Pairs Trading in MCX Commodities

Project Report By: Himanshu Babbar, EPAT Batch-31

Introduction

Pairs trading is a very popular strategy when trading equities. The idea is to trade a pair of co-integrated stocks, buying one and simultaneously selling the other. I wondered if the same concept applied to commodities, and thus, chose to work on it as a part of my final project.

Pairs Selected

- Gold and Silver
- Gold and Copper
- Aluminium and Nickel
- Lead and Copper

Pair Selection Criteria

- I imported continuous contracts from quandl. The prices in the continuous contracts imported were not adjusted and the time-period chosen for in-sample testing was 01-Jan-2014 to 01-Jan-2016. The commodities for which the data was imported are:

    - Gold (GC)
    - Silver (SI)
    - Aluminium (AL)
    - Lead (PB)
    - Nickel (NI)
    - Zinc (ZN)
    - Copper (CU)
    - Crude Oil (CL)
    - Natural Gas (NG)
    - Cardamom (CRDM)
    - Cotton (CT)

- I then calculated rolling spreads (using regression) with a window size of 360 days on each day for each possible pair of the commodities selected above.

- Using the rolling spreads calculated above, I performed ADF test for each day and filtered the days on which the ADF test confirmed stationarity of the spread on that day with a 95% confidence interval.

- Then, I checked co-integration on the days on which the ADF test confirmed stationarity and selected the pairs that co-integrated with a 95% confidence interval on more that 40% of the days on which ADF test confirmed stationarity.
  For Example, in case of Gold and Silver out of 519 days, ADF test confirmed stationarity on 112 days and out of those 112 days, the pair was found to be co-integrated on 95 days, that is, on 84.82% of the days.

- Next, I checked the correlation for the pairs selected by following the above steps, and out of those I selected only those pairs which had a positive correlation of atleast 60%. The pairs finally selected

are:

Commodity Pairs	Percentage Days	Correlation Coefficient
Gold and Silver	84.82	0.904516712818
Gold and Copper	71.57	0.735004225111
Aluminium and Nickel	82.35	0.664762423283
Lead and Copper	97.34	0.883771810352

Methodology

- Once the pairs were selected, I calculated the spread between the pair components using the log of their price ratio.

- Then, I calculated simple moving average (*ma*) and moving standard deviation (*std*) of the spread, with a window size of 30 days. These values were then used to calculate Z-Scores using the following formula:

    - $Z = (x-ma) / std$

- Once the Z-Scores are calculated, we can use these scores to generate trade signals. There can be following two strategies for entering or exiting a trade:

    1. Price based entry and exit, that is, we enter and exit our positions only on the basis of Z-Scores, that is:
        - enter trade, by going long on the pair, when Z-Score < -2.0
        - enter trade, by going short on the pair, when Z-Score > 2.0
        - exit when Z-Score reverts to 0.0

    2. Follow a strict rule while both entry and exit, that is, in addition to the above conditions for the Z-Scores, I also check whether or not pair is co-integrated on the day of entry or exit.

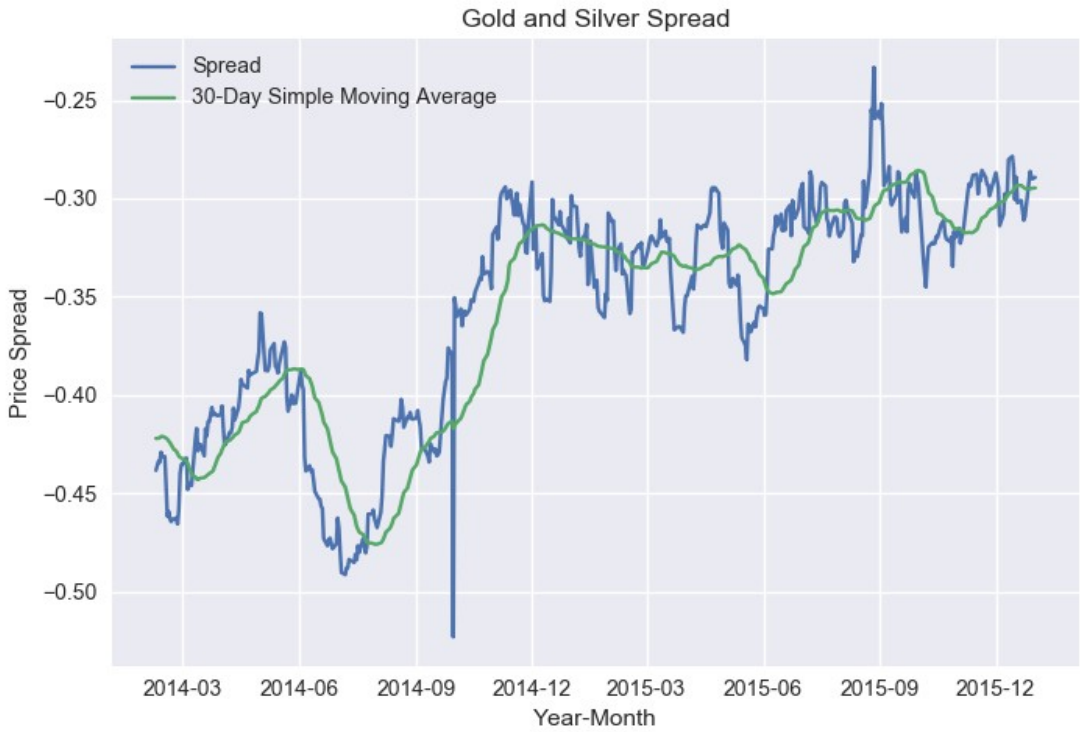
- For both the strategies, it is maintained that once we enter in the trade, we do not enter the trade again unless we exit. That is, only one position is maintained at any given time.

Following sections present the backtesting results for in-sample back-test as well as out of sample back-test for both the strategies.

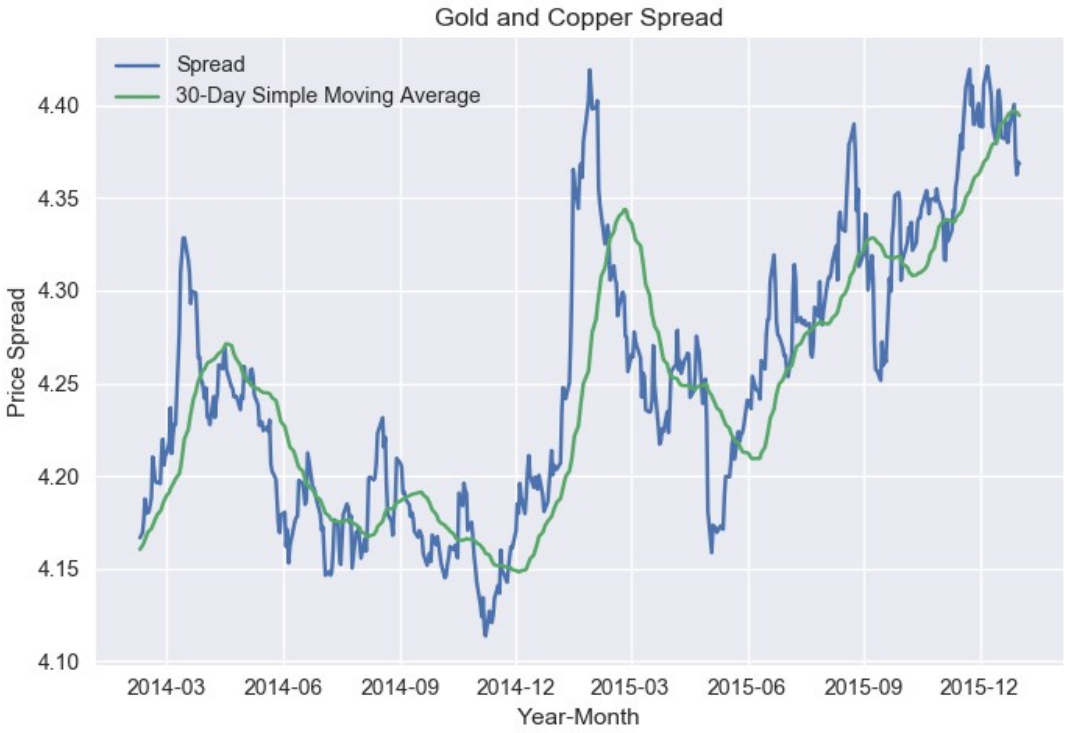
In-Sample Backtest

For in-sample backtesting I have used continuous contracts from 01-Jan-2014 to 01-Jan-2016. The spreads and the Z-Score plots for each pair are as shown below.

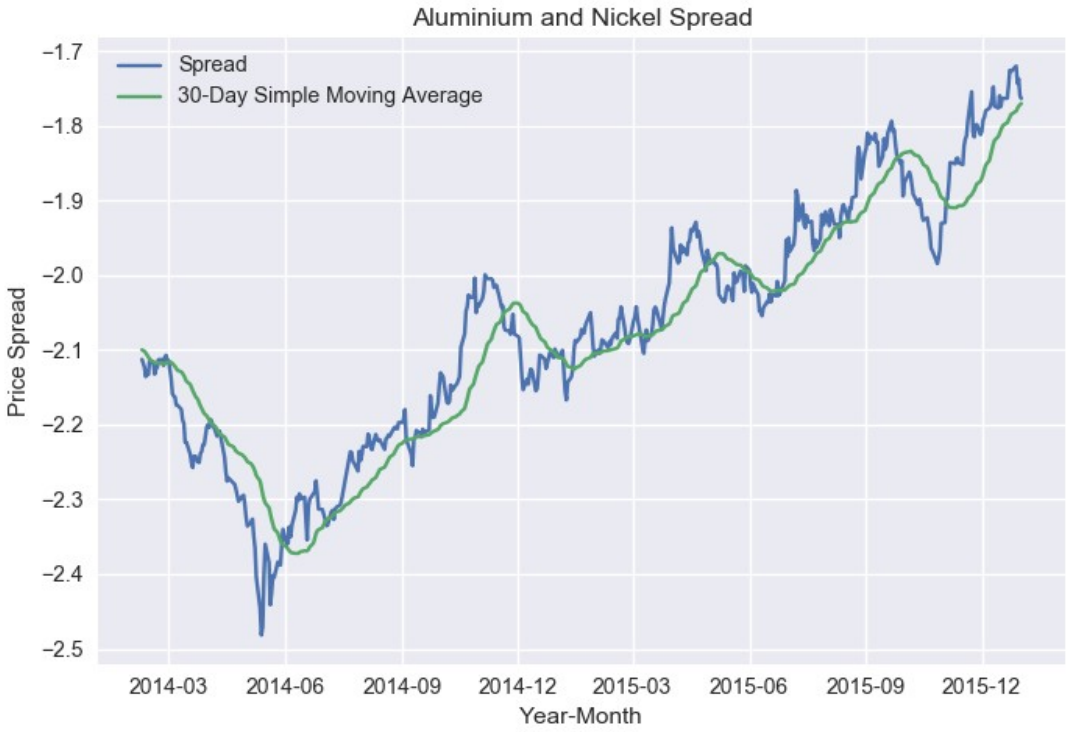
- Gold and Silver

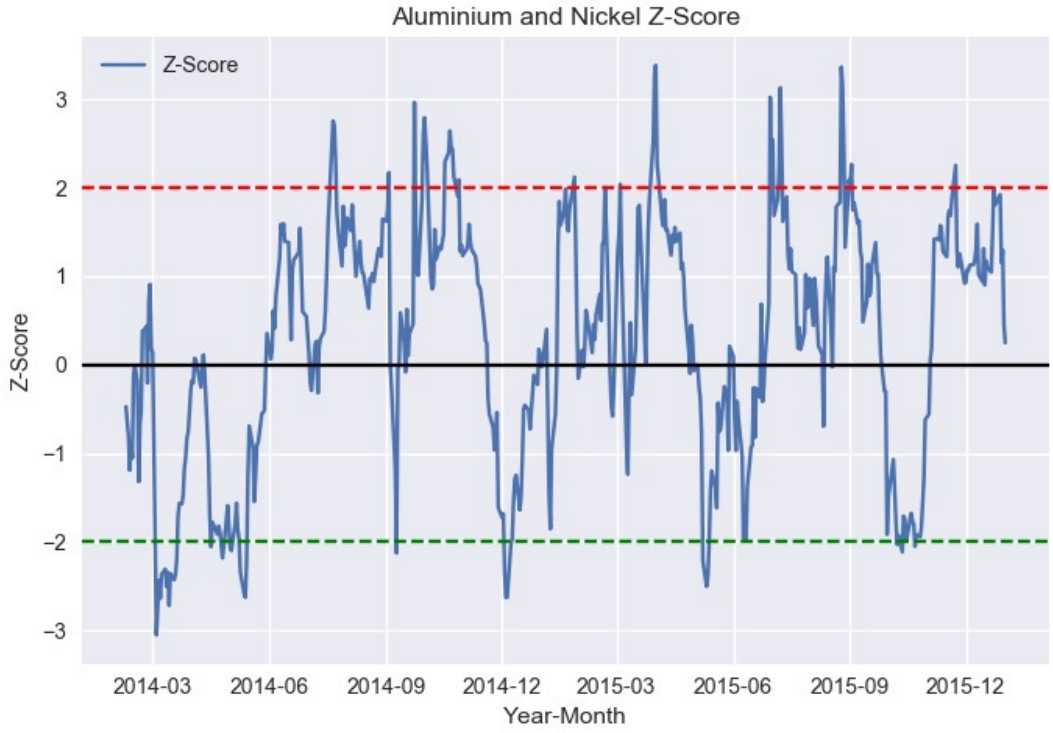


- Gold and Copper

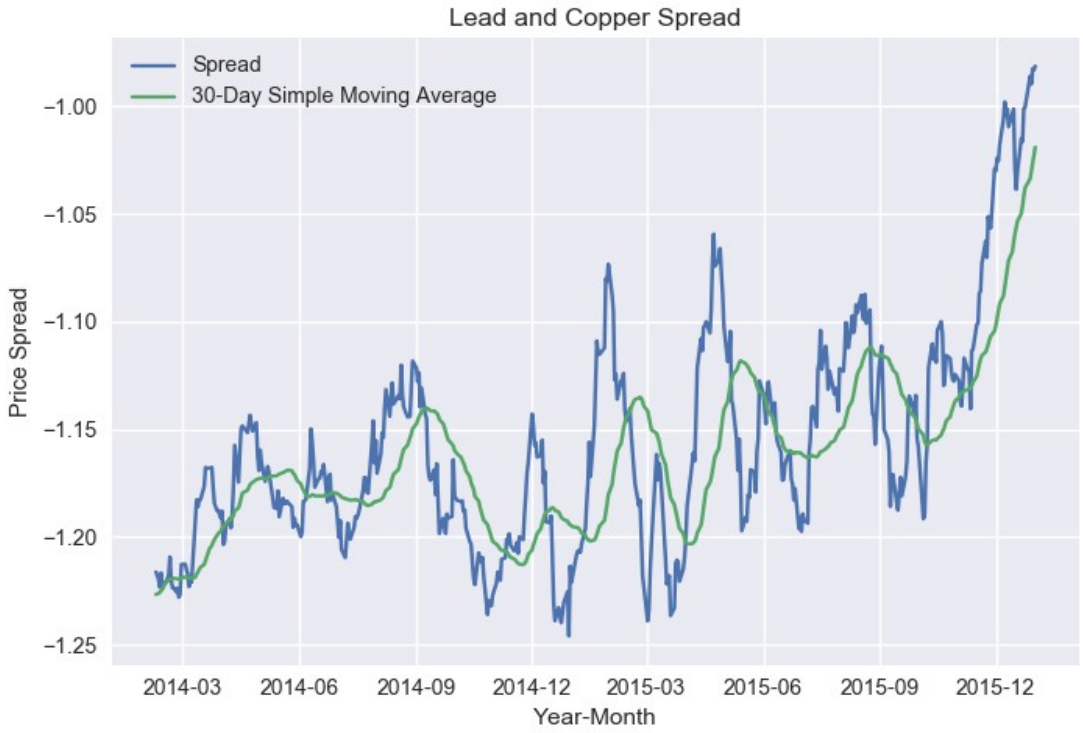


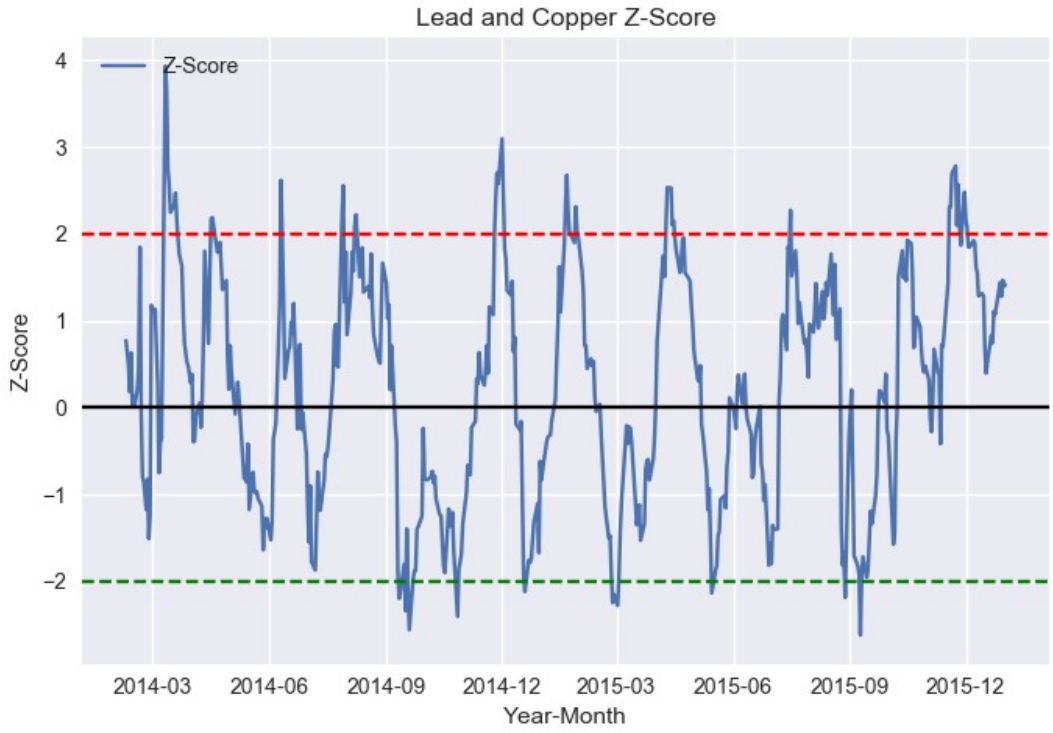
- Aluminium and Nickel





- Lead and Copper





The plots of the price spreads and Z-Scores confirm mean reversion, and thus, we can move forward and generate trade signals.

As discussed earlier, there can be following two strategies for generating trade signals:

- Strategy 1:
  Generate entry and exit signals only on the basis of Z-Scores.

- Strategy 2:
  Check both co-integration and Z-Score while entring and exiting.

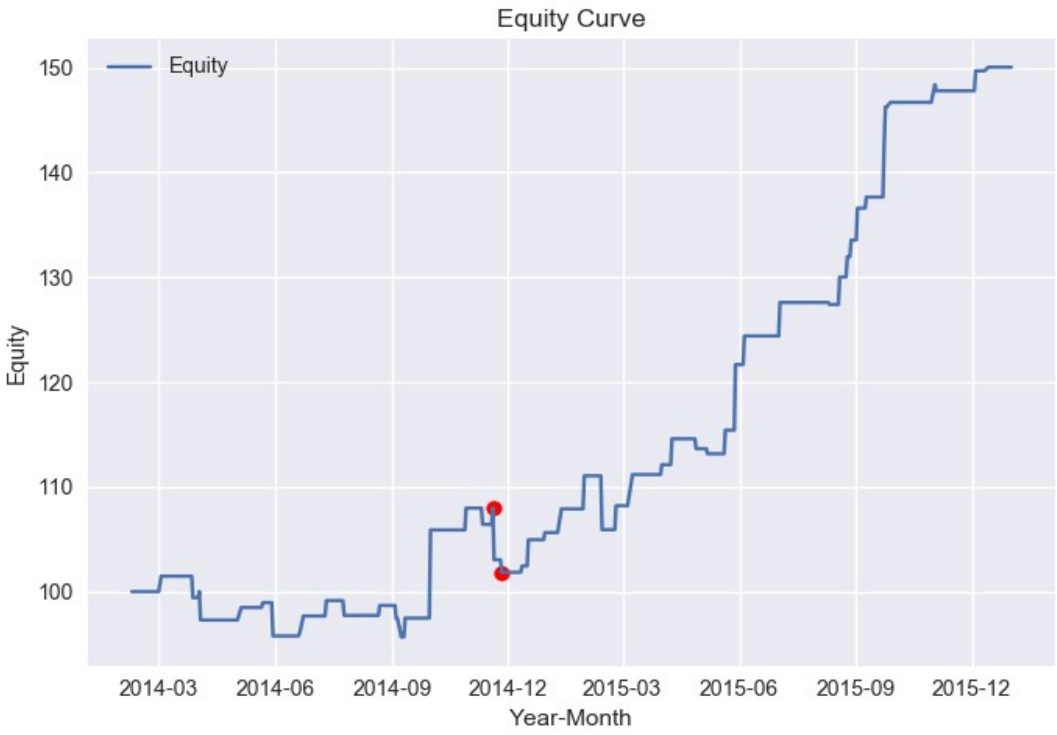
I back-tested a portfolio of the pairs selected using both the strategies discussed above. I assume an investment of Rs. 100 divided equally among all the pairs for the back-testing purpose. The performance parameters and the equity curve for both the strategies are as follows:

- Strategy 1:

  - Performance Parameters:

Parameter	**Value**
CAGR	22.496799%
Sharpe-Ratio	1.782392
Positive Trades	36
Negative Trades	14
Hit-Ratio	72%
Average Profit	1.891101%
Average Loss	-1.862480%
Maximum Drawdown	-5.690773%

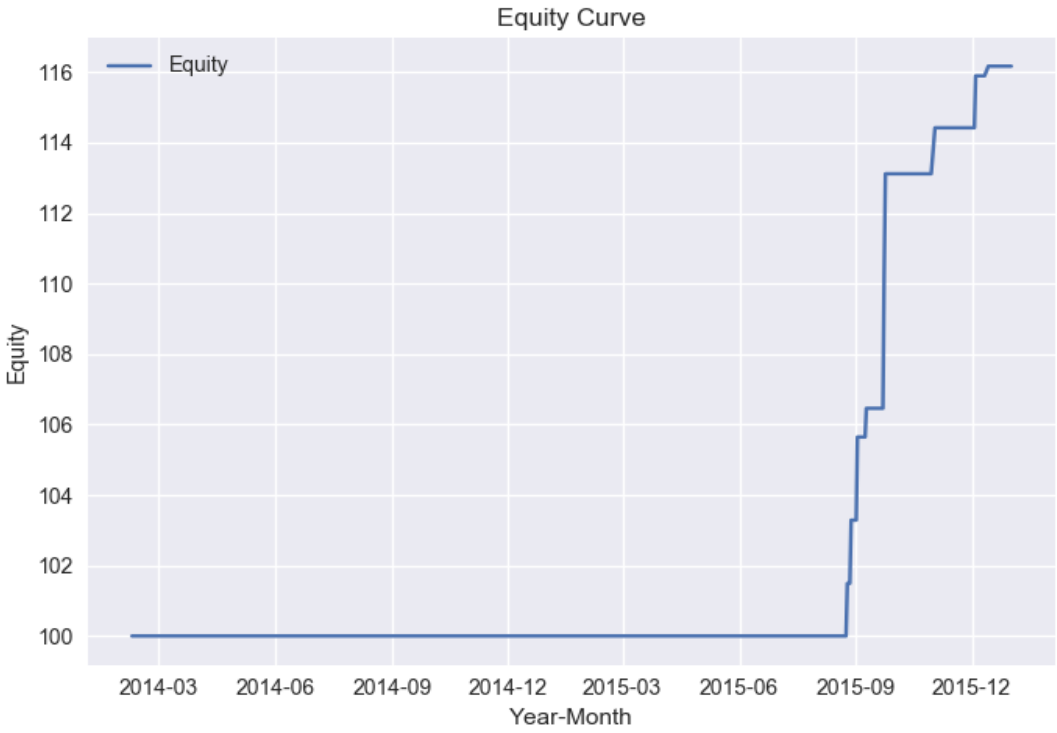
  - Equity Curve (with Maximum Drawdown):



- Strategy 2:
    - Performance Parameters:

Parameter	**Value**
CAGR	7.781883%
Sharpe-Ratio	1.884935
Positive Trades	9
Negative Trades	0
Hit-Ratio	100%
Average Profit	1.683726%
Average Loss	0.00%
Maximum Drawdown	0.00%

    - Equity Curve (with Maximum Drawdown):



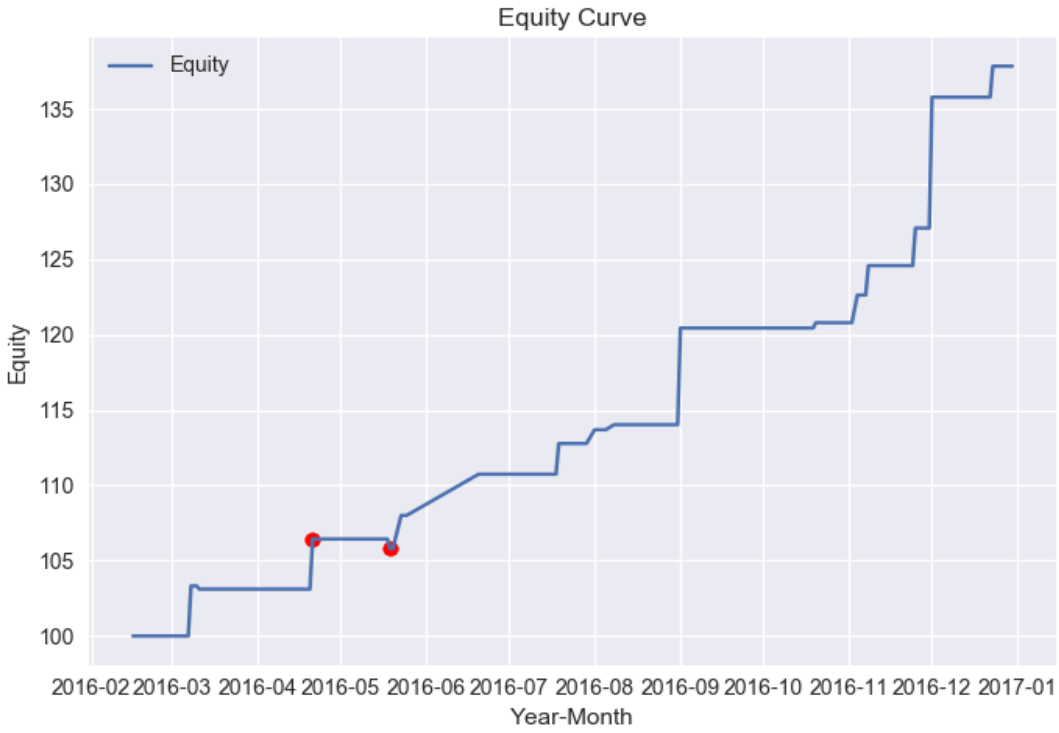
Out of Sample Backtest

For out of sample backtesting I have used continuous contracts from 02-Jan-2016 to 01-Jan-2017. The results for both the strategies are as follows:

- Strategy 1:
    - Performance Parameters:

Parameter	**Value**
CAGR	17.403%
Sharpe-Ratio	3.284380
Positive Trades	14
Negative Trades	2
Hit-Ratio	87.5%
Average Profit	2.393493%
Average Loss	-0.403976%
Maximum Drawdown	-0.600298%

    - Equity Curve (with Maximum Drawdown):



- Strategy 2:
    - No trade signal is generated during out of sample time-period.

Conclusion

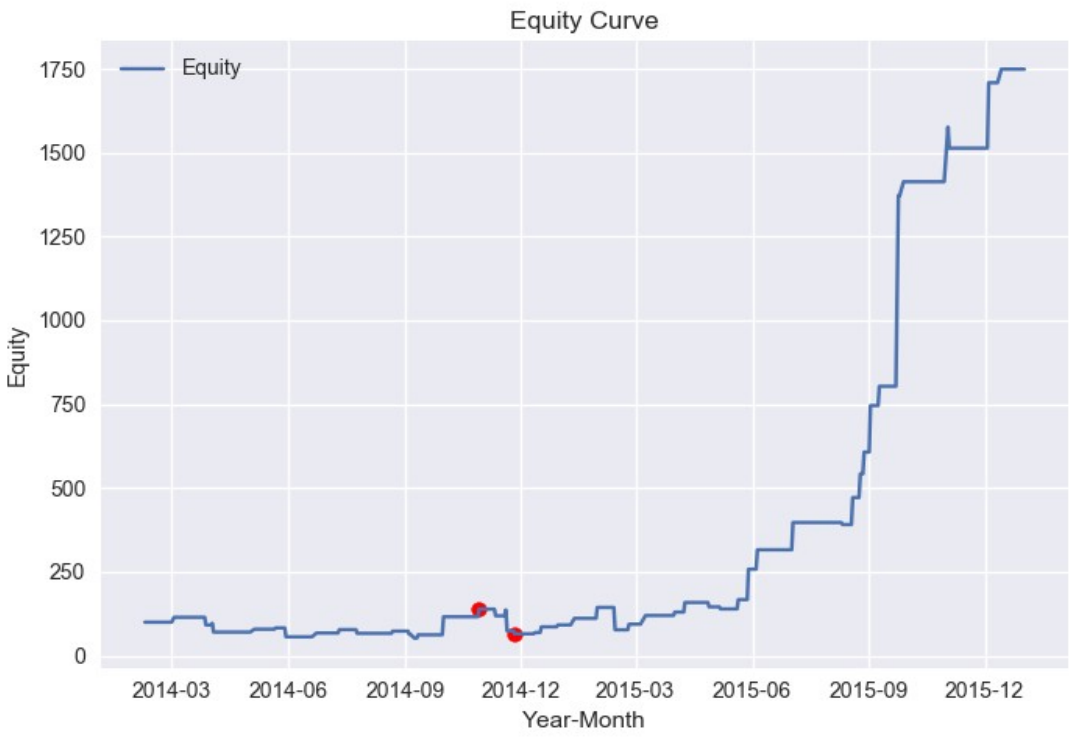
The backtesting results show that although Strategy 2 has a slightly better performance than Strategy 1 in terms of Sharpe ratio and hit-ratio, we notice that number of trades have reduced significantly. Moreover, we see that in out of sample backtesting, there is no trade on the basis of Strategy 2 throughout the out of sample time-period.

Strategy 1 on the other hand seems to be a more practical strategy. The strategy generates a Compound Annual Growth Rate (CAGR) of 22.5% for the in-sample period. Also, as shown by the equity curve, the cumulative return is around 50% for the in-sample period.

As seen during out of sample backtesting, the performance of Strategy 1 improves in terms of Sharpe-Ratio, hit-ratio, average profit to average loss ratio and maximum drawdown. However, it deteriorates in terms of CAGR and cummulative return.

All the results presented above have considered a leverage of 1. However, it is common to have a leverage of 10 for trading commodities. The equity curve for Strategy 1, using a leverage of 10 are is follows:

- Equity Curve (with leverage 10):



As can be seen, the cumulative return acheived with a leverage of 10 is 1649.12%. However, maximum drawdown also increases to -52.96%.

Caveats

- The strategy has not considered slippage and transaction costs involved in trading.
- The strategy uses continuous contracts available from quandl. However, for a more accurate performance evaluation of the strategy, one must consider the factor of rolling the contracts and costs associated with it.

Python Code

```
import quandl
import pandas as pd
import statsmodels.api as sm
import statsmodels.tsa.stattools as stats
import numpy as np
import matplotlib.pyplot as plt

# Start and End Dates
start = '2014-01-01'
end = '2016-01-01'

# Imports Continuous Contracts from Quandl
def import_data():
    # MCX Gold Contract
    gc = quandl.get("CHRIS/MCX_GC1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start, end_date=end)
    # MCX Silver Contract
    si = quandl.get("CHRIS/MCX_SI1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Aluminium Contract
    al = quandl.get("CHRIS/MCX_AL1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Lead Contract
    pb = quandl.get("CHRIS/MCX_PB1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Nickel Contract
    ni = quandl.get("CHRIS/MCX_NI1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Zinc Contract
    zn = quandl.get("CHRIS/MCX_ZN1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Copper Contract
    cu = quandl.get("CHRIS/MCX_CU1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Crude Oil Contract
    cl = quandl.get("CHRIS/MCX_CL1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Natural Gas Contract
    ng = quandl.get("CHRIS/MCX_NG1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Cardamom Contract
    crdm = quandl.get("CHRIS/MCX_CRDM1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)
    # MCX Cotton Contract
    ct = quandl.get("CHRIS/MCX_CT1", authtoken="KfYDcdPudPzu6X2si99D", start_date=start , end_date=end)

    # List containing commodity data
    data = [gc, si, al, pb, ni, zn, cu, cl, ng, crdm, ct]
    # List containing commodity symbols
    symbols=['GC', 'SI', 'AL', 'PB', 'NI', 'ZN', 'CU', 'CL', 'NG', 'CRDM', 'CT']
    return data, symbols

# Finds pairs from the imported data
def find_pairs(data, symbols):
    lookback = 360
    pairs_sym = []
    pairs_df = []
    for i in range(len(symbols)):
        for j in range(i+1, len(symbols)):
            df = pd.concat([data[i].Close, data[j].Close], axis=1)
            df = df.fillna(method='ffill')
            df = df.dropna()
            df.columns=['Y', 'X']
            # Get days on which ADF test confirms stationarity
            # and the days on which the pair is co-integrated
            # coint: contains days on which pair is co-integrated.
            # st: contains days on ADF test confirms stationarity for the pair.
            coint, st = get_coint(df, lookback)
            # Calculate percentage of the days.
            # If percentage is more than 40%, add it to list of pairs
            try:
                days = float(len(coint.dropna()))/len(st)
                res = np.corrcoef(df.Y, df.X)
                if days > 0.4 and res[0,1] > 0.6:
                    pairs_sym.append((symbols[i], symbols[j], days, res[0,1]))
                    pairs_df.append(df)
```

```
            except:
                continue
    return pairs_df, pairs_sym

# Finds days on which ADF test and co-integration test both pass for a pair
def get_coint(df, lookback):
    # Find rolling spreads with a window = lookback
    spreads = {}
    for i in range(lookback, len(df)):
        x_cons = sm.add_constant(df.X.iloc[i-lookback:i])
        res = sm.OLS(df.Y.iloc[i-lookback:i], x_cons).fit()
        spread = df.Y.iloc[i-lookback:i]-res.params[1]*df.X.iloc[i-lookback:i]
        spreads[df.index[i]] = spreads.get(df.index[i], spread)

    # Find days on which ADF test confirms stationarity
    st = pd.Series(index=df.index)
    for key, spread in spreads.iteritems():
        r = stats.adfuller(spread, maxlag=1)
        if r[1] < 0.05:
            st.ix[key] = r
    st = st.dropna()

    # Out of days on which ADF test confirms stationarity,
    # find the days on which co-integration is confirmed
    coint = pd.Series(index=st.index)
    for i in st.index:
        loc = df.index.get_loc(i)
        r = stats.coint(df.Y.iloc[loc-lookback:loc], df.X.iloc[loc-lookback:loc])
        if r[1] < 0.05:
            coint.ix[i] = r[1]
    coint = coint.dropna()
    # coint: contains days on which pair is co-integrated.
    # st: contains days on ADF test confirms stationarity for the pair.
    return coint, st

# Backtests a strategy. Generates Z-Scores, Trade Signal and Returns.
def backtest(df, coint=None, strategy=1):
    # Find the price ratio between the pair
    ratio = np.log(df.Y/df.X)
    ratio.name = 'Ratio'

    # Calculate Simple Moving Average and
    # Moving Standard Deviation of the price ratio
    mavg = ratio.rolling(window = 30).mean()
    mavg.name = 'MA_30'
    std = ratio.rolling(window = 30).std()
    std.name = 'STD_30'

    # Calculate Z-Score
    z_score = (ratio-mavg)/std
    z_score.name = 'Z'

    df_1 = pd.concat([df, ratio, mavg, std, z_score], axis=1)
    df_1 = df_1.dropna()

    # Find log returns for each commodity in the pair
    Y_log = np.log(df_1.Y/df.Y.shift(1))
    Y_log.name = 'Y_log'
    X_log = np.log(df_1.X/df.X.shift(1))
    X_log.name = 'X_log'

    df_1 = pd.concat([df_1, Y_log, X_log], axis=1)
    df_1 = df_1.dropna()

    # Z-Score thresholds
    entryZ = 2.0
    exitZ = 0.0
```

```
inLong = None
inShort = None

Y_sign = pd.Series(index=df_1.index)
Y_sign.name = 'Y_sign'
X_sign = pd.Series(index=df_1.index)
X_sign.name = 'X_sign'

rets = pd.Series(index=df_1.index)
rets.name = 'returns'

# Generate Trade signals and returns for Strategy 1
if coint is None:
    for i in range(len(df_1)):
        if (inLong == None) and (df_1.Z.iloc[i] < -entryZ):
            Y_sign.iloc[i] = 1
            X_sign.iloc[i] = -1
            inLong = i
            inShort = None

        if (inLong != None) and (df_1.Z.iloc[i] >= -exitZ):
            Y_sign.iloc[i] = -1
            X_sign.iloc[i] = 1
            rets_y = np.sum(df_1.Y_log.iloc[inLong+1:i+1])*Y_sign.iloc[inLong]
            rets_x = np.sum(df_1.X_log.iloc[inLong+1:i+1])*X_sign.iloc[inLong]
            rets.iloc[i] = (rets_y+rets_x)/2
            inLong = None
            inShort = None

        if (inShort == None) and (df_1.Z.iloc[i] > entryZ):
            Y_sign.iloc[i] = -1
            X_sign.iloc[i] = 1
            inLong = None
            inShort = i

        if (inShort != None) and (df_1.Z.iloc[i] <= exitZ):
            Y_sign.iloc[i] = 1
            X_sign.iloc[i] = -1
            rets_y = np.sum(df_1.Y_log.iloc[inShort+1:i+1])*Y_sign.iloc[inShort]
            rets_x = np.sum(df_1.X_log.iloc[inShort+1:i+1])*X_sign.iloc[inShort]
            rets.iloc[i] = (rets_y+rets_x)/2
            inLong = None
            inShort = None

# Generate Trade signals and returns for Strategy 1
elif coint is not None and strategy == 2:
    for i in range(len(df_1)):
        index = df_1.index[i]
        if index in coint.index and (inLong == None) and (df_1.Z.iloc[i] < -entryZ):
            Y_sign.iloc[i] = 1
            X_sign.iloc[i] = -1
            inLong = i
            inShort = None

        if index in coint.index and (inLong != None) and (df_1.Z.iloc[i] >= -exitZ):
            Y_sign.iloc[i] = -1
            X_sign.iloc[i] = 1
            rets_y = np.sum(df_1.Y_log.iloc[inLong+1:i+1])*Y_sign.iloc[inLong]
            rets_x = np.sum(df_1.X_log.iloc[inLong+1:i+1])*X_sign.iloc[inLong]
            rets.iloc[i] = (rets_y+rets_x)/2
            inLong = None
            inShort = None

        if index in coint.index and (inShort == None) and (df_1.Z.iloc[i] > entryZ):
            Y_sign.iloc[i] = -1
```

```
            X_sign.iloc[i] = 1
            inLong = None
            inShort = i

        if index in coint.index and (inShort != None) and (df_1.Z.iloc[i] <= exitZ):
            Y_sign.iloc[i] = 1
            X_sign.iloc[i] = -1
            rets_y = np.sum(df_1.Y_log.iloc[inShort+1:i+1])*Y_sign.iloc[inShort]
            rets_x = np.sum(df_1.X_log.iloc[inShort+1:i+1])*X_sign.iloc[inShort]
            rets.iloc[i] = (rets_y+rets_x)/2
            inLong = None
            inShort = None

    df_1 = pd.concat([df_1, Y_sign, X_sign, rets], axis=1)
    return df_1

# Backtest on the basis of strategy selected
def backtest_strategy(pairs_df, pairs_sym, strategy=1):
    rets = None
    for i in range(len(pairs_df)):
        df = pairs_df[i]
        df_1 = None
        if strategy == 1:
            df_1 = backtest(df)
        elif strategy == 2:
            coint,st = get_coint(df, lookback=360)
            df_1 = backtest(df, coint, strategy=2)

        ret = df_1.returns
        ret.name = pairs_sym[i][0]+'_'+pairs_sym[i][1]
        if rets is None:
            rets = ret
        else:
            rets = pd.concat([rets, ret], axis=1)
    return rets

# Performs out of sample backtesting of pairs
def out_sample_test(pairs, start_dt, end_dt, strategy=1):
    rets = None
    for p in pairs:
        y = quandl.get('CHRIS/MCX_'+p[0]+'1', authtoken="KfYDcdPudPzu6X2si99D", start_date=start_dt, end_date=end_dt)
        x = quandl.get('CHRIS/MCX_'+p[1]+'1', authtoken="KfYDcdPudPzu6X2si99D", start_date=start_dt, end_date=end_dt)

        df = pd.concat([y.Close, x.Close], axis=1)
        df = df.fillna(method='ffill')
        df = df.dropna()
        df.columns=['Y', 'X']

        df_1 = None
        if strategy == 1:
            df_1 = backtest(df)
        elif strategy == 2:
            coint,st = get_coint(df, lookback=360)
            df_1 = backtest(df, coint, strategy=2)

        ret = df_1.returns
        ret.name = p[0]+'_'+p[1]
        if rets is None:
            rets = ret
        else:
            rets = pd.concat([rets, ret], axis=1)
    return rets

# Calculates performance parameters and generates equity curve
def calc_performance(rets, num_of_pairs, lev=1):
    # Combine daily returns of all pairs to generate
    # daily returns of the portfolio
    total_rets = pd.Series(index=rets.index)
```

```
total_rets.name = 'Total Returns'
for idx in total_rets.index:
    total_rets.ix[idx] = np.sum(rets.ix[idx])

# Generate Equity Curve
eq_curve = pd.Series(index=total_rets.index)
for i in range(len(total_rets)):
    if i > 0:
        if total_rets.iloc[i]==0:
            eq_curve.iloc[i] = eq_curve.iloc[i-1]
        else:
            eq_curve.iloc[i] = eq_curve.iloc[i-1]*(1+lev*total_rets.iloc[i])
    else:
        # Initially, an investment of Rs. 100
        eq_curve.iloc[i]=100.0

## Calculate various performance parameters
# CAGR
cagr = ((eq_curve.iloc[-1]/eq_curve.iloc[0])**(1.0/2)-1)*100
# Positive Trades
pos_trades = [total_rets.ix[idx] for idx in total_rets.index if total_rets.ix[idx] > 0]
# Negative Trades
neg_trades = [total_rets.ix[idx] for idx in total_rets.index if total_rets.ix[idx] < 0]
# Number of Positive Trades
num_profit_trades = len(pos_trades)
# Number of Negative Trades
num_loss_trades = len(neg_trades)
# Hit Ratio
hit_ratio = 0
if num_profit_trades > 0:
    hit_ratio = float(num_profit_trades)/(num_profit_trades+num_loss_trades)
# Average Profit
avg_profit = np.mean(pos_trades)

# Sharpe Ratio
sharpe = 0.0
if not all(total_rets[0] == item for item in total_rets):
    sharpe = np.sqrt(252)*(np.mean(total_rets)/np.std(total_rets))

# Maximum Drawdown
prev_max = 0
prev_min = 0
max_loc = 0
for i in range(1, len(eq_curve)):
    if eq_curve.iloc[i] > eq_curve.iloc[max_loc]:
        max_loc = i
    elif (eq_curve.iloc[max_loc]-eq_curve.iloc[i]) > (eq_curve.iloc[prev_max]-eq_curve.iloc[prev_min]):
        prev_max = max_loc
        prev_min = i
max_dd = (eq_curve.iloc[prev_min]/eq_curve.iloc[prev_max])-1

l = ['CAGR', 'Sharpe-Ratio', 'Positive Trades', 'Negative Trades', 'Hit-Ratio', 'Average Profit']
v = [cagr, sharpe, num_profit_trades, num_loss_trades, hit_ratio*100, avg_profit*100]
if hit_ratio < 1.0:
    # Average Loss
    avg_loss = np.mean(neg_trades)
    l.append('Average Loss')
    v.append(avg_loss*100)
l.append('Maximum Drawdown')
v.append(max_dd*100)
labels = pd.Series(l)
vals = pd.Series(v)

df_params = pd.concat([labels, vals], axis=1)
df_params.columns = ['Parameter', 'Value']
df_params.index = range(1, len(df_params)+1)
```

```
# Print performance parameters
print df_params

# Plot Equity Curve along with maximum drawdown
plt.plot(eq_curve.index, eq_curve, label='Equity')
plt.title('Equity Curve')
plt.xlabel('Year-Month')
plt.ylabel('Equity')
plt.legend(loc='upper left')

if prev_min > prev_max:
    plt.scatter(eq_curve.index[prev_max],eq_curve.iloc[prev_max], color='red')
    plt.scatter(eq_curve.index[prev_min],eq_curve.iloc[prev_min], color='red')
return eq_curve
```

Bibliography

Satistical Arbitrage and Pairs Trading Lecture by Mr. Shaurya Chandra, QuantInsti.