Problem 1

Show that the innovations $v_t = Y_t - E[X_t \mid Y_{1:t-1}]$ in the local level model are mutually independent.

Proof

Let's first find the joint distributions of the observations Y_i.

- $p(Y_2, Y_1) = p(Y_2 \mid Y_1) \cdot p(Y_1)$
- $p(Y_3, Y_2, Y_1) = p(Y_3 \mid Y_{1:2}) \cdot p(Y_1, Y_2)$
  $= p(Y_3 \mid Y_{1:2}) \cdot p(Y_2 \mid Y_1) \cdot p(Y_1)$
- $p(Y_4, Y_3, Y_2, Y_1) = p(Y_4 \mid Y_{1:3}) \cdot p(Y_1, Y_2, Y_3)$
  $= p(Y_4 \mid Y_{1:3}) \cdot p(Y_3 \mid Y_{1:2}) \cdot p(Y_2 \mid Y_1) \cdot p(Y_1)$

Based on the derivations above, we say that the joint density function of the first n observations is:

$$p_{Y_1, Y_2, \dots, Y_n}(y_1, y_2, \dots, y_n) = p_{Y_1}(y_1) \cdot \prod_{t=2}^{n} p_{Y_{1:t-1}}(y_t \mid y_1, y_2, \dots, y_t) \qquad (1)$$

(this can be proved easily by induction)

We recall that:

$$v_t = y_t - \mu_t = y_t - E[X_t \mid Y_{1:t-1}] \equiv \text{a linear function of of } y_1, \dots, y_{t-1} \qquad (2)$$

Now, our goal is to transform (1) into the joint density function of $v_1, v_2, \dots, v_n$. To do this, we will apply Jacobian for multivariate probability density function. It says:

$$f_{V_1, V_2, \dots, V_n}(v_1, v_2, \dots, v_n) = \frac{f_{Y_1, Y_2, \dots, Y_n}(y_1, y_2, \dots, y_n)}{\left| \dfrac{\partial(v_1, v_2, \dots, v_n)}{\partial(y_1, y_2, \dots, y_n)} \right|} \qquad (3)$$

where

$$\frac{\partial(v_1, v_2, \dots, v_n)}{\partial(y_1, y_2, \dots, y_n)} = \det \begin{pmatrix} \dfrac{\partial y_1}{\partial v_1} & \cdots & \dfrac{\partial y_1}{\partial v_n} \\ \vdots & & \\ \dfrac{\partial y_n}{\partial v_1} & & \dfrac{\partial y_n}{\partial v_n} \end{pmatrix}$$

Because of (2), we conclude that the matrix is the identity and therefore

$$\left| \frac{\partial(v_1, v_2, \dots, v_n)}{\partial(y_1, y_2, \dots, y_n)} \right| = 1.$$

Expanding (1), we get:

$$p_{Y_1, Y_2, \dots, Y_n}(y_1, y_2, \dots, y_n) = p_{Y_1}(y_1) \cdot p(Y_2 \mid Y_1) \cdot p(Y_3 \mid Y_1, Y_2) \cdot p(Y_4 \mid Y_{1:3}) \cdot \ldots \cdot p(Y_n \mid Y_{1:n-1}) \qquad (4)$$

Since the Jacobian $= 1$ and $p_{V_t}(v_t) = p_{Y_t \mid Y_{1:t-1}}(y_t \mid y_{1:t-1})$, (4) becomes:

$$p_{V_1, V_2, \dots, V_n}(v_1, v_2, \dots, v_n) = p(v_1) \cdot p(v_2) \cdot \ldots \cdot p(v_n)$$
$$= \prod_{t=1}^{n} p(v_t)$$

Since the joint density factors, we conclude that $v_1, v_2, \dots, v_n$ are mutually independent.

MTH9893 Time Series Analysis HW5

Group 13: Yawei Wang, Gongshun Yin

Problem 1

From the lecture note, $v_1, v_2, \ldots v_t$ are joint normal with expection equal zero.

So to prove v_t are mutually independent, it just need to prove v_s, v_t are uncorrelated for any $s

The prove using "Tower Property":

If $H \subset G$, then $E[E[Y|H]|G] = E[Y|H] = E[E[Y|G]|H]$

$$
\begin{aligned}
Cov(v_t, v_s) &= E[v_t v_s] \\
&= E[E[v_t v_s | Y_{1:s}]] \\
&= E[E[v_t | Y_{1:s}] v_s] \\
&= E[E[Y_t - E[X_t | Y_{1:t-1}] | Y_{1:s}] v_s] \\
&= E[(E[Y_t | Y_{1:s}] - E[E[X_t | Y_{1:t-1}] | Y_{1:s}]) v_s] \\
&= E[(E[X_t + \eta_t | Y_{1:s}] - E[X_t | Y_{1:s}]) v_s] \\
&= E[(E[X_t | Y_{1:s}] - E[X_t | Y_{1:s}]) v_s] \\
&= 0
\end{aligned}
$$

MTH9893_HW5_Group14

- Weihao Li
- Wenli Dong

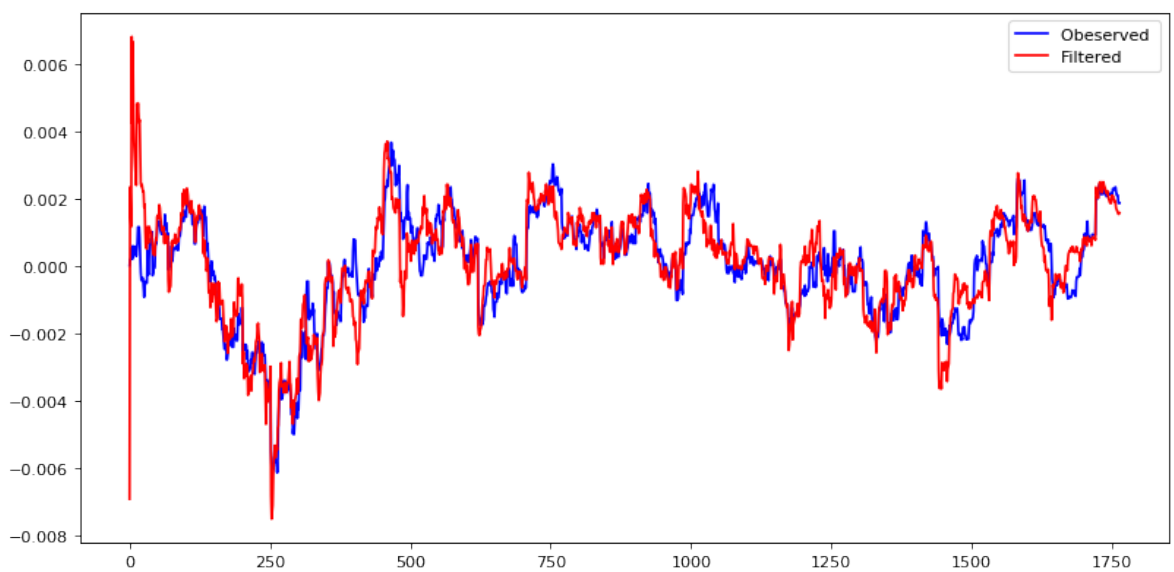
In [73]:
```
#import nessesary package
import math
import numpy as np
import pandas as pd
import statsmodels.api as sm
from pykalman import KalmanFilter
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
```

In [59]:
```
#parsing and reading in data
# data parsing
SPX = pd.read_excel("HW5.xlsx", skiprows=1,  parse_cols='A:B')
APPL= pd.read_excel("HW5.xlsx", skiprows=1,  parse_cols='D:E')
data = SPX.merge(APPL,how = 'inner',on = 'Date')

# Change price to daily return
APPL = ((data.PX_LAST_y- data.PX_LAST_y.shift())/ data.PX_LAST_y)[1:]
SPX =  ((data.PX_LAST_x- data.PX_LAST_x.shift())/ data.PX_LAST_x)[1:]
```

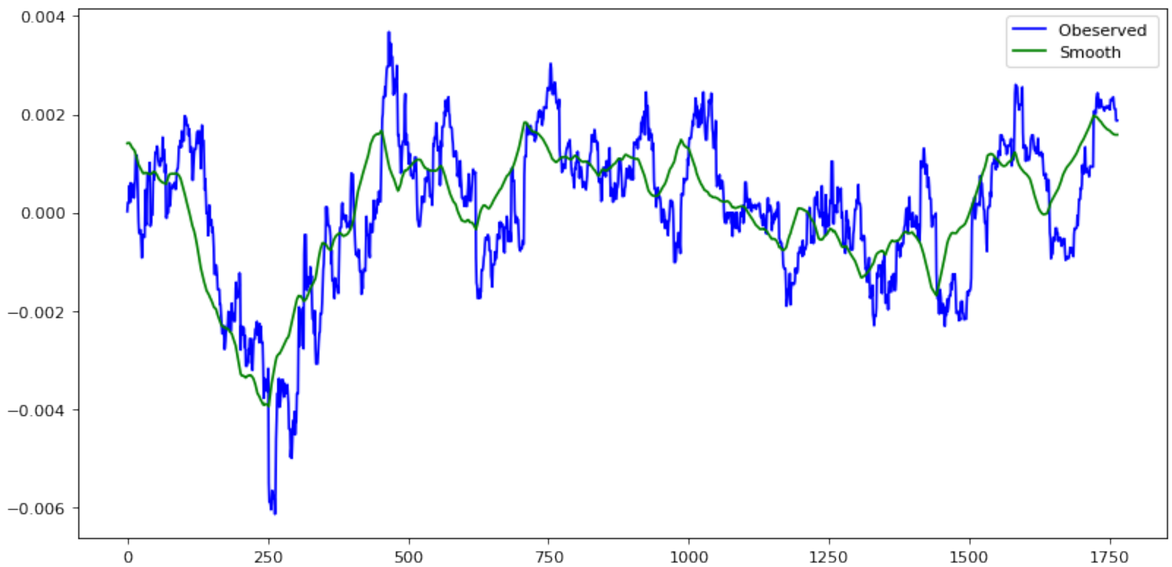
In [60]:
```
#get the legth of the return data
length = len(data.index)-1

# add a constant term for the ols regreesion
SPX_sm = sm.add_constant(SPX)
#Xs to store alphas, betas to store betas
Xs = []
betas = []

# OLS for window = 63
for i in xrange(63,length +1):
    res = sm.OLS(APPL[i-63:i], SPX_sm[i-63:i]).fit()
    Xs.append(res.params[0])
    betas.append(res.params[1])

# Calculate Ys by definition
Ys = np.array(APPL[62:length+1]) -  np.array(betas)*np.array(SPX[62:le
ngth+1])
```

In [61]:
```
# applying Kalman Filter
#set initia state as diffuse prior, set convariance by em algorithm
kf = KalmanFilter(transition_matrices=[1],
                  observation_matrices=[1],
                  transition_offsets=[0],
                  observation_offsets=[0],
                  initial_state_mean=0,
                  initial_state_covariance=[2**32],
)
kf = kf.em(Ys,n_iter = 300,em_vars=['transition_covariance','observati
on_covariance'])
# store the filered values and smooth values
(filtered_state_means, filtered_state_covariances) = kf.filter(Ys)
(smoothed_state_means, smoothed_state_covariances) = kf.smooth(Ys)
```

In [62]:
```
# plot the filtered values with observed value
plt.figure(figsize=(12, 6), dpi=80)
plt.plot(Xs,linestyle='-',color='blue')
plt.plot(filtered_state_means,linestyle='-',color='red')
plt.legend(['Obeserved ','Filtered'])
plt.show()
```

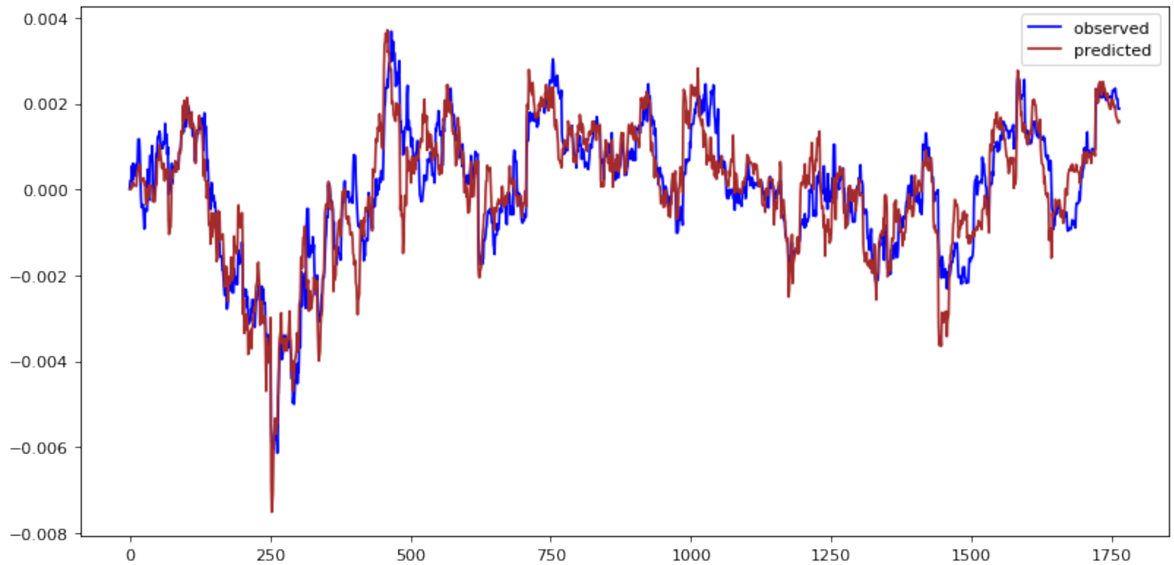


In [63]:
```
# plot the smoothing values with observed value
plt.figure(figsize=(12, 6), dpi=80)
plt.plot(Xs,linestyle='-',color='blue')
plt.plot(smoothed_state_means,linestyle='-',color='green')
plt.legend(['Obeserved ','Smooth'])
plt.show()
```



In [83]:
```
# Now we try to say something about its predict power
#initalized two list with the length as observations
(predicted_state_means, predicted_state_covariances) = kf.filter(Ys)
predicted_state_means[0] = 0
predicted_state_covariances[0] = 0

# use observation at t and mean/variance estimates at t-1 to predict m
ean at time t
for i in range(1,len(Ys)):
    (predicted_state_means[i], predicted_state_covariances[i]) =  kf.f
ilter_update(predicted_state_means[i-1], predicted_state_covariances[i
-1], Ys[i])
```

In [65]:
```
# plot the smoothing values with observed value
plt.figure(figsize=(12, 6), dpi=80)
plt.plot(Xs,linestyle='-',color='blue')
plt.plot(predicted_state_means,linestyle='-',color='brown')
plt.legend(['observed ','predicted'])
plt.show()
```



In [84]:
```
mse = mean_squared_error(Xs,predicted_state_means)
print "mse over varriance of observed value: ",mse/np.var(Xs)
print "\nFrom the picture above, we can see that the predicted value d
eviate a lot from the observed value sometime,\
but generally speaking, it does show the direction and trend of the ob
served value."
print "\nFurthur more, by comparing the mean square error and the vari
ance of the observed alpha\
we find that the mse is small, which indicated a good prediction."
```

```
mse over varriance of observed value:  0.198049182801

From the picture above, we can see that the predicted value deviate
a lot from the observed value sometime,but generally speaking, it do
es show the direction and trend of the observed value.

Furthur more, by comparing the mean square error and the variance of
the observed alphawe find that the mse is small, which indicated a g
ood prediction.
```