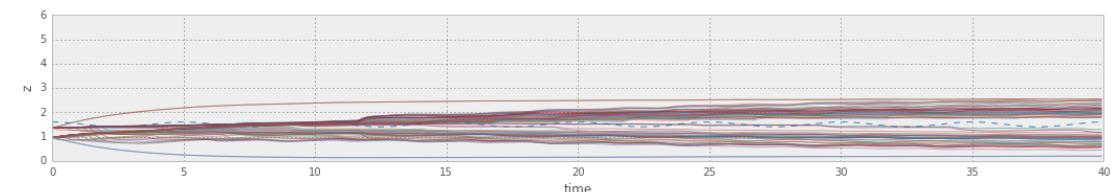
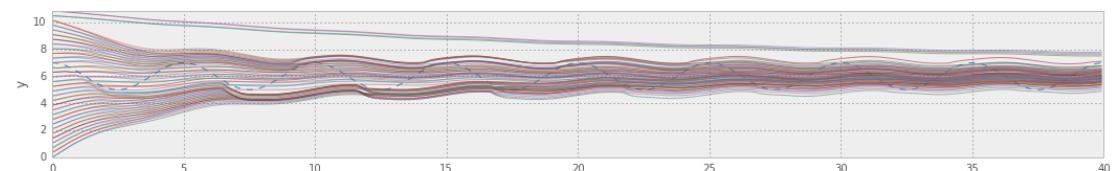
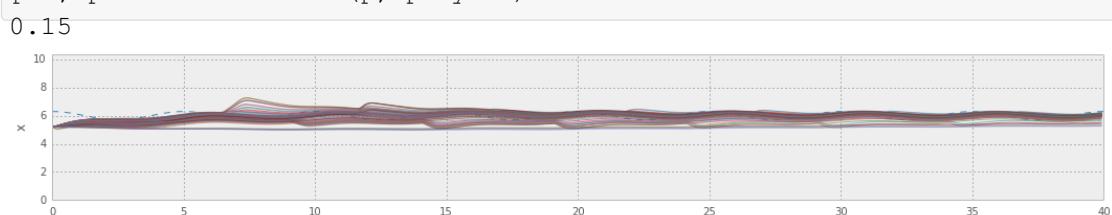
    else:
        for i_ax, axe, d in zip(range(3), ['x', 'y', 'z'], [d_x, d_y, d_z]):
            ax = fig.add_subplot(3, 1, 1+ i_ax)
            ax.plot(time, positions_[i_ax, :, :].T, '--')
            ax.plot(time[::T_step], particles[i_ax, :, ::T_step].T, alpha=.5)
            ax.plot(time[::T_step], particles[i_ax + 3, :, ::T_step].T, alpha=.5)
            #ax.errorbar(time[::T_step], particles[i_ax, :, ::T_step].mean(axis=0)
            #ax.errorbar(time[::T_step], particles[i_ax + 3, :, ::T_step].mean(axis=0)
            ax.set_ylabel(axe)
            ax.set_xlim([0., d])
            ax.set_xlabel('time')

    return positions_, particles

```

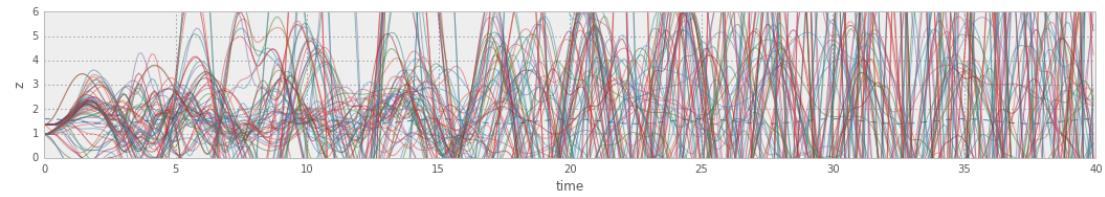
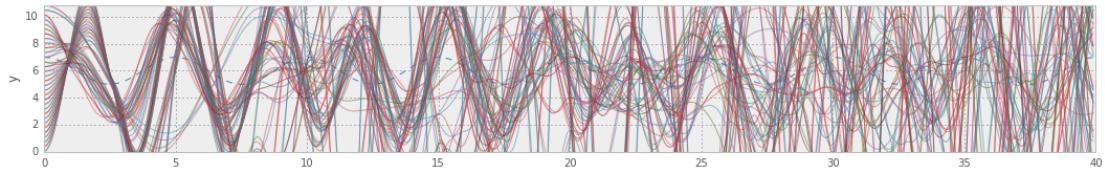
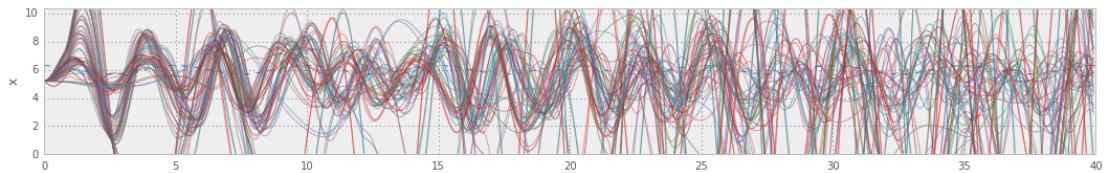
2 stabilité avec paramètres

```
In [5]: players = [ {'center': [6., 6, 1.5], 'amp': [.3, 1., .1], 'T': 5.}]
      print p['damp']
      pos, particles = simul(p, players)
```

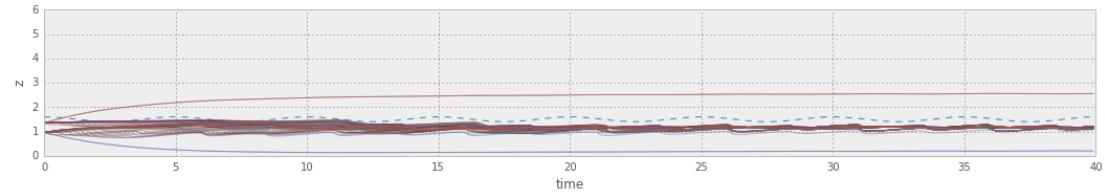
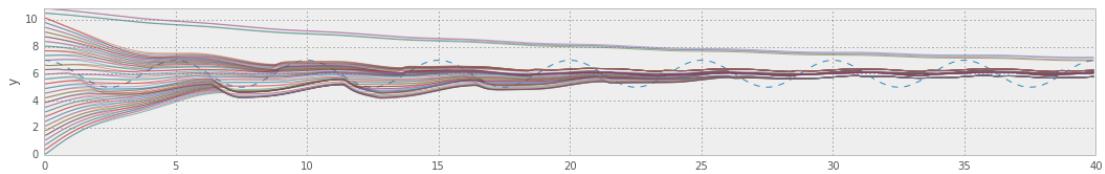
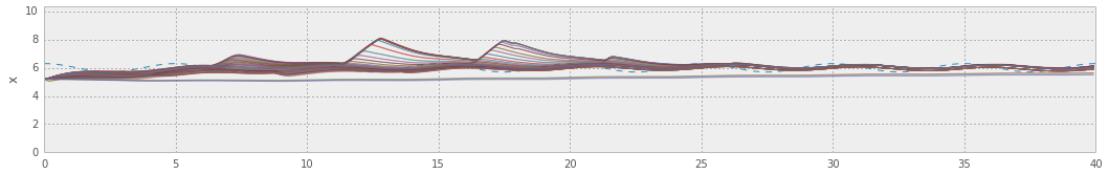


```
In [6]: p_ = p.copy()
p_['damp'] = .0
print p_['damp']
pos, particles = simul(p_, players)
```

0.0



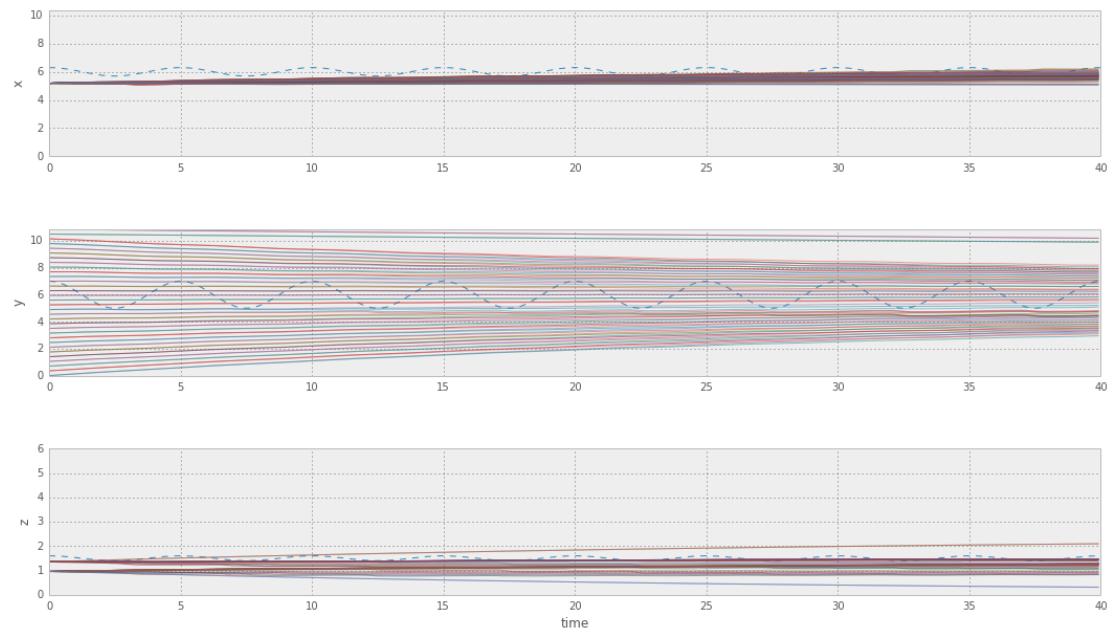
```
In [7]:  
p_ = p.copy()  
p_[‘G_repulsion’] = 0.  
pos, particles = simul(p_, players)
```



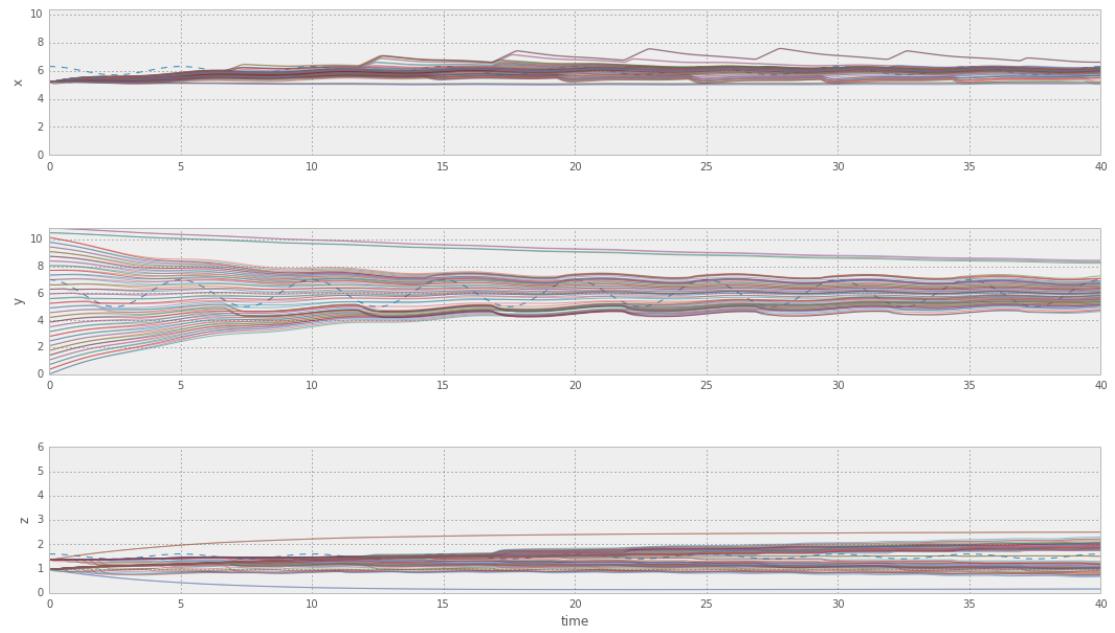
2.1 stabilité avec dt

```
In [8]:  
players = [ {‘center’: [6., 6, 1.5], ‘amp’: [.3,1.,.1], ‘T’: 5.} ]  
# players = [ {‘center’: [VPs[0][‘x’] + 1., VPs[0][‘y’], VPs[0][‘z’]], ‘amp’: [.3, 1.,  
for dt in np.logspace(-3., 0., 4, endpoint=False):  
    print ‘dt = ’, dt  
    pos, particles = simul(p, players, dt=dt)  
    plt.show()
```

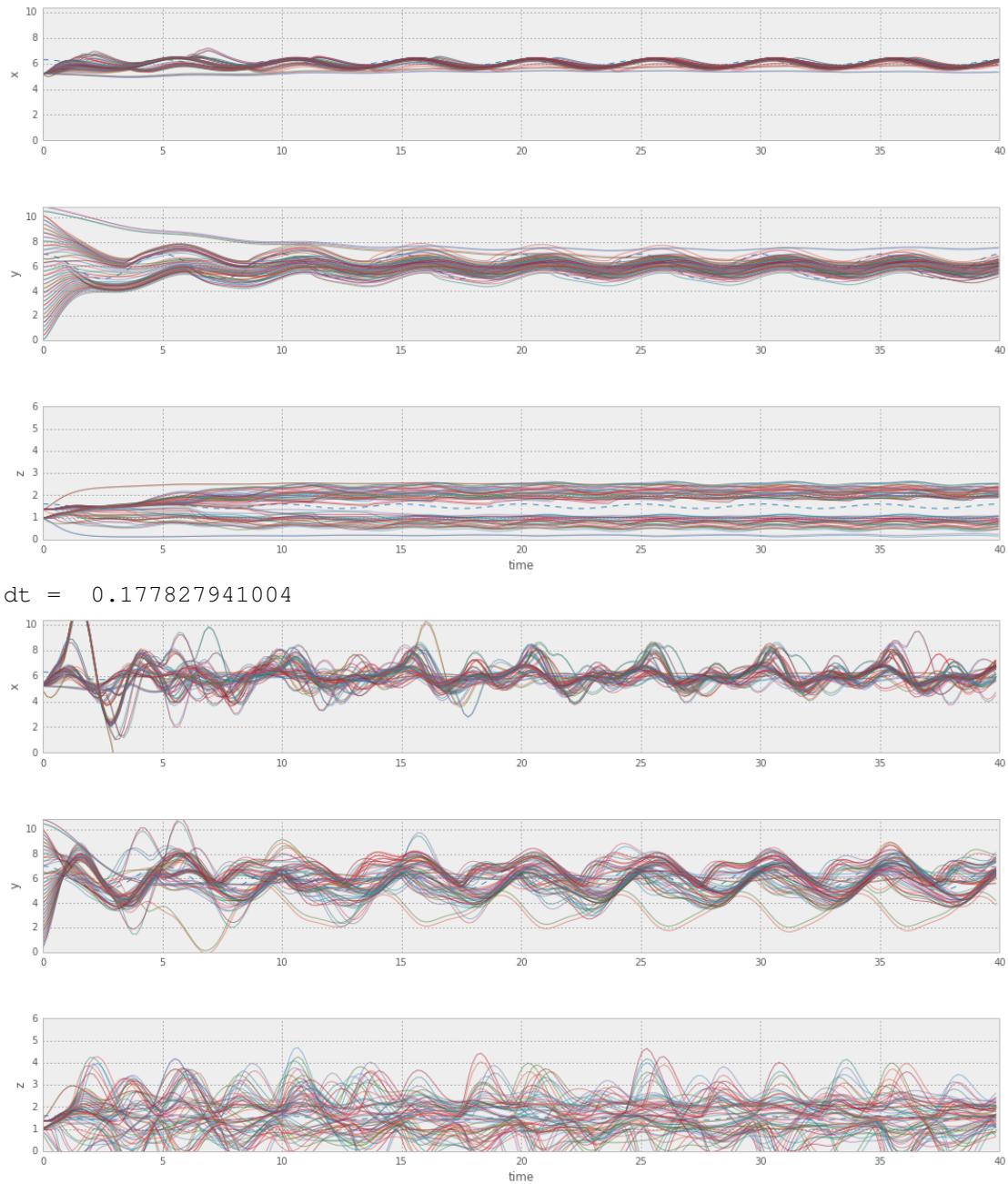
$dt = 0.001$



$dt = 0.0056234132519$



$dt = 0.0316227766017$



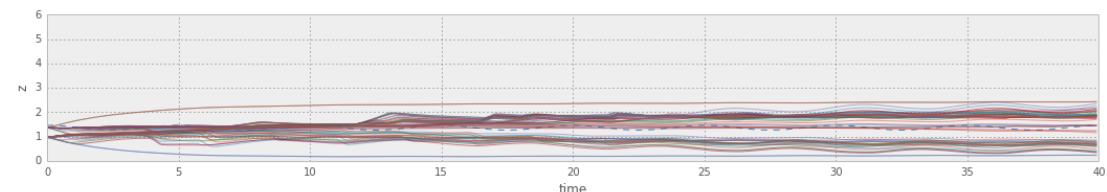
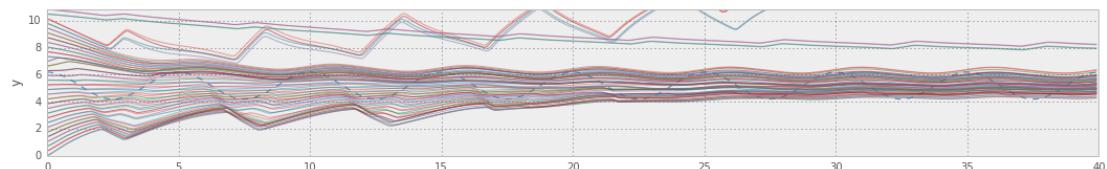
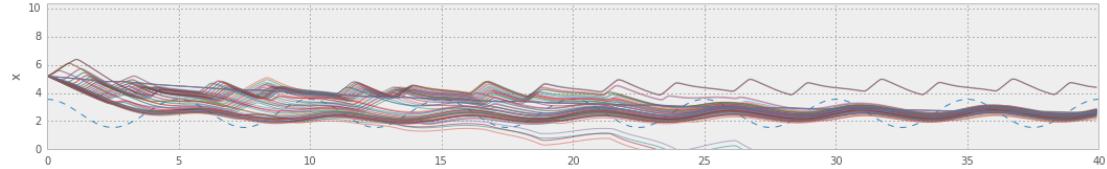
3 players instables

On définit un player seul qui se déplace devant le vidéoprojecteur, à une distance

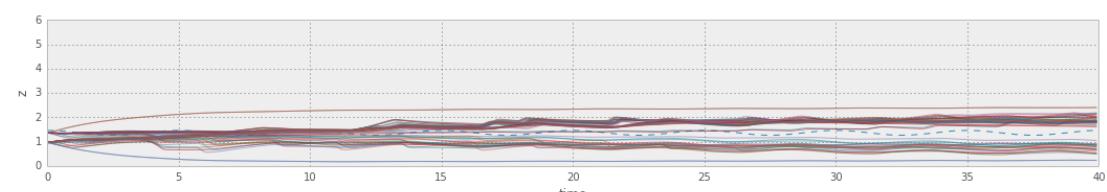
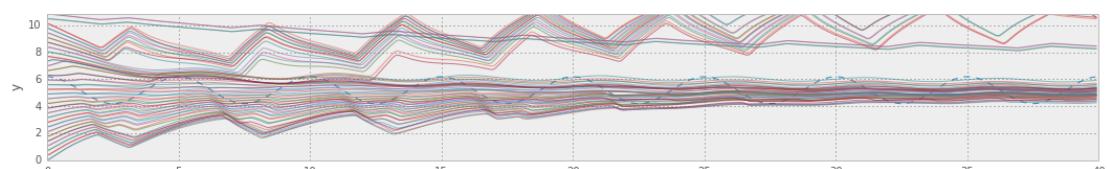
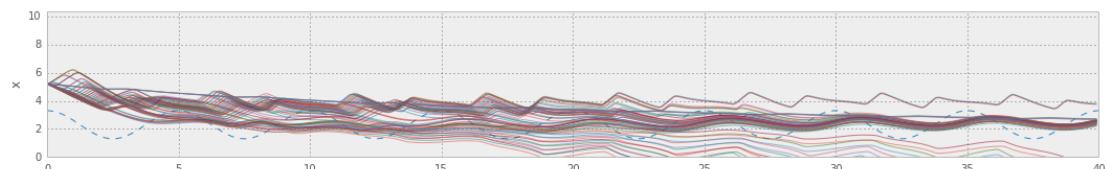
```
In [*]:  
    print dt  
    i_vp = 1  
    print VPs[i_vp]  
  
0.177827941004  
{'cz': 1.36, 'cy': 5.425, 'cx': 10.38, 'pc_min': 0.001, 'address':  
'10.42.0.56', 'y': 5.19, 'x': 0.55, 'pc_max': 1000000.0, 'z': 1.36,  
'foc': 21.802535306060136}
```

```
In [*]:  
for distance in np.linspace(2., 0., 8, endpoint=False):  
    print 'Distance du player au VP = ', distance  
    players = [ {'center': [VPs[i_vp]['x'] + distance, VPs[i_vp]['y'], VPs[i_vp]['z']],  
                'amp': [1., 1., .1], 'T': 5.}]  
    pos, particles = simul(p, players)  
    plt.show()
```

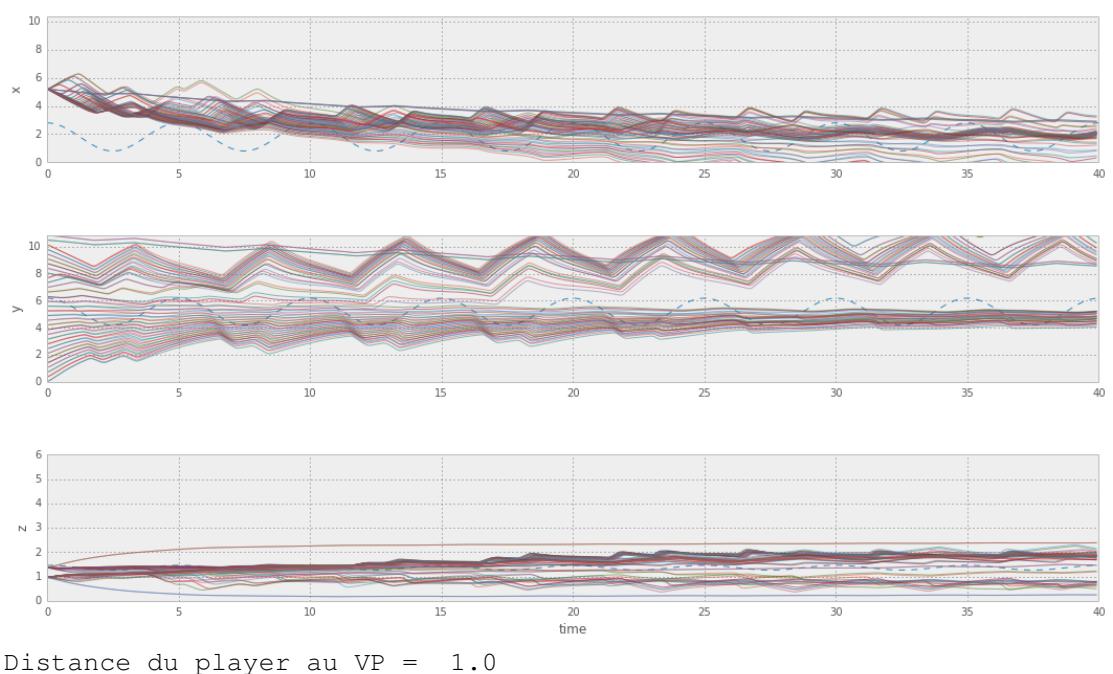
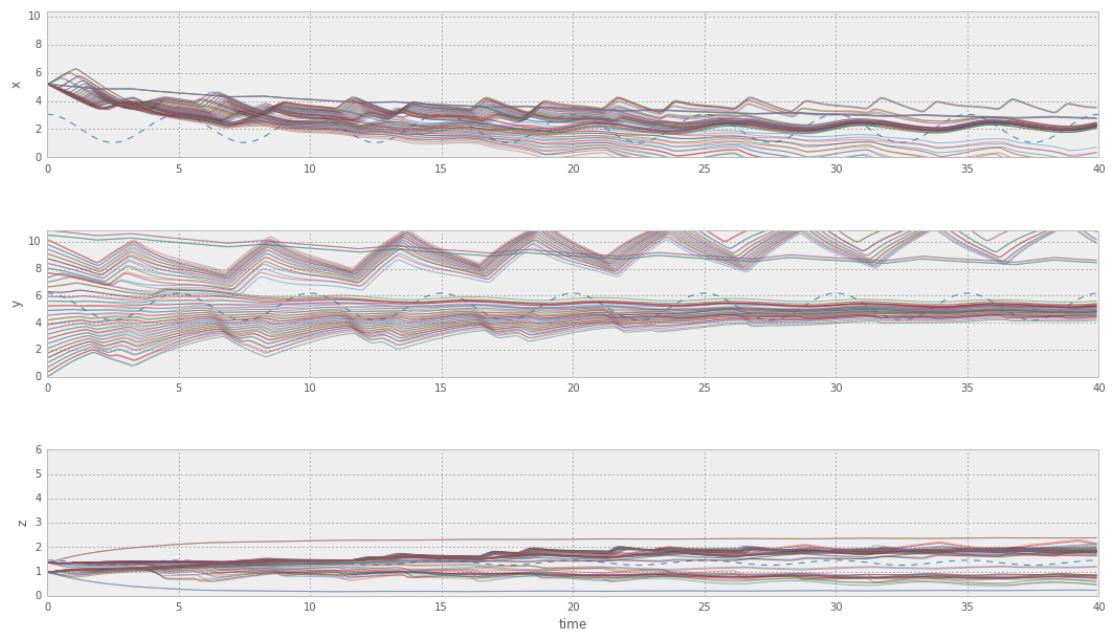
Distance du player au VP = 2.0

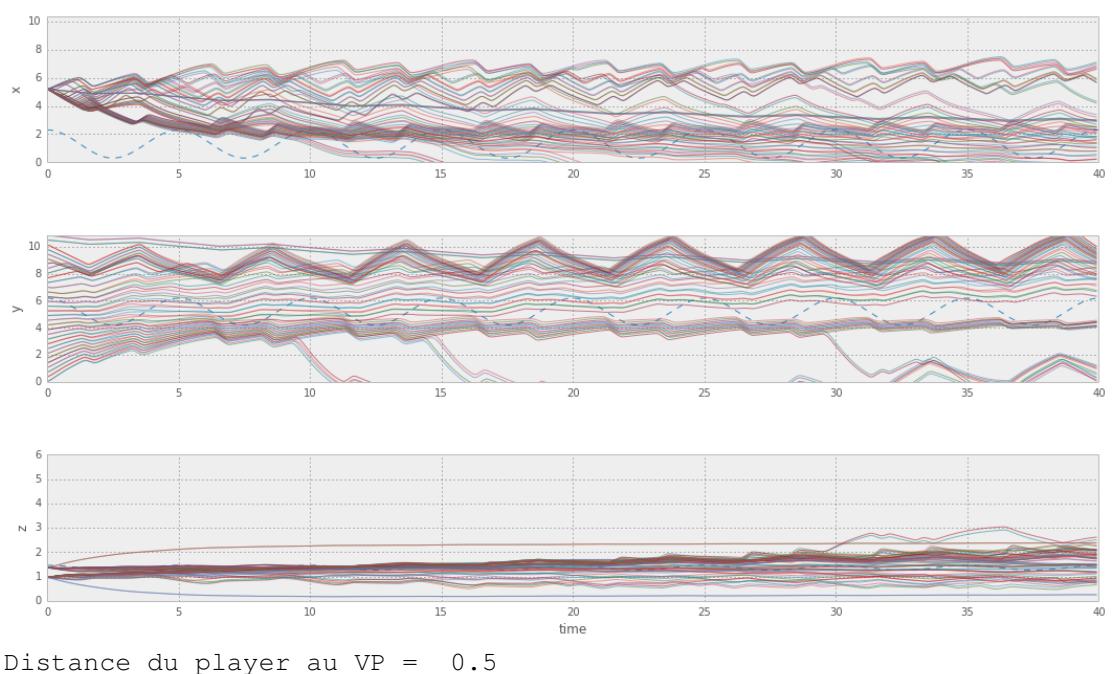
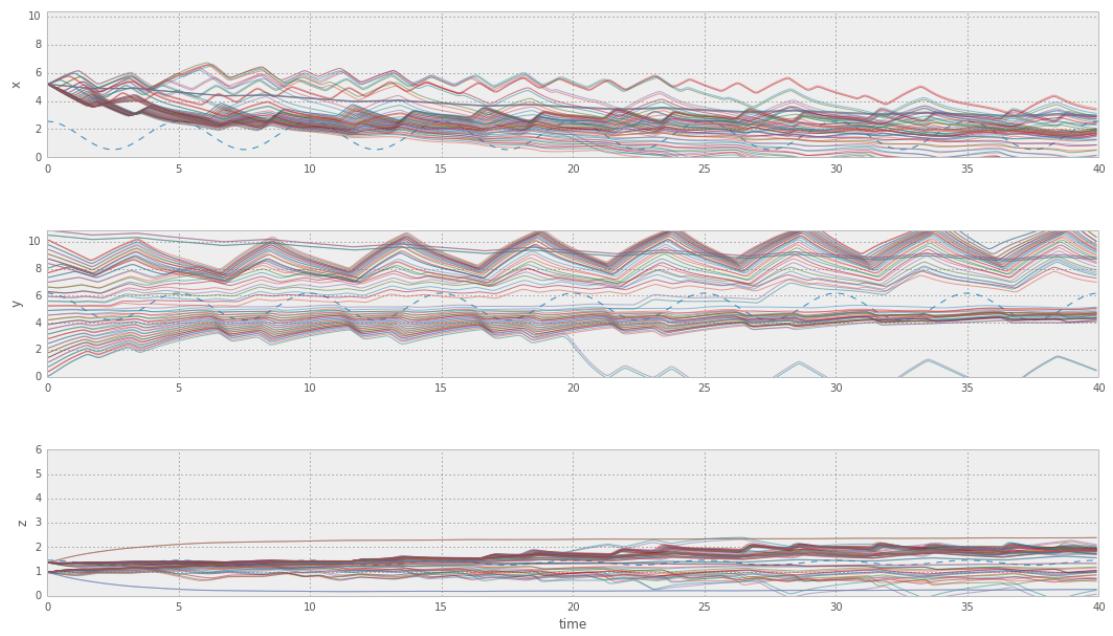


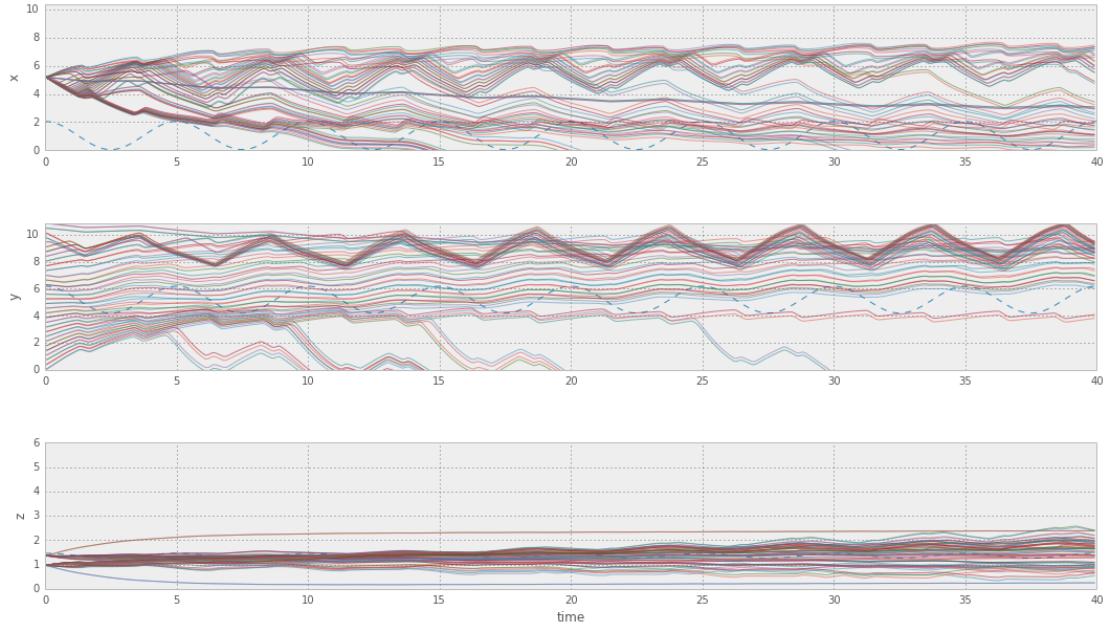
Distance du player au VP = 1.75



Distance du player au VP = 1.5







(note: il faut bien regarder l'amplitude des variations sur les axes)

4 multi players

```
In [*]: players = [{"center": [10., 3, 1.5], "amp": [.3,1.,.1], "T": 3.}, {"center": [10., 6, 1.5], "amp": [.3,1.,.1], "T": 5.}]
pos, particles = simul(p, players)
```

5 étude de la stabilité avec une autre personne

TODO: voir si on peut éviter de masquer des personnes dans le même axe en annihilant cette force et en donnant ainsi plus de forces aux interactions venant des autres VPs / dynamique

```
In [*]: for distance in np.linspace(2., 0., 8, endpoint=False):
    players = [{"center": [10., 6, 1.5], "amp": [.3,1.,.1], "T": 15.}]
    print 'Distance du player au VP = ', distance
    players = [{"center": [VPs[i_vp]['x'] + distance, VPs[i_vp]['y'], VPs[i_vp]['z']], "amp": [1., 1., .1], "T": 5.}]
    pos, particles = simul(p, players)
    plt.show()
```

6 étude de la stabilité avec quatre autres personnes

```
In [*]: for distance in np.linspace(2., 0., 8, endpoint=False):
    players = [{"center": [5., 6, 1.5], "amp": [1.3,1.,.1], "T": 15.}, {"center": [6., 4, 1.5], "amp": [2.3,1.,.1], "T": 3.}, {"center": [5., 3, 1.9], "amp": [4.3,1.,.1], "T": 15.}, {"center": [8., 4, 1.], "amp": [5.3,1.,.1], "T": 3.}]
    print 'Distance du player au VP = ', distance
    players = [{"center": [VPs[i_vp]['x'] + distance, VPs[i_vp]['y'], VPs[i_vp]['z']], "amp": [1., 1., .1], "T": 5.}]
```

```
pos, particles = simul(p, players)
plt.show()
```

7 étude quantitative paramétrique de la stabilité

```
In [*]:  
def simul_param(p, distance=0.25, N_distance=3, N_start=2., t_stop=10., i_vp=1, displa  
    print "Studying stability by changing model along variable(s) :", params  
  
    first_key = params.keys()[0]  
    N_params = np.size(params[first_key])  
    SE = np.zeros((4, N_params, N_distance))  
  
    for i_distance in np.arange(N_distance):  
        players = [{  
            'center': [5., 6, 1.5], 'amp': [1.3, 1., 1.], 'T': 15.},  
            {'center': [6., 4, 1.5], 'amp': [2.3, 1., 1.], 'T': 3.},  
            {'center': [VPs[i_vp]['x'] + (i_distance+1)*distance, VPs[i_vp]['y'],  
            'amp': [1., 1., 1.], 'T': 5.}]  
        for i_param, param in enumerate(params[first_key]):  
            new_params = p.copy()  
            for key in params.keys():  
                new_params[key] = params[key][i_param]  
  
            #v.execute('positions, particles = simul(players=players, display=False, p  
            positions, particles = simul(players=players, display=False, p=new_params,  
            error = (positions[:, -1, :][:, np.newaxis, :] - particles[:3, :, :])**2  
            error += (positions[:, -1, :][:, np.newaxis, :] - particles[3:6, :, :])**2  
            error_V = particles[6:, :, :]**2  
            SE[:3, i_param, i_distance] = np.sqrt(error[:, :, int(N_start/dt):]).mean(a  
            SE[-1, i_param, i_distance] = np.sqrt(error_V[:, :, int(N_start/dt):]).mean(a  
  
    if display:  
        fig = plt.figure(figsize=(18, 9))  
        for i_ax, axe in zip(range(4), ['x', 'y', 'z', 'v']):  
            ax = fig.add_subplot(4, 1, 1+i_ax)  
            ax.plot(params[first_key], SE[i_ax, :, :], 'o-')  
            ax.set_ylabel(r'SE_' + axe)  
            ax.set_xlabel(first_key)
```

7.1 paramètres dynamiques majeurs

```
In [*]:  
params = {'G_gravite_perc': np.linspace(.0, 100., 8, endpoint=True)}  
simul_param(p, **params)  
  
In [*]:  
#params = {'kurt_gravitation': np.linspace(-2, 5., 6, endpoint=True)}  
#simul_param(p, **params)  
  
In [*]:  
params = {'G_rot_perc': np.linspace(.0, 100., 8, endpoint=True)}  
simul_param(p, **params)  
  
In [*]:  
params = {'G_gravite_axis': np.linspace(.0, 35., 7, endpoint=True)}  
simul_param(p, **params)
```

7.2 paramètres dynamiques secondaires (textures)

```

In [*]: params = {'G_tabou': np.linspace(.0, 100., 11, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'G_poussee': np.linspace(.0, 1., 8, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'G_repulsion': np.linspace(.0, .4, 11, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'G_spring': np.linspace(.0, 50., 11, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'G_struct': np.linspace(.0, .1, 8, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'distance_struct': np.linspace(.0, .6, 8, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'distance_m': np.linspace(.0, .6, 8, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'distance_tabou': np.linspace(.0, .6, 8, endpoint=True)}
         simul_param(p, **params)

In [*]: params = {'l_seg_min': np.linspace(.1, .6, 6, endpoint=True)}
         simul_param(p, **params)

In [*]: #params = {'kurt_struct': np.linspace(-2, 0., 5, endpoint=True)}
         #simul_param(p, **params)

```

7.3 paramètres d'échelle

```

In [*]: params = {'scale': np.linspace(p['scale_max']*.7, p['scale_max']*1.4, 8, endpoint=True)
          simul_param(p, **params)

In [*]: params = {'eps': np.logspace(-4, -2, 8, endpoint=True)}
          simul_param(p, **params)

In [*]: params = {'speed_0': np.logspace(-2., 0., 8, endpoint=True)}
          simul_param(p, **params)

In [*]: params = {'damp': np.linspace(.0, .2, 11, endpoint=True)}
          simul_param(p, **params)

In [*]:

```