Extended & Unscented Kalman

April 14, 2015

```
In [98]: import numpy as np
         import scipy, scipy.linalg, scipy.signal
         from pykalman import AdditiveUnscentedKalmanFilter
         import matplotlib.pyplot as plt
         import pickle
         import pylab
         import csv

         %matplotlib inline
         pylab.rcParams['figure.figsize'] = (14.0, 8.0)
```

1 Filtre de Kalman etendu

```
In [2]: def filter_predict(transition_function,transition_function_jacobian,transition_covariance,
                           current_state_mean,current_state_covariance):
            """
                Compute the predict step of a first order Extended Kalman Filter

                :param transition_function: python function that applies one step in the dynamical mode
                :type transition_function: (nStateDim,) numpy.Array -> (nStateDim,) numpy.Array
                :param transition_function_jacobian: python function that computes the jacobian of the
                :type transition_function_jacobian: (nStateDim,) numpy.Array -> (nStateDim,nStateDim) n
                :param transition_covariance: covariance of the additive noise in the dynamical model
                :type transition_covariance: (nStateDim,nStateDim) numpy.Array
                :param current_state_mean: current state mean
                :type current_state_mean: (nStateDim,) numpy.Array
                :param current_state_covariance: current state covariance
                :type current_state_covariance: (nStateDim,nStateDim) numpy.Array
                :returns: predicted state mean, predicted state covariance
                :rtype: ( (nStateDim,) numpy.Array, (nStateDim,nStateDim) numpy.Array )
            """
            predicted_state_mean=transition_function(current_state_mean)

            transition_matrix=transition_function_jacobian(current_state_mean)
            predicted_state_covariance=transition_matrix.dot(current_state_covariance).dot(transition_ma

            return (predicted_state_mean,predicted_state_covariance)

        def filter_update(observation_function,observation_function_jacobian,observation_covariance,
                          predicted_state_mean,predicted_state_covariance,observation):
            """
                Compute the update step of a first order Extended Kalman Filter
```

```
        :param observation_function: python function that applies the observation model on a gi
        :type observation_function: (nStateDim,) numpy.Array -> (nObservationDim,) numpy.Array
        :param observation_function_jacobian: python function that computes the jacobian of the
        :type observation_function_jacobian: (nStateDim,) numpy.Array -> (nObservationDim,nStat
        :param observation_covariance: covariance of the additive noise in the observation mode
        :type observation_covariance: (nObservationDim,nObservationDim) numpy.Array
        :param predicted_state_mean: predicted state mean
        :type predicted_state_mean: (nStateDim,) numpy.Array
        :param predicted_state_covariance: predicted state covariance
        :type predicted_state_covariance: (nStateDim,nStateDim) numpy.Array
        :param observation: current observation
        :type observation: (nObservationDim,) numpy.Array
        :returns: new_state_mean, new state covariance
        :rtype: ( (nStateDim,) numpy.Array, (nStateDim,nStateDim) numpy.Array )
    """
    error=observation-observation_function(predicted_state_mean)

    observation_matrix=observation_function_jacobian(predicted_state_mean)

    S=observation_matrix.dot(predicted_state_covariance).dot(observation_matrix.T)+observation_
    K=predicted_state_covariance.dot(observation_matrix.T).dot(np.linalg.inv(S))

    new_state_mean=predicted_state_mean+K.dot(error)
    new_state_covariance=predicted_state_covariance-K.dot(S).dot(K.T)

    return (new_state_mean,new_state_covariance)
```

In [32]:
```
data = pickle.load(open('pendulum.pick', 'rb'))

states = data['states']
observations = data['observations']
param = data['param']
time = data['time']
```

In [44]:
```
param
```

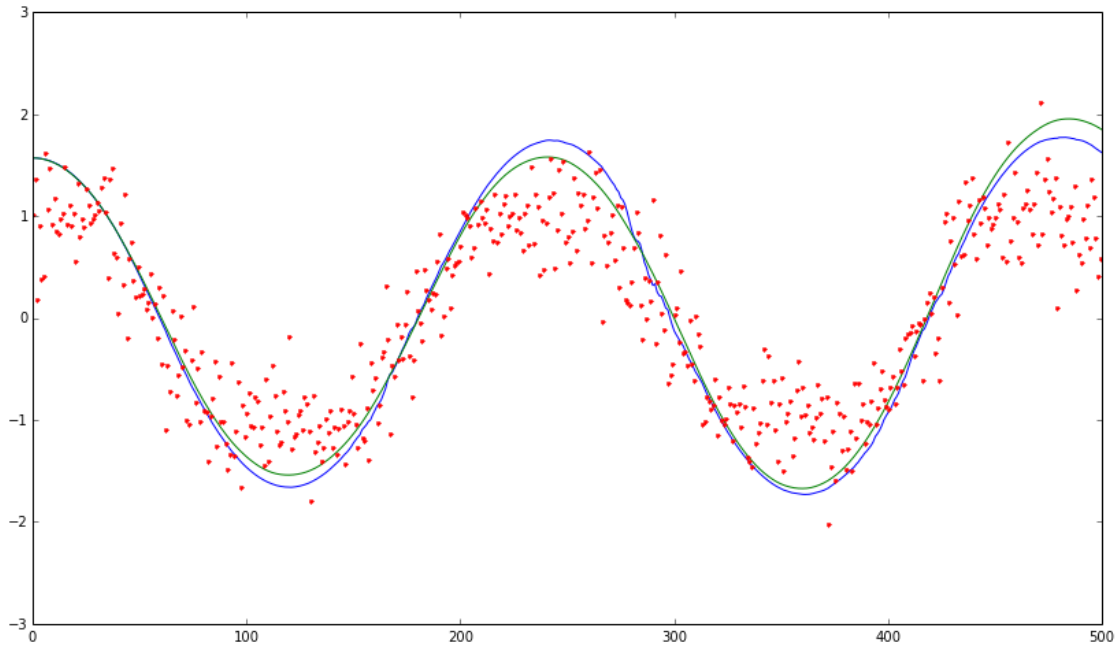
Out[44]:
```
{'dt': 0.01, 'g': 9.81, 'p0': array([[ 0.001,  0.   ],
        [ 0.   ,  0.001]]), 'q': 0.05, 'r': 0.32, 'x0': array([ 1.57079633,  0.        ])}
```

In [84]:
```
f  = lambda x : np.array([x[0] + x[1] * param['dt'], x[1] - param['g'] * np.sin(x[0]) * param[
jf = lambda x : np.array([[1, param['dt']],
                          [-param['g'] * np.cos(x[0]) * param['dt'], 1]])

h  = lambda x : np.array([np.sin(x[0])])
jh = lambda x : np.array([np.cos(x[0]), 0]).reshape((1,2))

Q = np.array([[param['q']*param['dt']**3/3, param['q']*param['dt']**2/2],
              [param['q']*param['dt']**2/2, param['q']*param['dt']]])
```

In [96]:
```
# data size
n = int(len(observations))

# init
stateMean = np.zeros((n+1, 2))
stateCov  = np.zeros((n+1, 2, 2))
```

```
stateMean[0] = param['x0']
stateCov[0]  = param['p0']

for i in range(n):
    (stateMeanPred,  stateCovPred)  = filter_predict(f, jf, Q, stateMean[i], stateCov[i])
    (stateMean[i+1], stateCov[i+1]) = filter_update(h, jh, np.array([param['r']]), stateMeanPr


plt.plot(stateMean[:,0])
plt.plot(states[:,0])
plt.plot(observations, '.')
```

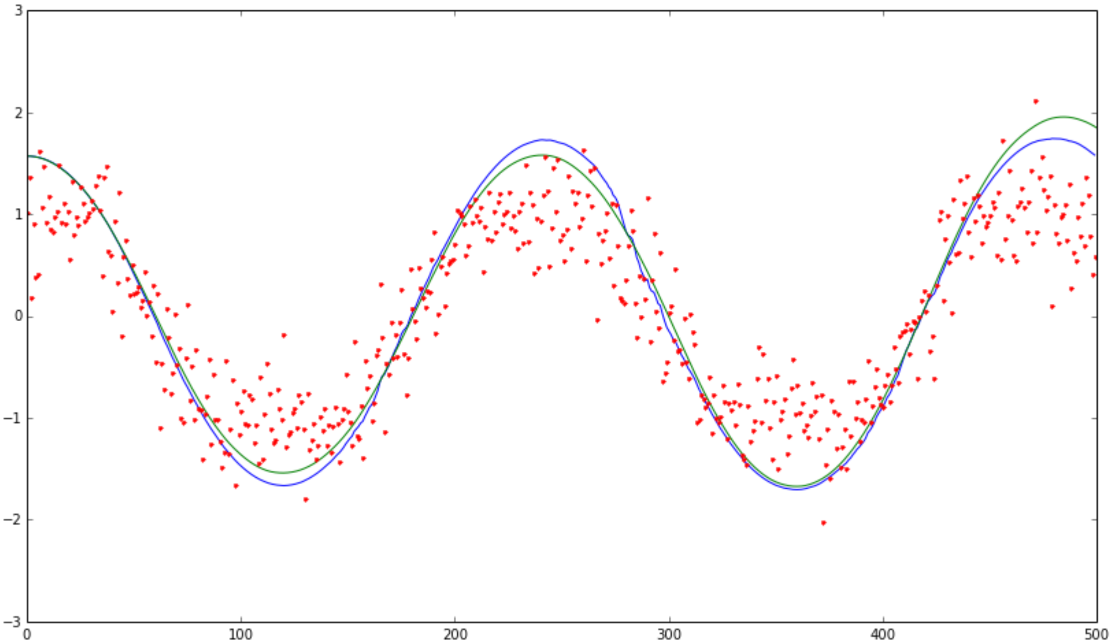
Out[96]: [<matplotlib.lines.Line2D at 0x151df470>]



2 Unscented Kalman Filter

In [107]:
```
ukf = AdditiveUnscentedKalmanFilter(f, h, Q, param['r'], param['x0'], param['p0'])
stateMean = ukf.filter(observations)[0]

plt.plot(stateMean[:,0])
plt.plot(states[:,0])
plt.plot(observations, '.')
```

Out[107]: [<matplotlib.lines.Line2D at 0x16c21d30>]


3
2
1
0
−1
−2
−3
0
100
200
300
400
500