MS&E 448 Final Project: Statistical arbitrage

Jonathan Tuck Raphael Abbou Vin Sachidananda

June 10, 2020

1 Introduction

Statistical arbitrage comprises a group of trading strategies which seek to identify, through quantitative means, mispriced assets by analyzing relative price movements. As an example, consider a universe with two equities at time t, A_t and B_t, issued by public companies which each control 50% of electricity sales in the United States. Assuming that the two companies are similarly structured with regard to fundamentals, the two equities should tend to move in tandem; that is to say, the stochastic process $A_t - B_t$ should have mean reverting properties with $\mu = 0$. Informally, statistical arbitrage takes advantage of when the two equities' prices move out of tandem, *i.e.*$A_t - B_t > \epsilon, \epsilon > 0$, with the expectation that $A_{t+\delta} - B_{t+\delta} = 0$ at some later date $t + \delta$. To develop a trading strategy using this information, a trader may go short in equity A and long in B at time t and exit both positions at time $t + \delta$ netting a profit of ϵ. This simplified case is commonly referred to as *pairs trading*; statistical arbitrage generalizes the notion of pairs trading and has been applied on groups of equities, ETFs, currencies and derivatives.

In this paper, we propose two statistical arbitrage strategies which use sparse optimization and lagged correlation, respectively, to find groups of stocks whose relative price movements are mean reverting. Additionally, we propose methods for devising trading policies on these groups of stocks.

In particular, the contributions of our project are:

- Modeling and testing of statistical arbitrage strategies using sparse optimization formulations for identifying co-integration asset buckets, and
- Modeling and testing of statistical arbitrage strategies using lagged correlation metrics.

2 Background

2.1 Statistical arbitrage

In this section, we provide the intuition and mathematical properties associated with statistical arbitrage trading strategies.

2.1.1 Intuition

In developing a statistical arbitrage trading strategy, two challenges must be solved (i) first, one must identify groups of equities whose relative price differences are mean reverting and (ii) second, entry and exit points for trades must be identified in a manner that maximizes risk adjusted returns. Various statistical approaches have been used utilized to solve the first task including principal components analysis (PCA), autoregressive models, co-integration, volatility modeling, and time series analysis. The second task is typically solved by defining trading policies based on portfolio optimization and covariance estimation. In this paper, we devise two trading statistical arbitrage strategies. To solve the first step, these strategies respectively use sparse optimization and lagged correlation metrics to find groups of equities whose relative price movements are mean reverting. The second task is then solved with a policy which enters and exits trades using z-scores on the difference process of the groups derived in the first step.

2.1.2 Definition

First, one would like to identify groups of assets whose relative price movements are mean reverting. Let $\mathcal{E}, |\mathcal{E}| = K$ be the universe of assets under consideration and $y_t^i, t \in \mathcal{T}, i \in \mathcal{E}$ be the price of an asset i at time t. At time t, one is concerned with finding subsets of $\mathcal{E}$, $L, S \subsetneq \mathcal{E}$, such that for fixed finite δ:

$$\sum_{i \in L} y_t^i - \sum_{i \in S} y_t^i > \mu + \epsilon, \quad \mathbb{E}_{y_{t+\delta}^i}[\sum_{i \in L} y_{t+\delta}^i - \sum_{i \in S} y_{t+\delta}^i] = \mu, \quad \epsilon \in \mathbb{R}$$

The second step, which is common across many families of trading strategies, seeks to maximize returns subject to risk using a trading policy Π on L, S. Here we provide a policy which seeks to maximizes the Sharpe ratio, a commonly used metric for risk adjusted returns.

Assume $R = \gamma$ is the risk free return (U.S. Treasury bonds). A policy Π is able to enter a position, do nothing, or short a position at time $t \in \{0, 1, ...\delta\}$ which correspond to the decisions in $\{1, 0, -1\}$ respectively. Notationally, consider a policy with δ timesteps, $\Pi \in \{1, 0, -1\}^\delta$, where the decision made at timestep $t \in \{0, 1, ...\delta\}$ is denoted as Π_t.

$$\Pi^* = \underset{\Pi_t \in \{1,0,-1\}^\delta}{\operatorname{argmax}} \frac{E[(\Pi_t \alpha) - \gamma]}{\sigma(\Pi_t \alpha)}$$

For a process, $\sum_{i \in L} y_t^i - \sum_{i \in S} y_t^i$, which has been identified as having mean reverting properties, one can quantify the expected risk adjusted return for a decision made at time t, Π_t, carried out through the entire horizon, $t + \delta$. At time t, let $\sum_{i \in L} y_t^i - \sum_{i \in S} y_t^i = \mu + \epsilon$ and let σ be the standard deviation of the returns.

$$\Pi^t = \max_{\Pi_t \in \{1,0,-1\}} \Pi_t \frac{\epsilon - \gamma}{\sigma\sqrt{\delta - t}},$$

Intuitively, one would like to enter and exit positions as $|\epsilon - \gamma|$ becomes large relative to σ.

2.2 Co-integration

In this section, we briefly define and discuss co-integration, which is a condition that is typically desired of a collection of time series signals used in statistical arbitrage.

Order of integration. A time series signal x_t is said to be integrated of order d if $\mathbf{diff}^d(x_t)$, the signal obtained by applying the difference operator to x_t d times, is stationary. (In this paper, we only consider the case of $d = 1$, as the difference in stock prices is typically taken as stationary.)

We call a collection of m time series signals, $(y_t)_1, \ldots, (y_t)_m$, that are each integrated of order d, *co-integrated* if the time series $\sum_{i=1}^m w_i(y_t)_i$ is integrated of order less than d. As a simple example, consider two time series A_t and B_t. Then, we call A_t and B_t co-integrated if it holds that $A_t + \kappa B_t$ is stationary for some $\kappa \in \mathbf{R}$. Informally, if a collection of time series signals are co-integrated, then their statistical properties tend to stay constant over time. (In practice, it is quite hard for this to be true exactly; as such, we are generally concerned with whether this holds approximately, rather than exactly.)

Testing for co-integration. There are many statistical tests available to test if two time series signals are co-integrated. Among the most common are the Engel-Granger test [EG87], the Johansen test [Joh91], and the Phillips–Ouliaris test [PO90]. In practice, for baskets of time series, one time series is compared to the linear combination of the other time series. For a concrete example of using one of these tests, see §5.1 for an instance of the Engel-Granger test.

2.3 Lead-lag models

In this section, we will describe another approach that leads to the creation a basket of co-integrated stocks. We believe that the co-integration for a pair of stocks is related to a lead-lag correlation effect: let's assume that we have two stocks, whose prices are denoted P_t and Q_t, and that the returns of P at time t are correlated with the lagged returns of Q at time $d + dt$. A first strategy would be to go long (respectively, short) Q whenever the returns of P are positive (respectively, negative). However, we manage to extend this idea to link it with the idea or pair-trading and come up with what we expect to be a more robust strategy. Assuming that both P and Q have no alpha, if the return of P at a given time is excessively large, we want to go long Q (as its lagged return is expected to follow the same

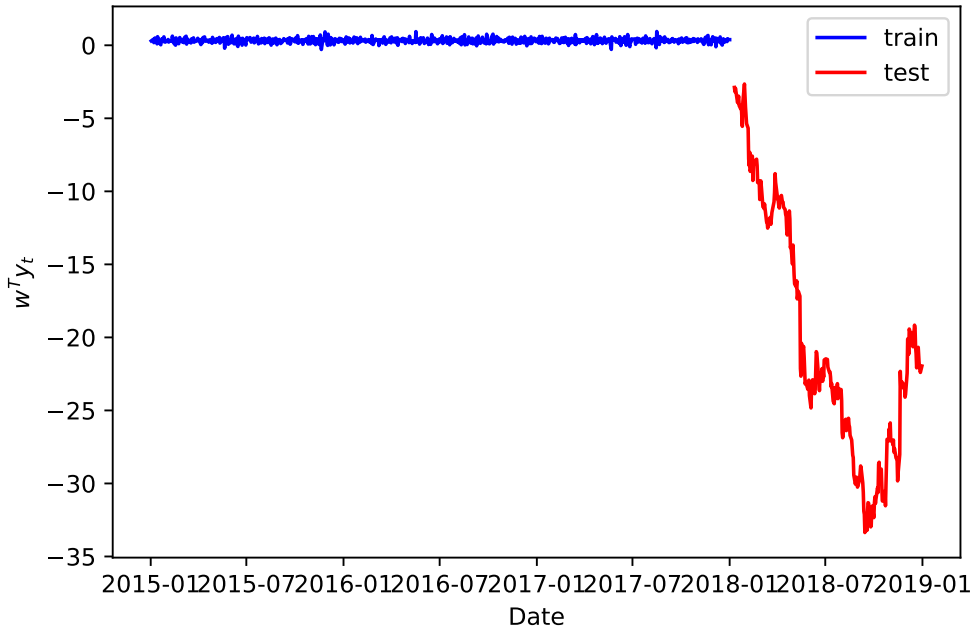

Figure 1: An example of the basic problem (1) overfitting.

behavior), but also short P, as we expect that price to revert back with respect to Q. We thus get a strategy for a pair that exhibits lead-lag correlation that is similar to pair-trading.

3 Sparse statistical arbitrage

Basic problem. The basic statistical arbitrage problem can be cast as a constrained quadratic program

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{T}(w^T y_t - \mu)^2 \\ \text{subject to} & w \in \mathcal{C} \end{array} \tag{1}$$

where $w \in \mathbf{R}^m$ and $\mu \in \mathbf{R}$ are the optimization variables, and $\mathcal{C}$ encodes other constraints. For convex $\mathcal{C}$, the problem (1) is a convex optimization problem and can be solved efficiently [BV04].

An issue with this basic problem is that this approach, by itself, tends to significantly overfit in practice. That is, in training, $w^T y_t$ tends to produce a time series that is perfectly stationary; in test, however, it is very rare that the same statistical properties hold. We illustrate this in figure 1, with $m = 500$ stocks.

Sparse problem. To the basic problem (1) we add regularization to reduce overfitting. The new problem is to be solved is then

$$\begin{array}{ll} \text{minimize} & \sum_{t=1}^{T}(w^T y_t - \mu)^2 + \lambda\|w\|_1 \\ \text{subject to} & w \in \mathcal{C}, \end{array} \tag{2}$$

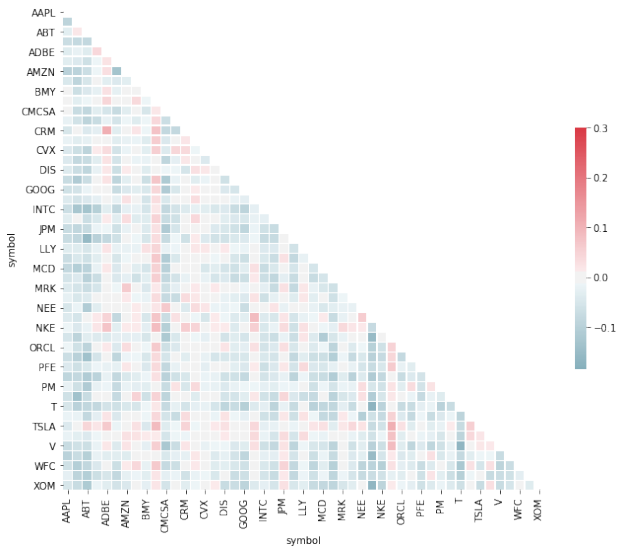

Figure 2: Lagged Correlation Matrix.

where $w \in \mathbf{R}^m$ and $\mu \in \mathbf{R}$ are the optimization variables, and $\mathcal{C}$ encodes other constraints, like in (1). Additionally, $\lambda > 0$ is a hyper-parameter on the ℓ_1 norm of w, encouraging the minimizer to be sparse, resulting in a portfolio of a small number of assets.

Polishing. In practice, once we find $w^\star$, the minimizer of (2), it is useful to then solve the problem (1), with the additional constraint that $w_i = 0$ for all $i \in \mathcal{I}$, where $\mathcal{I}$ is the index set of all indices i for which $w_i^\star = 0$.

4 Lagged correlation

The ideas that we present in this section are inspired by a paper that proposes ways to establish lead-lag relationships between stocks [CCK15]. In this approach, we have used data from Maystreet.

Lead-Lag correlations Computation. We say that two stocks P and Q have a lead-lag correlation when $\frac{dP_t}{P_t}$ and $\frac{dQ_{t+dt}}{Q_{t+dt}}$ are significantly correlated. In order to compute the lagged correlation for a given training period, we compute the matrix of returns A of our stocks, and B the matrix of lagged returns, and $\Sigma = AB$ will be our matrix of lagged returns. We first study on the 50 largest CAP US companies our lagged returns for data sampled every 15 minutes (Figures 1), and try to get cluster of lagged-correlated stocks, as this will be interesting when we will establish baskets of stocks (Figure 3), as described below. We get promising results that show that there is room for us to get baskets of lagged-correlated stocks.

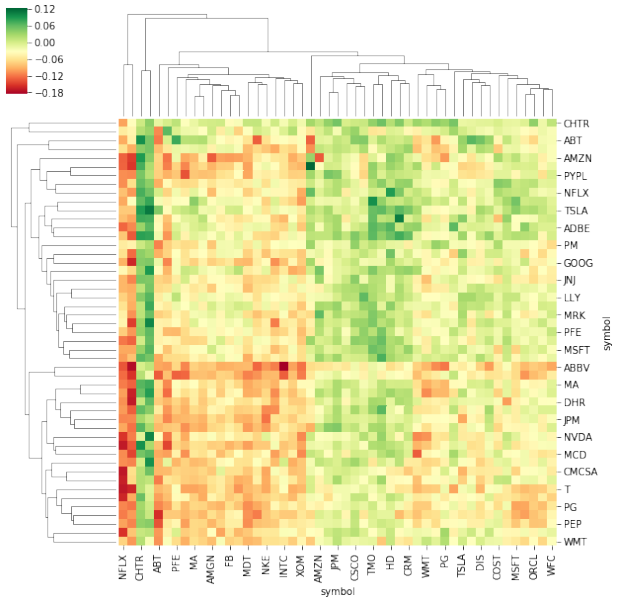

Figure 3: Lagged Correlation Clusters.

Strategy description. In order to generate our baskets of co-integrated stocks, we will establish Lagged Correlation graph: for all pairs of stocks that are lagged-correlated, we draw an edge between these two nodes. We then define a basket of stock as a connected component of the graph: all the stocks in this connected component will be indirectly lagged-correlated. Then for each basket, we regress the returns of the most connected node against the returns of the other stocks, as we assume that this node will provide the largest number of non-zero coefficients. We add a L_2 penalty (Ridge regression) in order to avoid flipping of coefficients while we regress on different train sets, as we are working by design with correlated stocks.

The difference between the lead stocks and the laggers weighted by the regression coefficients is assumed to be mean-reverting. Each time our basket go above (respectively, under) one standard deviation of its mean (both being computed on the training set), we go short (respectively, long) of the basket. We rebalance our betas every months, and backtest our PnL on the next (out-of-sample) month.

Establishing Lead-Lag correlations. When we say that two stocks are lagged-correlated, we want to be confident about that fact. What we do is we use a bootstrap technique, where for each bootstrap b, we shuffle the rows of the lagged matrix of returns B, and then we compute a new lagged-correlation matrix Σ_b. We thus get a distribution of randomized lagged correlation for each pair of stocks, and we consider that two stocks are positively (respectively, negatively) lagged-correlated if their original lagged correlation is in the top (respectively, bottom) 5% of this distribution (5% corresponds to the p-value that we selected for our test). As we are doing multiple tests (one for each pair of stock, which is $N^2 = 2500$ tests, where $N = 50$ is the number of stocks in our universe), we add a Bonferroni correction to each of our test (*i.e.*, we divide our p-value by N^2).

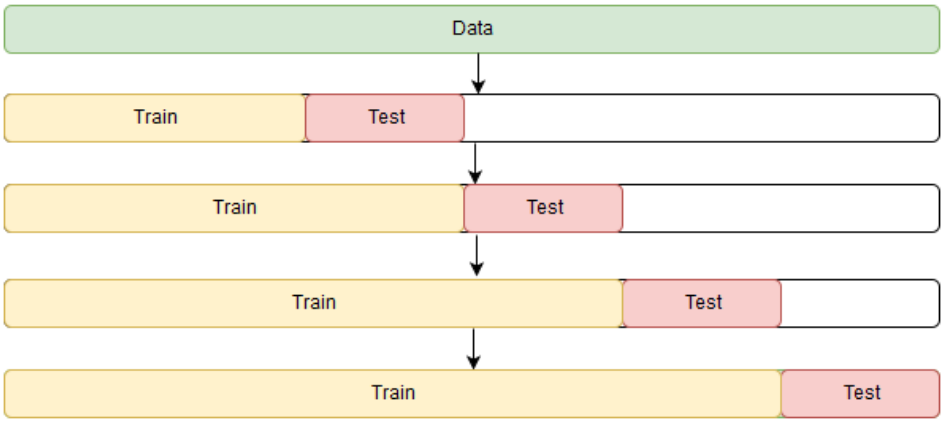

Figure 4: Cross-validation procedure used.

5 Experiments

Validation. In order to retain the temporal structure of the time series data, it is critical to not shuffle the data randomly while validating a model. We therefore employ the procedure illustrated in figure 4. For some number of folds (taken to be five in the experiments), we select a date, train on all data before that date, and test on data after that date.

5.1 Sparse optimization

For this example, we consider $m = 28$ stocks classified in the "Energy" sector of the S&P 500. We train and validate from January 1, 2014 to January 1, 2016, and test from January 2, 2016 to January 1, 2017.

Baskets. In practice, one would collect many baskets of stocks and perform the procedure outlined in this section; empirically, we found many portfolios using this procedure, which enjoyed similar performance. To simplify our analysis, we look at only one of those baskets.

Constraints. We add a constraint on market neutrality. Each of the assets each have a market beta, collected in $\beta \in \mathbf{R}^m$, which is taken as problem data. Then, we can add a market neutrality constraint as

$$|w^T\beta| \leq \epsilon.$$

This forces the market portfolio to be insensitive to the market by a factor of $1/\epsilon$. (The market, by definition, has a market beta of one.) Therefore, we have

$$\mathcal{C} = \{w \in \mathbf{R}^m \mid -\epsilon \leq w^T\beta \leq \epsilon\}.$$

```
Nonzero weights:
APA 0.39888001844326315
COG -0.4113619169466383
CVX -0.8906814059244456
DVN 0.2599217257736349
EQT -0.37149490868708585
FTI 0.7475351494216929
HAL 0.41189146949001026
HP -0.26619894347392736
KMI -0.7567502256169396
MPC -1.1515298928003135
MRO 1.3938999332485456
NBL 0.6427229074945807
NOV -0.5664895524348179
OXY -0.9299458067924226
PXD -0.1457447179318823
RRC 0.4317833087790374
SLB -0.8402034439359056
SPY 1.0
```

Figure 5: Overall portfolio of example in §5.1.

Results. We use market neutrality bound $\epsilon = 0.1$ and ℓ_1-regularization hyper-parameter $\lambda = 1.2$; λ was chosen using a crude hyper-parameter search over the validation set. The overall portfolio is given in figure 5. The overall portfolio includes 18 unique stocks.

Figure 6 plots the overall portfolio over the training and test set dates. The portfolio remains stable over the course of the entire training and test set, and the portfolio beta is both close to zero over the entire time frame.

Policy. We use this portfolio in a simple trading policy. At any given time, we are allowed to be long 1 share, short 1 share, or have no shares in the overall portfolio. We short/long 1 share when $\mu + \sigma \leq w^T y_t \leq \mu + 2\sigma$ / $\mu - \sigma \geq w^T y_t \geq \mu - 2\sigma$, respectively. Here, μ and σ are now the rolling, 30-day, backward means and standard deviations, respectively, which can be seen in figure 11. We run the policy for the entirety of the test set dates. We find that the policy yields a 16% return over the course of the test, with a maximum drawdown of approximately 8%.

Testing for co-integration. As a further example, we run an Engel-Granger test on the portfolio to test for co-integration, the results of which are seen in figure 8. On the training set, the t-statistic of the test is -6.30, with an approximate p-value of 3.42×10^{-7}. On the test set, the t-statistic of the test is -3.84, with an approximate p-value of 1.18×10^{-2}, suggesting that the portfolio is co-integrated with SPY over both the training and test sets.

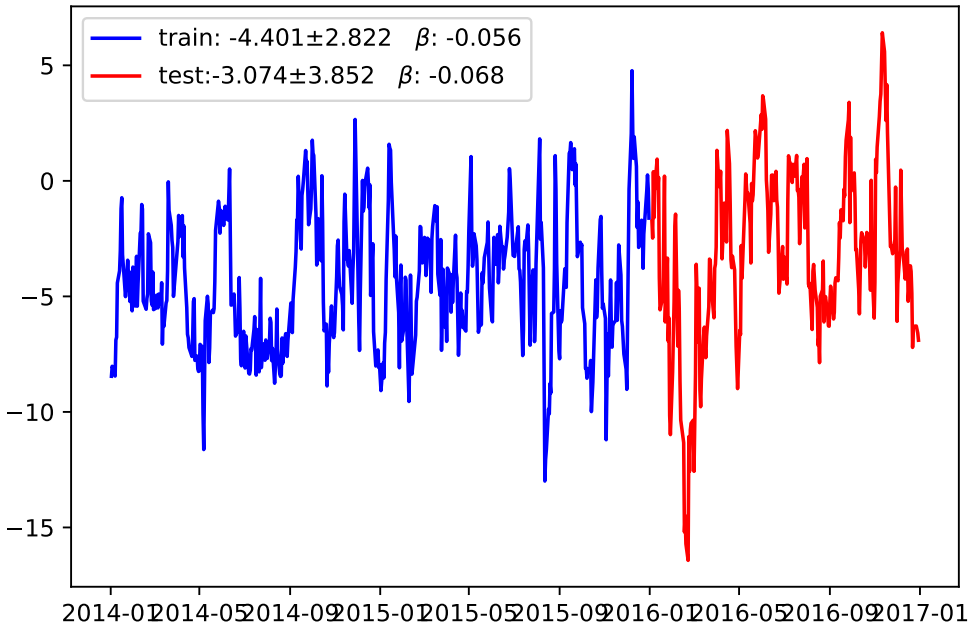

Figure 6: Overall portfolio over the training and test set dates, for the example in §5.1.

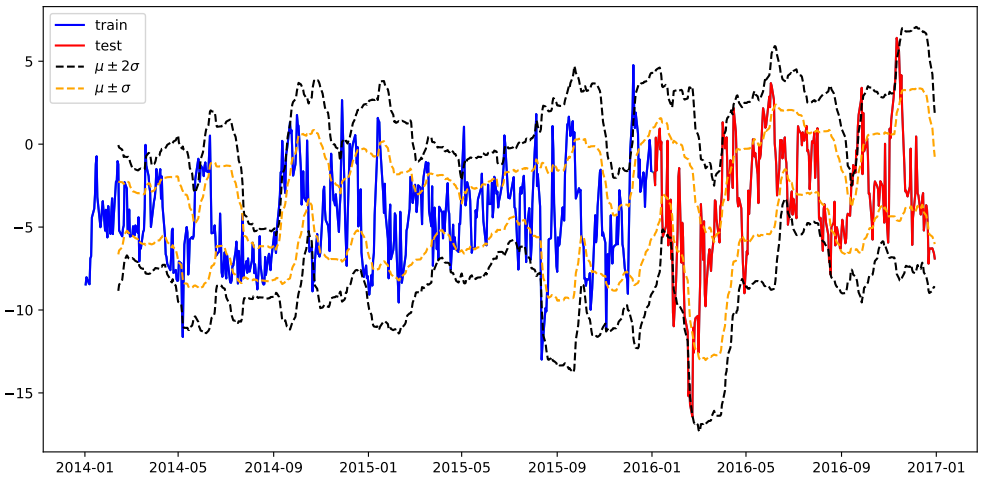

Figure 7: Sparse portfolio over the training and test sets, with bounds corresponding to $\mu \pm \sigma$ and $\mu \pm 2\sigma$.

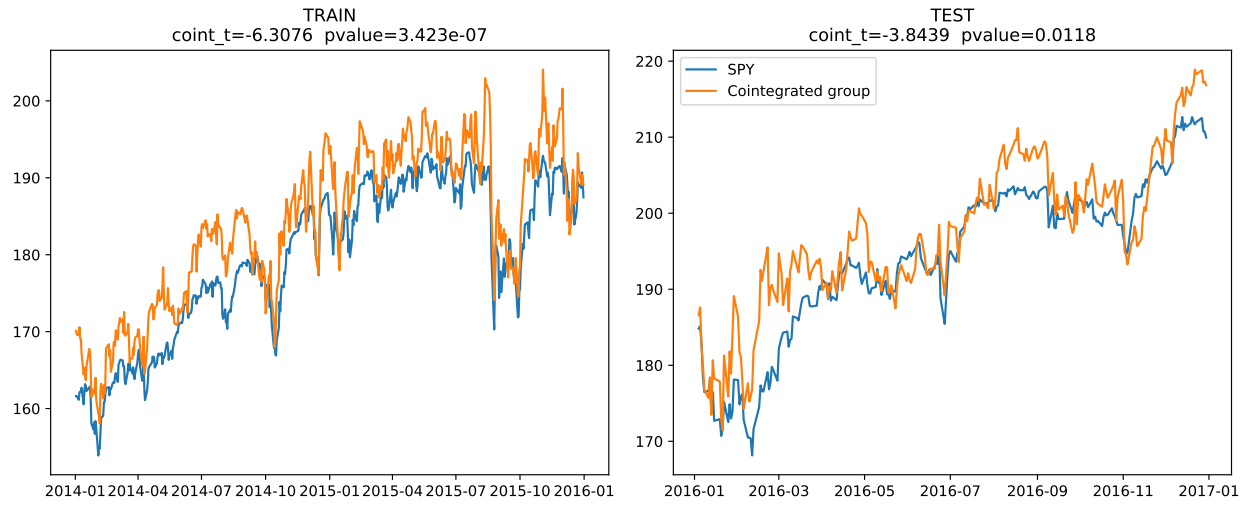

Figure 8: Results of Engel-Granger test for example in §5.1.

5.2 Lagged correlation

We manage to find (Figure 9) the presence of statistically significant lagged-correlation relationships in our universe of stocks. Interestingly, we can notice that these links drastically vanish when we increase the data frequency to daily data. We would interpret that phenomenon by saying that because of transaction costs, the market exhibits inefficiencies at smaller time-scales, as shot-term lead-lag arbitrage would probably result in high trading costs, hence the existence of these lagged correlation at that time scale.

In our final strategy, we use 15 minute data. We cross-validate our strategy with a rolling window of one month. We notice that our baskets of stocks are stable and stationary: we define a coefficient that we call 'retention', which is the ratio of stocks in a basket that stays from one month to the other, and get values between $70 - 75\%$ in 2016-2017. Finally, without taking into account trading costs, we get very exciting out-of-sample results: over 30% cumulative returns for 2016-2017 and over 70% for 2017-2018 (Figure 10 and 11)! As we are trading often (multiples times a day), we believe that trading costs and bid-ask spreads would have a very important impact on the real PnL of the strategy, but these results look very promising.

6 Conclusion

Through this project, we have proposed and analyzed two statistical arbitrage strategies which make use of sparse optimization and lagged correlation. Interestingly, we find that our sparse optimization approach is able to find cointegrated baskets of stocks. We test our strategies and find that both are able to return in excess of 10% for each trading year simulated.

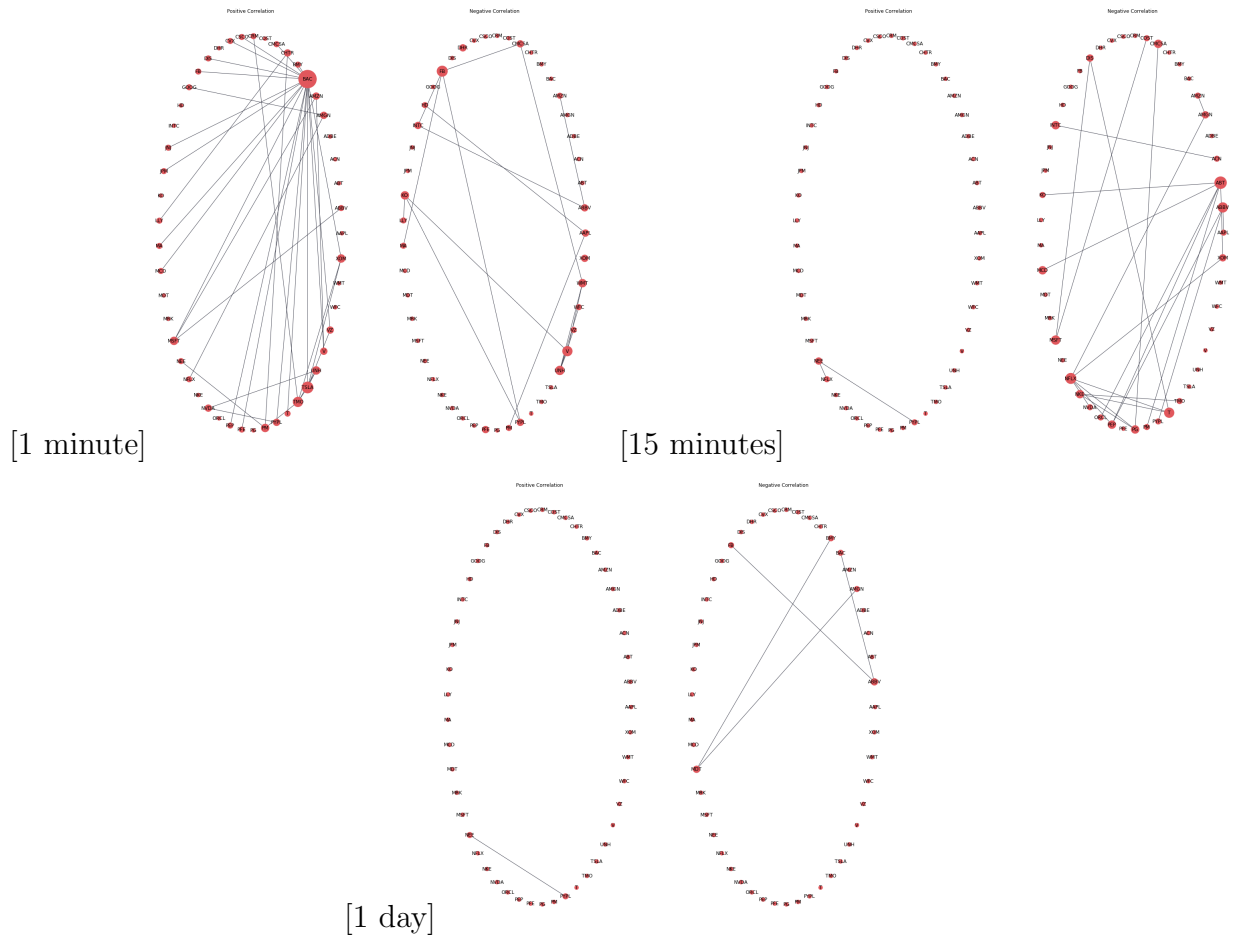

Figure 9: Impact of Data Frequency on Lagged-Correlation

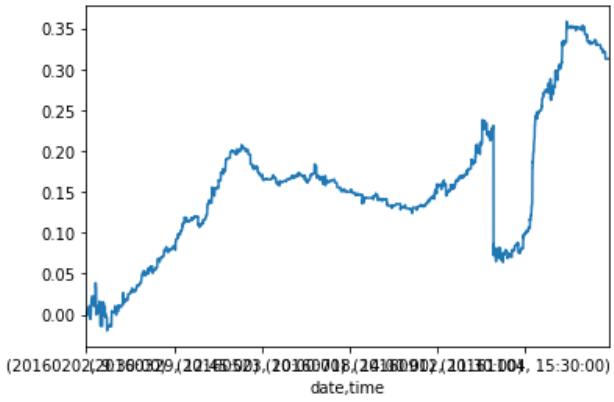

Figure 10: Cumulative Returns in percents for the 2017-2018 period.

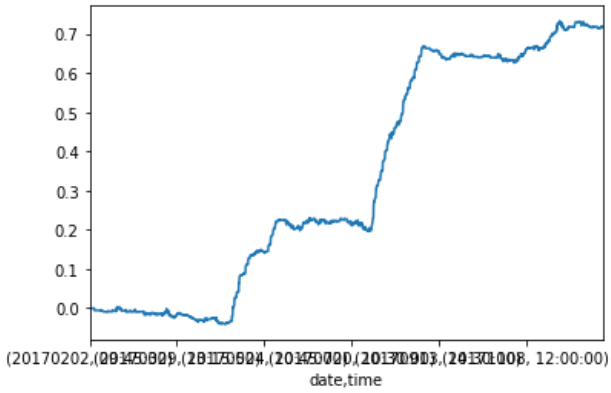

Figure 11: Cumulative Returns in percents for the 2016-2017 period.

Acknowledgements

The authors would like to thank Lisa Borland and Enguerrand Horel for their useful suggestions during the project.

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CCK15] R. N. Mantegna H. E. Stanley C. Curme, M. Tumminello and D. Kenett. Emergence of statistically validated financial intradaylead-lag relationships. *Quantitative Finance*, 55(8):375—-1386, 2015.

[EG87] R. F. Engle and C. W. J. Granger. Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):251–276, 1987.

[Joh91] S. Johansen. Estimation and hypothesis testing of cointegration vectors in gaussian vector autoregressive models. *Econometrica*, 59(6):1551–1580, 1991.

[PO90] P. C. B. Phillips and S. Ouliaris. Asymptotic properties of residual based tests for cointegration. *Econometrica*, 58(1):165–193, 1990.

Appendix

6.1 Data Processing Scripts

In this section, we provide the code used to query stock quotes used in our experimentation.

6.1.1 CRSP Daily Quotes

In this subsection, we provide code used to query daily stock quotes from the Center of Research in Securities Prices (CRSP) using the Wharton Research Data Services (WRDS) Python API.

```
import wrds
import pandas as pd
import datetime
from dateutil.relativedelta import relativedelta
import cPickle

# Connect to WRDS
db = wrds.Connection()

###############################################################################
#### SQL Query - Get largest N market cap stocks at start of each month ####
###############################################################################
# Initialize dictionary to store top N gvkeys for every month and specify
    timeframe of interest
N = 1500
gvkey_month, tickers_month, cusip_month = {}, {}, {}

start_date, end_date = '2018-01-01', '2018-03-01'
curr_date = datetime.datetime.strptime(start_date,'%Y-%m-%d')
last_date = datetime.datetime.strptime(end_date,'%Y-%m-%d')

# Reference df for primary security
q10 = ("select gvkey,primiss from compm.secm")
primiss_df = db.raw_sql(q10)

while curr_date < last_date:
    curr_date_string = curr_date.strftime('%Y-%m-%d')
    print(curr_date.date())

    # Query to get list of N companies with top market cap for the given month
    q1a = ("select distinct
        a.gvkey,a.latest,b.cshoq,b.prccq,b.mkvaltq,b.cshoq*b.prccq as
        market_cap,b.curcdq "
         "from "
            "(select gvkey,max(datadate) as latest "
             "from "
             "compm.fundq where datadate < '%s' "
             "group by gvkey) a inner join "
                "(select gvkey,datadate,mkvaltq,cshoq,prccq,curcdq "
                   "from compm.fundq where cshoq>0 and prccq>0 and curcdq='USD'
```

```
                    and mkvaltq>0) b "
        "on a.gvkey = b.gvkey and a.latest=b.datadate "
         "order by market_cap desc "
        "limit %i")%(curr_date_string, N)

    # merge the security flag
    mrk_df = db.raw_sql(q1a)
    mrk_df = mrk_df.merge(primiss_df,on='gvkey',how='left')
    gvkey_list_month = mrk_df['gvkey'][mrk_df['primiss']=='P'].values.tolist()
    gvkey_month[curr_date.date()] = set(gvkey_list_month)

    # increment the date for next month
    curr_date = curr_date + relativedelta(months=1)

# Map from gvkey to ticker for each month
cusip_ticker_map = {}
for date in gvkey_month:
    # change format to be compatible with sql query
    query_set = list(gvkey_month[date])
    query_set = tuple(["'%s'"%str(i) for i in query_set])
    query_set = ",".join(query_set)

    # Query to get fundamental Data
    q2 = ("select datadate,gvkey,tic,cusip "
        "from compm.fundq "
         "where gvkey in (%s) and datadate > '%s' ")%(query_set, date)
    fundq_df = db.raw_sql(q2)
    tickers_month[date] = list(set(fundq_df.tic))
    cusip_month[date] = list(set(fundq_df.cusip))

    month_cusip_ticker = fundq_df.groupby('cusip')
    month_cusip_ticker = dict(month_cusip_ticker['tic'].unique())
    for cusip in month_cusip_ticker:
        if cusip not in cusip_ticker_map:
            cusip_ticker_map[cusip] = list(month_cusip_ticker[cusip])
        else:
            cusip_ticker_map[cusip] += list(month_cusip_ticker[cusip])

        cusip_ticker_map[cusip] = list(set(cusip_ticker_map[cusip]))

cusip_ticker_map = {k[:-1]: v[0] for k, v in cusip_ticker_map.items()}

# Get timeframe and tickers to pull data for from input dictionary
start_date, end_date = None, None
cusip_to_query = []
```

```
for month in cusip_month:
   if start_date is None:
      start_date, end_date = month, month
   elif month < start_date:
      start_date = month
   elif month > end_date:
      end_date = month
   else:
      None
   cusip_to_query += [ticker[:-1] for ticker in cusip_month[month]]

cusip_to_query = list(set(cusip_to_query))
cusip_to_query = tuple(["'%s'"%str(i) for i in cusip_to_query])
cusip_to_query = ",".join(cusip_to_query)

# Query to get list of N companies with top market cap for the given month
q1 = ("select * from crsp.dsf where date between '%s' and '%s' and cusip in
    (%s)")%(start_date, end_date, cusip_to_query)
price_df_all = db.raw_sql(q1).sort_values('date')

# Add in ticker values
price_df_all['tic'] = price_df_all['cusip'].map(cusip_ticker_map)

# get data from stock events table
q2 = ("select * from crsp.dse where date between '%s' and '%s' and cusip in
    (%s)")%(start_date, end_date, cusip_to_query)
event_df_all = db.raw_sql(q2).sort_values('date')

# merge events and price data
price_event_df_all = pd.merge(price_df_all, event_df_all, how='outer',
    left_on=['cusip','date'], right_on = ['cusip','date'])
```

6.1.2 TAQ Intraday Quotes

In this subsection, we provide code used to query daily stock quotes from NASDAQ Trade and Quote (TAQ) using the Wharton Research Data Services (WRDS) Python API.

```
import wrds
import pandas as pd
import datetime
from dateutil.relativedelta import relativedelta
import cPickle

# Connect to WRDS
db = wrds.Connection()
```

```
################################################################################
#### SQL Query - Given a list of equities and timeframe, get daily price data
    ####
################################################################################
outfile = 'price_data_intraday.csv'
cusip_ticker_map = {k[:-1]: v[0] for k, v in cusip_ticker_map.items()}

# Get timeframe and tickers to pull data for from input dictionary
cusip_to_query = []
ticker_to_query = []
for month in cusip_month:
   cusip_to_query += [cusip[:-1] for cusip in cusip_month[month]]

cusip_to_query = list(set(cusip_to_query))
cusip_to_query = tuple(["'%s'"%str(i) for i in cusip_to_query])
cusip_to_query = ",".join(cusip_to_query)

# Query to get list of N companies with top market cap for the given month
q1 = ("select * from crsp.dsf where date between '%s' and '%s' and cusip in
    (%s)")%(start_date, end_date, cusip_to_query)
price_df_all = db.raw_sql(q1).sort_values('date')

# Add in ticker values
price_df_all['tic'] = price_df_all['cusip'].map(cusip_ticker_map)

# get data from stock events table
q2 = ("select * from crsp.dse where date between '%s' and '%s' and cusip in
    (%s)")%(start_date, end_date, cusip_to_query)
event_df_all = db.raw_sql(q2).sort_values('date')

# merge events and price data
price_event_df_all = pd.merge(price_df_all, event_df_all, how='outer',
    left_on=['cusip','date'], right_on = ['cusip','date'])

# Write to specified filename
price_event_df_all.to_csv(outfile, encoding='utf-8', index=False)

################################################################################
#### SQL Query - Given a list of equities and dates, get intraday price data ####
################################################################################
stock_day_map = price_event_df_all.groupby('date')['tic'].apply(lambda x:
    x.values.tolist()).to_dict()
import os.path
```

```
dates = sorted(stock_day_map.keys())
for date in dates:
   tickers = tuple(set([stock for stock in stock_day_map[date] if type(stock) is
       unicode]))
   print(date, len(tickers))
   day, month, year = str(date.day), str(date.month), str(date.year)
   if len(day) == 1: day = "0" + day
   if len(month) == 1: month = "0" + month
   if not os.path.isfile("intraday_%s%s%s.csv"%(year,month,day)):
      curr_table = "taqm_%s.nbbom_%s%s%s"%(year, year, month, day)
      x = db.raw_sql("""
          SELECT date_trunc(\'minute\', time_m) as time, sym_root, (AVG(best_bid)
              + AVG(best_ask)) / 2 as price
          FROM """ + curr_table + """ where sym_root in %(syms)s and
              time_m between '09:30:00.0' and '16:00:00.0' group by sym_root, time
      """, params={"syms": tickers})

      x.to_csv("intraday_%s%s%s.csv"%(year,month,day), encoding='utf-8',
          index=False)
```