Pair Trading Strategy in Korean ETFs

Yeonsik Jang

EPAT batch 40

Project Abstract

I applied pair trading strategy based on cointegration relationship between Korean ETFs. The optimal hedge ratio and entry/exit condition was dynamically given from using Kalman filter.

Introduction

Cointegrated relationship between a pair means that we can find stationary linear combinations of pairs. Once the pair is cointegrated, it is reasonable to apply mean-reversion strategy since the cointegration pair itself would possesses mean-reverting property with mean of zero and standard deviation. Here, I have collected and performed cointegration test for Korean ETFs' historical data from January 2014 to March 2019, then constructed cointegrated pair. One important quantity to be determined is its 'hedge ratio', a ratio of long/short positions between pairs. The hedge ratio can be determined by ordinary least square (OLS) method, eigenvector method through Johansen test, or dynamically calculated using Kalman filter. Among, Kalman filtering has significant advantage in that it can calculate optimal hedge ratios dynamically and parameter-free. This fact is crucial because with using Kalman filter, we don't have to determine hyperparemeters accompanied by human's subjective intervention. Furthermore, we can also use dynamically updated measurement standard deviation as entry/exit point of the trading, so we don't have to set an artificial trading signal criteria. After adopting Kalman filter to determine hedge ratio and occurring trading signals, I performed a backtest for pair trading in Korean ETFs and analyzed their performance.

Data Download

The full-ticker-list of Korean ETFs are downloaded from Korea Exchange (KRX). Being used this ticker list, the historical ETF data was downloaded through wed-crawling via http://finance.naver.com/

```
for num in range(0,len(etf_list)):
    url = 'http://finance.naver.com/item/sise_day.nhn?code={code}'.format(code=etf_list['ticker'][num])
    df = pd.DataFrame()
    for page in range(1, 128):
        pg_url = '{url}&page={page}'.format(url=url, page=page)
        df = df.append(pd.read_html(pg_url, header=0)[0], ignore_index=True)
    df = df[["날짜",'종가']]
    df = df.rename(columns= {'날짜': 'Date', '종가': etf_list['name'][num]})
    df = df.dropna()
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.sort_values(by=['Date'], ascending=True)
    df.set_index('Date', inplace=True)
    df.to_csv(etf_list['name'][num]+".csv")
    print("{}. {} 저장 완료".format(num+1, etf_list['name'][num]))

df = pd.read_csv(etf_list['name'][0]+".csv")
for i in range(1,len(etf_list)):
    frame = pd.read_csv(etf_list['name'][i]+".csv")
    df = pd.merge(df,frame)
df.to_csv('ETFs_cl')
```

Cointegration Test

The cointegrated augmented Dickey-Fuller test (CADF test) was performed to find cointegrated Korean ETF pairs. In order to avoid data snoop bias, the dataset was divided as train set (60 %) and test set (40 %). The CADF test was performed using only trainset data. The critical p value was set as 0.01 (1 %).

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as sm
import statsmodels.tsa.stattools as ts
import statsmodels.tsa.vector_ar.vecm as vm

def get_cointegrated_pairs(dataframe, critical_level = 0.01, train_test_ratio = 0.6):

#     dataframe.set_index('Date', inplace=True)

    # train set / test set split
    train_df = dataframe[:int(len(dataframe.index)*train_test_ratio)]
    test_df = dataframe[int(len(dataframe.index)*train_test_ratio):]

    # Create matrix
    n = train_df.shape[1] # number of etfs
    pvalue_matrix = np.ones((n,n))
    keys = train_df.keys()
    pairs = []

    for i in range(n):
        for j in range(i+1, n):
            pair_1 = train_df[keys[i]]
            pair_2 = train_df[keys[j]] # obtain the price of two contract

            result = ts.coint(pair_1, pair_2) # get conintegration
            pvalue = result[1] # get the pvalue
            pvalue_matrix[i, j] = pvalue

            if pvalue < critical_level: # if p-value less than the critical level
                pairs.append((keys[i], keys[j], pvalue)) # record the contract with that p-value

        if i%10==0:
            print("{}th calculation completed".format(i))

    return pvalue_matrix, pairs
```

As a result of CADF test, a set of cointegration pairs are shown below.

	pair1	pair2	pvalue
0	KODEX 자동차	KODEX 일본TOPIX100	0.005357
1	TIGER 방송통신	KODEX 콩선물(H)	0.003286
2	KODEX 삼성그룹	KODEX 구리선물(H)	0.007496
3	KODEX 삼성그룹	TIGER 헬스케어	0.008940
4	KOSEF 블루칩	TIGER 농산물선물Enhanced(H)	0.006785
5	KOSEF 고배당	TIGER 미국S&P500선물(H)	0.004069
6	KINDEX 삼성그룹섹터가중	KODEX 구리선물(H)	0.004509
7	KODEX 골드선물(H)	TIGER 금은선물(H)	0.002441
8	TIGER 미국나스닥100	TIGER 200 경기소비재	0.000737
9	KOSEF 미국달러선물	KBSTAR 수출주	0.008353
10	TIGER 삼성그룹펀더멘털	KODEX 구리선물(H)	0.006318
11	TIGER 삼성그룹펀더멘털	KBSTAR 우량업종	0.004510
12	KODEX 구리선물(H)	KBSTAR 우량업종	0.002264
13	TIGER 금속선물(H)	KBSTAR 우량업종	0.001762
14	KBSTAR 우량업종	TIGER 구리실물	0.002605
15	KODEX 보험	KODEX 은선물(H)	0.005944
16	파워 코스피100	KODEX MSCI Korea	0.000623

There are 17 cointegrated-ETF pairs.

Kalman Filter as Dynamic Regression

Kalman filter is an optimal linear algorithm that updates the expected value of a hidden variable based on the latest value of an observable variable. The ‘optimal’ means that Kalman filter could be the best estimator available that minimizes the mean square error if the noises are given in Gaussian. Kalman filtering has two variables: observable variable, hidden variable, and two transition matrix: state transition matrix for hidden variable and observation model for observable variable. The state is linear model and updates iteratively, where the expected value of hidden variable at time t is just its observation at t-1. For pairs trading, hidden variable will be the hedge ratio ‘beta’, and observable variables ‘x’, and ‘y’ are prices of pairs. The python iteration code for Kalman filtering is given below.

```
for i in range(len(df_pairs)):
    pair_1 = df_pairs.iloc[i][1]
    pair_2 = df_pairs.iloc[i][0]
    pair_data = etfs[[pair_1,pair_2]]

    # Fist 2 datapoints should be removed because of Kalman filter iteration result
    df = etfs[[pair_1,pair_2]].iloc[2:]
    ret = df.pct_change()
    x = etfs[pair_1]
    y = etfs[pair_2]


    x=np.array(ts.add_constant(x))[:,[1,0]]

    delta=0.0001

    yhat=np.full(y.shape[0], np.nan)
    e=yhat.copy()
    Q=yhat.copy()

    # For clarity, we denote R(t|t) by P(t). Initialize R, P and beta.
    R=np.zeros((2,2))
    P=R.copy()
    beta=np.full((2, x.shape[0]), np.nan)
    Vw=delta/(1-delta)*np.eye(2)
    Ve=0.001
    beta[:, 0]=0

    for t in range(len(y)):
        if t > 0:
            beta[:, t]=beta[:, t-1]
            R=P+Vw

        yhat[t]=np.dot(x[t, :], beta[:, t])
    #     print('FIRST: yhat[t]=', yhat[t])

        Q[t]=np.dot(np.dot(x[t, :], R), x[t, :].T)+Ve
    #     print('Q[t]=', Q[t])

        # Observe y(t)
        e[t]=y.values[t]-yhat[t] # measurement prediction error
    #     print('e[t]=', e[t])
    #     print('SECOND: yhat[t]=', yhat[t])


        K=np.dot(R, x[t, :].T)/Q[t] #  Kalman gain
    #     print(K)

        beta[:, t]=beta[:, t]+np.dot(K, e[t]) #  State update
    #     print(beta[:, t])

        P=R-np.dot(np.dot(K, x[t, :]), R) # State covariance update
```

Trading Strategy Identification

As a result of Kalman filtering, we can compute measurement error e(t), which equals to spread. We can buy this spread e(t) when the e(t) is below its standard deviation and sell vice versa. The standard deviation of the spread e(t) can be easily calculated by iteratively updating measurement variance Q(t) (see the above code). Hence, the trading strategy is identified as below python code.

```
pair_data['yhat'] = yhat
pair_data['beta'] = beta[0,:]
pair_data['spread'] = e
pair_data['std'] = np.sqrt(Q)

# Fist 2 datapoints should be removed because of Kalman filter iteration result
pair_data = pair_data[pair_data.yhat !=0]

# Entry/Exit condition
pair_data['longsEntry'] = pair_data['spread']< -pair_data['std']
pair_data['longsExit'] = pair_data['spread'] > 0
pair_data['shortsEntry'] = pair_data['spread']> pair_data['std']
pair_data['shortsExit'] = pair_data['spread'] < 0


# Identifying position
pair_data['positionsLong']  = np.nan
pair_data['positionsShort']  = np.nan

pair_data['positionsLong'][0]  = 0
pair_data['positionsShort'][0]  = 0

pair_data.loc[pair_data.longsEntry, 'positionsLong'] = 1
pair_data.loc[pair_data.longsExit, 'positionsLong'] = 0
pair_data.positionsLong = pair_data.positionsLong.fillna(method='ffill')

pair_data.loc[pair_data.shortsEntry, 'positionsShort'] = -1
pair_data.loc[pair_data.shortsExit, 'positionsShort'] = 0
pair_data.positionsShort = pair_data.positionsShort.fillna(method='ffill')

pair_data['positions'] = pair_data.positionsLong + pair_data.positionsShort


pair_data['positions_size_x'] = pair_data.positions * (-pair_data.beta) * pair_data[df_pairs.iloc[i][1]]
pair_data['positions_size_y'] = pair_data.positions * pair_data[df_pairs.iloc[i][0]]
```

Backtesting

Once the entry and exit signals are determined as above, the backtest result can be obtained as below python code, including strategy returns, standard deviations of returns, pnl, APR, hit_ratio, annualized sharpe ratio, maximum drawdown, etc.

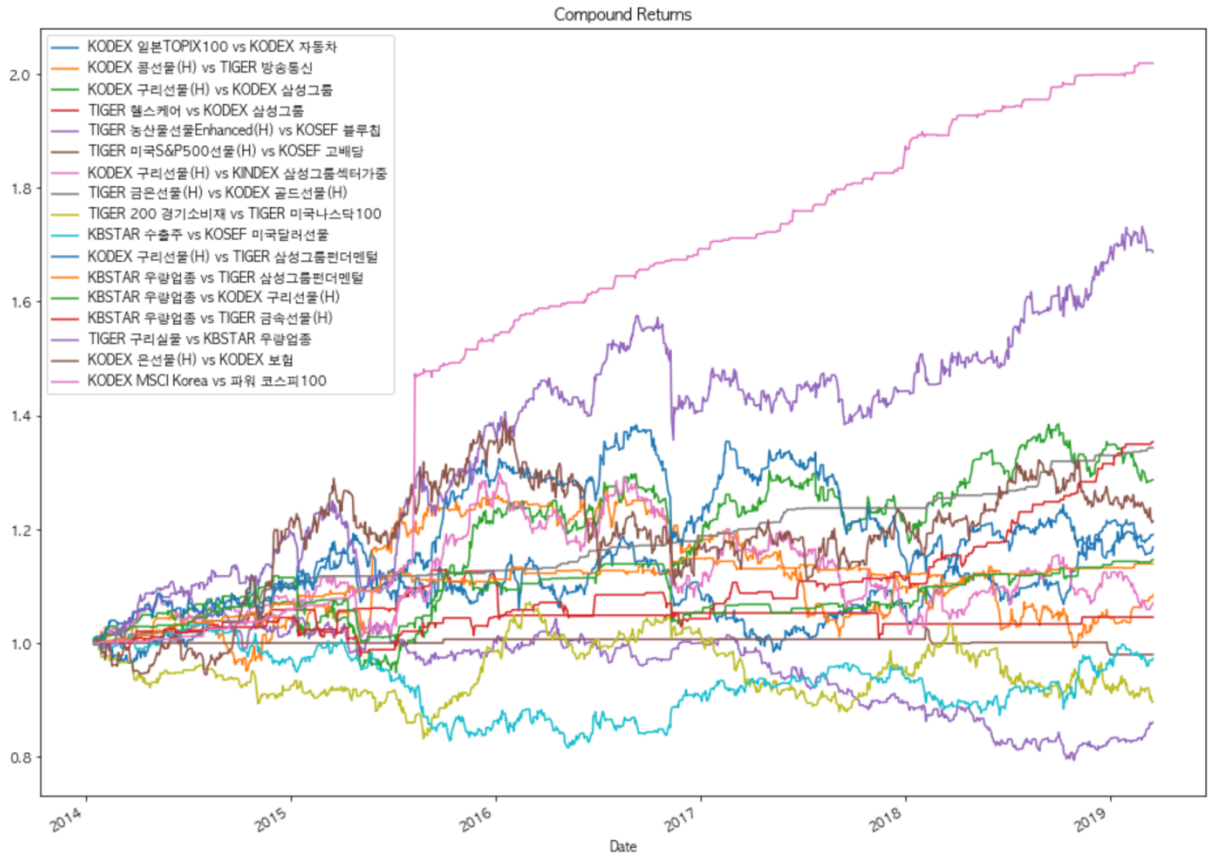
```
    # Backtesting
    pair_data['pnl'] = pair_data.positions_size_x.shift(1) * ret.values[:,0] + pair_data.positions_size_y.shift(1) * r
et.values[:,1]
    pair_data['returns'] = pair_data.pnl/(abs(pair_data.positions_size_x.shift())+abs(pair_data.positions_size_y.shift
()))

    pair_data.pnl = pair_data.pnl.fillna(0)
    pair_data.returns = pair_data.returns.fillna(0)

    pair_data['cumpnl'] = pair_data.pnl.cumsum()
    pair_data['cumret'] = (1+pair_data.returns).cumprod()


    # Analysing Backtest Output
    APR = ((pair_data.cumret[-1])**(252/len(pair_data.cumret))-1)
    hit_ratio = len(pair_data.pnl[pair_data.pnl>0])/len(pair_data.pnl[pair_data.pnl!=0])
    std_returns = pair_data.returns.std()
    sharpe_ratio = np.sqrt(252)*pair_data.returns.mean()/pair_data.returns.std()
    Daily_DD = (pair_data['cumret']/pair_data['cumret'].rolling(252,1).max() -1)
    Max_Daily_DD = Daily_DD.rolling(252,1).min()
    MDD = abs(min(Max_Daily_DD))

    # Updating summary tables
    df_summary.iloc[i] = [APR*100, hit_ratio, std_returns, sharpe_ratio, MDD*100]
    df_summary_cumret[pair_list[i]] = pair_data.cumret
    df_summary_drawdown[pair_list[i]] = Max_Daily_DD
```

Backtesting Results and Key Findings

Summary of the backtest result is given in the below table.

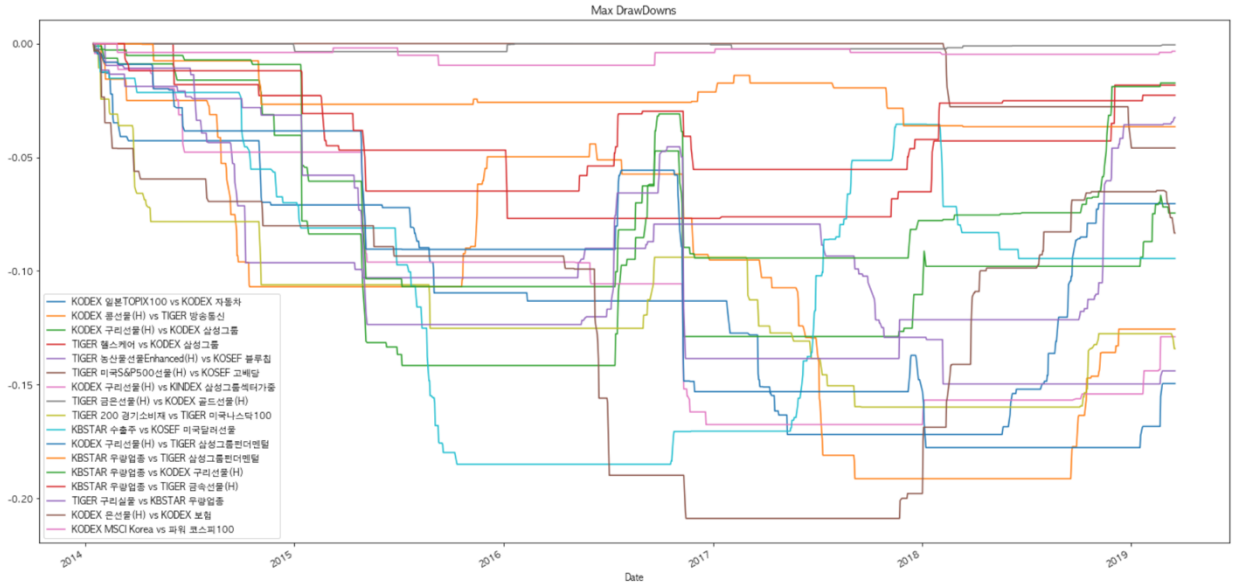
	APR (%)	hit_ratio	std_returns	sharpe_ratio	MDD (%)
KODEX 일본TOPIX100 vs KODEX 자동차	3.52975	0.520376	0.00643431	0.390736	17.2017
KODEX 콩선물(H) vs TIGER 방송통신	1.62438	0.502128	0.00644191	0.208194	19.1469
KODEX 구리선물(H) vs KODEX 삼성그룹	5.14116	0.536965	0.0057011	0.599306	14.1644
TIGER 헬스케어 vs KODEX 삼성그룹	0.888915	0.597222	0.00264539	0.231713	7.69462
TIGER 농산물선물Enhanced(H) vs KOSEF 블루칩	-2.94341	0.498667	0.0043717	-0.395766	14.9792
TIGER 미국S&P500선물(H) vs KOSEF 고배당	-0.40454	0.384615	0.000842464	-0.296437	4.59785
KODEX 구리선물(H) vs KINDEX 삼성그룹섹터가중	1.35729	0.519786	0.00613992	0.187042	16.7678
TIGER 금은선물(H) vs KODEX 골드선물(H)	6.03873	0.870968	0.000991138	3.7349	0.35418
TIGER 200 경기소비재 vs TIGER 미국나스닥100	-2.151	0.492958	0.00588318	-0.186053	15.9964
KBSTAR 수출주 vs KOSEF 미국달러선물	-0.551511	0.498807	0.00530033	-0.0237338	18.5127
KODEX 구리선물(H) vs TIGER 삼성그룹펀더멘털	3.14807	0.524554	0.00583647	0.380866	17.7782
KBSTAR 우량업종 vs TIGER 삼성그룹펀더멘털	2.62719	0.52439	0.00205174	0.812492	3.66347
KBSTAR 우량업종 vs KODEX 구리선물(H)	2.75432	0.573585	0.00334616	0.538229	10.695
KBSTAR 우량업종 vs TIGER 금속선물(H)	6.20383	0.566372	0.0030359	1.27308	6.48977
TIGER 구리실물 vs KBSTAR 우량업종	10.9543	0.553169	0.00622308	1.10187	13.8594
KODEX 은선물(H) vs KODEX 보험	3.94018	0.527909	0.00803488	0.366753	20.8967
KODEX MSCI Korea vs 파워 코스피100	14.9883	0.775735	0.00547352	1.64781	0.955258

The compound returns of the strategy for each pair is depicted as the graph below.



The compound returns vary from -3.94 % to 14.99 %

The maximum drawdowns within 1-year window is depicted as the graph below.



Chanllenges and Limitations

There are many points to improve this research more.

1. Need to consider transaction cost. Since this model dynamically updating hedge ratio, theorectically it is possible that infinite transaction cost for rebalancing can ocuur.

2. In the same vein, we must apply realistic (discrete) hedge ratios considering the tradeoff between rebalancing cost and optimizing hedge ratio.

3. The backtest was done only for individual pairs. Further research is needed on building portfolios that allocating assets for minimizing risk and maximizing utility.

Conclusion

I have applied pair trading strategy based on cointegration relationship between Korean ETFs. The optimal hedge ratio and entry/exit condition was dynamically given from using Kalman filter. As the backtest result shows, arbitrage opportunities for those pairs are not readily realized, but some pairs have a bit attractive APR and sharpe ratio.

Bibilography

Earnest P. Chan, Algorithmic Trading, Wiley, 2013.