Good Example

$$x_{n+1} = (I + \eta \begin{bmatrix} 0.1 & 0.5 \\ 0 & 0.1 \end{bmatrix}) x_n + \sqrt{\eta} \begin{bmatrix} 0.7 & -0.6 \\ 0 & 0.7 \end{bmatrix} u_n, \quad x_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$y_n = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_n + \sigma_0 v_n, \quad \sigma_0 = 0.5$$

In [1]:

```
#----------------------------------------
# library loading
#----------------------------------------
import time
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

#set suppress to not use scientific counting
np.set_printoptions(suppress=True)

#----------------------------------------
# Initialization Math Model
#----------------------------------------
cpu_start=time.perf_counter()

dimX=2; dimY=2
sigma0_train=0.5; eta=0.005
N=1000; n0=50; N_sample= 1000

#Deterministic Matrix
F0=np.array([[0.1,0.5],[0,0.1]])
F=np.eye(dimX)+eta*F0
G=np.sqrt(eta)*np.array([[0.7,-0.6],[0,0.7]])
H=np.array([[1,0],[0,1]])
x0=np.array([[1],[-1]])
#Covariance Matrix for random variable
Q0=np.eye(dimX) #covariance of random variable u_n
R0=sigma0_train*sigma0_train*np.eye(dimY) #covariance of random variable v_n

# generate u_n, v_n
# 1-d Gaussian: np.random.default_rng().normal(mean, std, size)
# n-d Gaussian: np.random.default_rng().multivariate_normal(mean,cov,size)
# note to reshape multivariate normal random variable to column vector.

rng=np.random.default_rng()
u=[rng.multivariate_normal(np.zeros(dimX),np.eye(dimX),1).reshape(dimX,1) for i in rang
e(N)]
v=[rng.multivariate_normal(np.zeros(dimY),np.eye(dimY),1).reshape(dimY,1) for i in rang
e(N+1)]
u=np.array(u)
v=np.array(v)

#----------------------------------------
# Monte Carlo Simulation for once
#----------------------------------------
def mc_simulation(F,G,H,u,v,N):
    """Monte Carlo Simulation
       N: time step horizon."""
    x_raw=np.zeros((N+1,dimX,1)); x_raw[0]=x0
    y_raw=np.zeros((N+1,dimY,1)); y_raw[0]=H@x0+sigma0_train*v[0] #!!!
    for k in range(N):
```

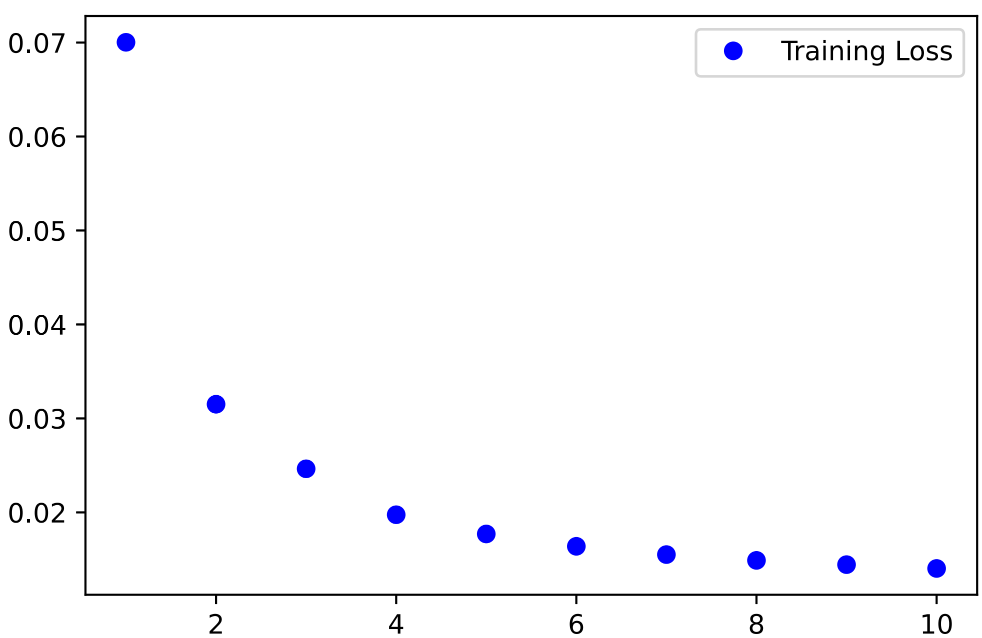
```
        x_raw[k+1]=F@x_raw[k]+G@u[k]  #!!!
        y_raw[k+1]=H@x_raw[k+1]+sigma0_train*v[k+1] #!!!
    return x_raw, y_raw

#----------------------------------------
# Kalman Filtering Algorithm
#----------------------------------------
def kalman_filtering(F,G,H,Q0,R0,x0,y_raw,N):
    """Kalman Filtering Algorithm"""
    #caution: need to specific x is dimX x 1 to be column vector
    x_hat=np.zeros((N+1,dimX,1)); x_hat[0]=x0
    R=np.zeros((N+1,dimX,dimX)); R[0]=np.zeros((dimX,dimX))  #!!!

    for k in range(N):
        #y_raw has to be column array or vector.
        inv=np.linalg.inv(H@R[k]@H.T+R0)
        x_hat[k+1]=F@x_hat[k]+F@R[k]@H.T@inv@(y_raw[k]-H@x_hat[k]) #!!!
        R[k+1]=F@(R[k]-R[k]@H.T@inv@H@R[k])@F.T+G@Q0@G.T           #!!!

    x_bar=[x_hat[k]+R[k]@H.T@np.linalg.inv(H@R[k]@H.T+R0)@(y_raw[k]-H@x_hat[k]) for k i
n range(N+1)]
    x_bar=np.array(x_bar) #make list to np.array

    return x_hat, x_bar

#------------------------------
# Generating tons of samples
#------------------------------

def sample_generator():

    datas=np.zeros(((N-n0+2)*N_sample,n0,dimY)) #for each sample path, we have N-n0+2 d
ata
    labels=np.zeros(((N-n0+2)*N_sample,dimX))

    x_bars=np.zeros(((N-n0+2)*N_sample,dimX)) #store Kalman filtering estimation value.
    x_hats=np.zeros(((N-n0+2)*N_sample,dimX))

    x_raws=np.zeros((N_sample, N+1, dimX, 1))
    y_raws=np.zeros((N_sample, N+1, dimY, 1))

    for i in range(N_sample):
        data=np.zeros((N-n0+2,n0,dimY)) #store data for each sample
        label=np.zeros((N-n0+2,dimX))
        # call mc_simulation function to generate sample
        x_raw,y_raw=mc_simulation(F,G,H,u,v,N)
        x_raws[i]=x_raw; y_raws[i]=y_raw

        # call kalman_filtering function to compute estimation
        # make sure here y_raw to be column vector
        x_hat, x_bar=kalman_filtering(F,G,H,Q0,R0,x0,y_raw,N)

        # convert x_raw...into row vector
        x_raw=x_raw.reshape(N+1,dimX)
        y_raw=y_raw.reshape(N+1,dimY)
        x_hat=x_hat.reshape(N+1,dimX)
        x_bar=x_bar.reshape(N+1,dimX)
```

```python
        # make data and label for each sample
        for k in range(N-n0+2):
            data[k]=y_raw[k:k+n0]
            label[k]=x_raw[k+n0-1]

        # put data and label into datas and labels with i representing sample number
        datas[i*(N-n0+2):(i+1)*(N-n0+2)]=data
        labels[i*(N-n0+2):(i+1)*(N-n0+2)]=label
        x_hats[i*(N-n0+2):(i+1)*(N-n0+2)]=x_hat[n0-1:]
        x_bars[i*(N-n0+2):(i+1)*(N-n0+2)]=x_bar[n0-1:]

    return datas,labels,x_hats,x_bars,x_raws,y_raws
```

In [2]:
```
#-----------------------------
# Data Preparation
#-----------------------------
# call sample_generator function to generate sample
datas, labels, x_hats,x_bars,x_raws, y_raws=sample_generator()
datas=datas.reshape(((N-n0+2)*N_sample,dimY*n0))
# convert numpy array into pandas dataframe
datas=pd.DataFrame(datas)
labels=pd.DataFrame(labels)
x_hats=pd.DataFrame(x_hats)
x_bars=pd.DataFrame(x_bars)

from sklearn.model_selection import train_test_split
seed1=3
np.random.seed(seed1)
training_data, test_data, training_label, test_label=train_test_split(datas,labels, tes
t_size=0.2, random_state=seed1)

# input data normalization
data_mean=training_data.mean(axis=0)
data_std=training_data.std(axis=0)

training_data=(training_data-data_mean)/data_std
test_data=(test_data-data_mean)/data_std

# output data normalization
#label_mean=training_label.mean(axis=0)
#label_std=training_label.std(axis=0)

#training_label=(training_label-label_mean)/label_std
#test_label=(test_label-label_mean)/label_std

#-------------------------------
# DNN Model building
#-------------------------------

from keras import models
from keras import layers
from keras import optimizers

def build_model():
    model=models.Sequential()
    model.add(layers.Dense(5,activation='relu',input_shape=(dimY*n0,)))
    model.add(layers.Dense(5,activation='relu'))
    model.add(layers.Dense(5,activation='relu'))
    model.add(layers.Dense(5,activation='relu'))
    model.add(layers.Dense(5,activation='relu'))
    model.add(layers.Dense(dimX))

    model.compile(optimizer=optimizers.SGD(lr=0.001),
                  loss='mean_squared_error',
                  metrics=[tf.keras.metrics.MeanSquaredError()])
    return model

model=build_model()
mymodel=model.fit(training_data,training_label,epochs=10, batch_size=8)
```

```python
#--------------------------------
# Evaluation Performance
#--------------------------------

from sklearn.metrics import mean_squared_error

test_mse_score, test_mae_score=model.evaluate(test_data,test_label)

# Normalization x_bars data to compare with test_label
#x_bars=(x_bars-label_mean)/label_std
# find test_label index in DataFrame
index=test_label.index.tolist()
kf_mse_err=mean_squared_error(x_bars.iloc[index],labels.iloc[index])

cpu_end=time.perf_counter()

print("The mse of deep filtering is {:.3%}".format(test_mse_score))
print("The mse of Kalman Filtering is {:.3%}".format(kf_mse_err))
print("The CPU consuming time is {:.5}".format(cpu_end-cpu_start))

#--------------------------------
# Training loss graph
#--------------------------------

history_dict=mymodel.history
loss_value=history_dict['loss']
#val_loss_value=history_dict['val_loss']
epochs=range(1,10+1)
import matplotlib.pyplot as plt
plt.plot(epochs, loss_value, 'bo',label='Training Loss')
#plt.plot(epochs, val_loss_value,'b',label='Validation Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
95200/95200 [==============================] - 37s 391us/step - loss: 0.0700 - mean_s
quared_error: 0.0700
Epoch 2/10
95200/95200 [==============================] - 37s 384us/step - loss: 0.0315 - mean_s
quared_error: 0.0315
Epoch 3/10
95200/95200 [==============================] - 37s 386us/step - loss: 0.0246 - mean_s
quared_error: 0.0246
Epoch 4/10
95200/95200 [==============================] - 37s 384us/step - loss: 0.0197 - mean_s
quared_error: 0.0197
Epoch 5/10
95200/95200 [==============================] - 36s 381us/step - loss: 0.0177 - mean_s
quared_error: 0.0177
Epoch 6/10
95200/95200 [==============================] - 36s 381us/step - loss: 0.0164 - mean_s
quared_error: 0.0164
Epoch 7/10
95200/95200 [==============================] - 36s 379us/step - loss: 0.0155 - mean_s
quared_error: 0.0155
Epoch 8/10
95200/95200 [==============================] - 36s 383us/step - loss: 0.0149 - mean_s
quared_error: 0.0149
Epoch 9/10
95200/95200 [==============================] - 38s 400us/step - loss: 0.0144 - mean_s
quared_error: 0.0144
Epoch 10/10
95200/95200 [==============================] - 39s 405us/step - loss: 0.0140 - mean_s
quared_error: 0.0140
5950/5950 [==============================] - 2s 348us/step - loss: 0.0139 - mean_squa
red_error: 0.0139
The mse of deep filtering is 1.389%
The mse of Kalman Filtering is 2.496%
The CPU consuming time is 445.51
```



In [3]:

```
#----------------------------------------
# plot on new data
#----------------------------------------
# new number of time step horizon.
N_new=1000
x_new,y_new=mc_simulation(F,G,H,u,v,N_new)
x_hat_new,x_bar_new=kalman_filtering(F,G,H,Q0,R0,x0,y_new,N_new)
y_new=y_new.reshape(N_new+1,dimY)
data_new=np.zeros((N_new-n0+2,n0,dimY))
for k in range(N_new-n0+2):
    data_new[k]=y_new[k:k+n0]
data_new=data_new.reshape(N_new-n0+2,n0*dimY)
data_new=pd.DataFrame(data_new)
# input normalization
data_new=(data_new-data_mean)/data_std
```

In [4]:

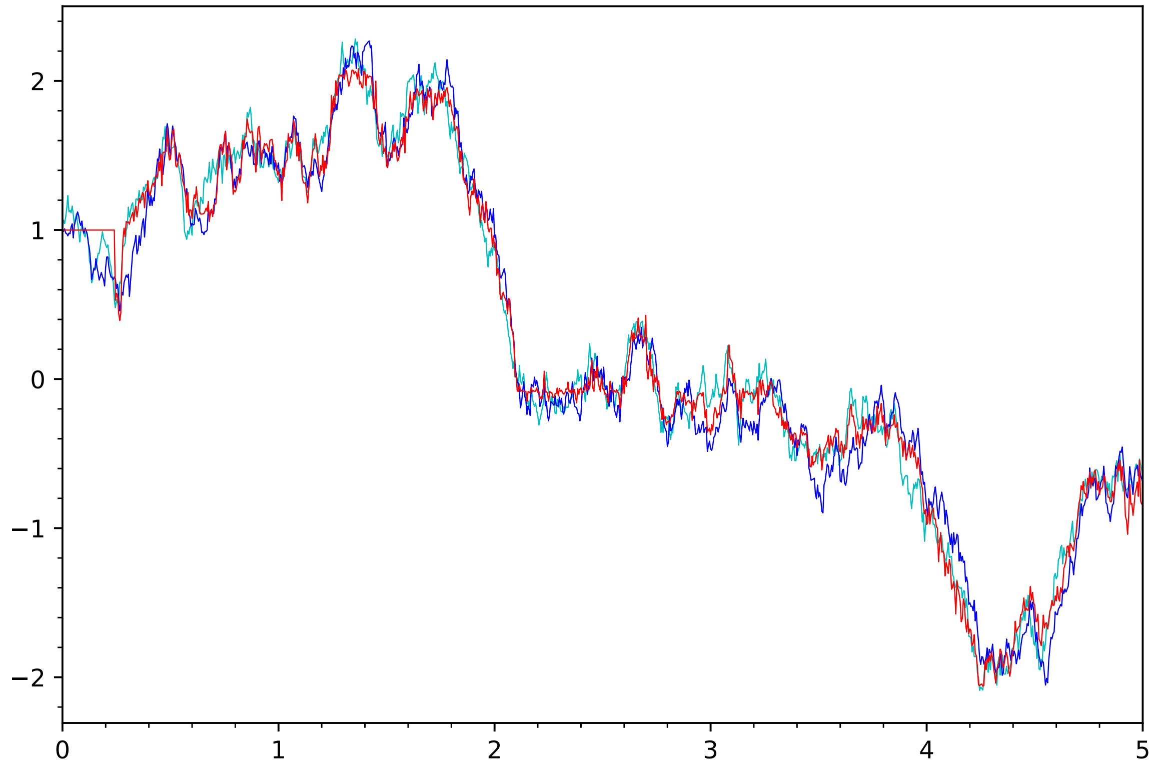
```
# deep filtering prediction value.
df_pred=model.predict(data_new)
# convert prediction value to original scale.
#for i in range(N_new-n0+2):
#    df_pred[i,:]=df_pred[i,:]*label_mean+label_std
```

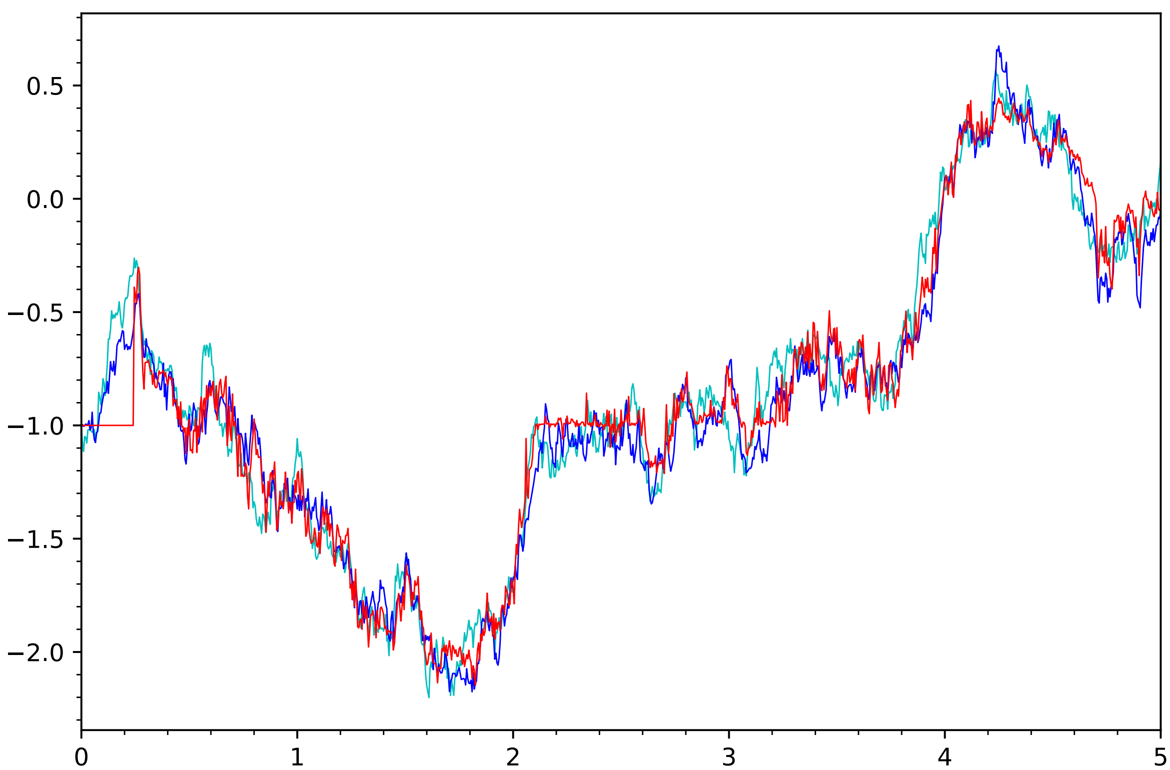
In [5]:

```
# For estimation before state index n0-1, we use x0 to replace it.
df_new=[x0 for k in range(n0-1)]
df_new=np.array(df_new)
df_new=df_new.reshape(n0-1,dimX)
df_new=np.vstack((df_new,df_pred))
```

In [9]:

```
axis=np.linspace(0,5,N_new+1)
fig,ax=plt.subplots(2,1,figsize=[8,12])
ax[0].plot(axis, x_new[:,0],'c',axis,x_bar_new[:,0],'b',axis,df_new[:,0],'r',linewidth=
0.5)
ax[0].minorticks_on()
ax[0].set_xlim((0,5))
ax[1].plot(axis, x_new[:,1],'c',axis, x_bar_new[:,1],'b',axis,df_new[:,1],'r',linewidth
=0.6)
ax[1].set_xlim((0,5))
ax[1].minorticks_on()
plt.show()
```





In [ ]: