Pairs Trading Algorithm

A. Jackson, D. Sauta, I. Bergl, J. Bonadeo

QFin UWA

A Project Completed by the Trading Team in conjunction with

The University of Western Australia in 2021

Quantitative Finance UWA

Chapter 1

Choosing a Pair

1.1 Background

Our task was to build a pairs trading algorithm from S&P 500 constituent companies which is profitable under recent market conditions. This project was completed on the Quant Connect platform.

Pairs trading is a type of statistical arbitrage based on historical correlation between two or more securities. The primary idea is to wait until some unusual degree of deviation in the price of the two assets and then take alternate positions in each asset by going long in the under-performing and short in the outperforming. If historical correlation has any predictive power, one would then expect mean reversion and profits derive from the re-convergence of price.

1.2 Co-integration

Although correlation may seem like a good proxy for this strategy, consider the case where both stocks are moving in the same direction, but one has a higher growth rate. Given that we are taking a long position in one and a short position in the other, our strategy will suffer losses as a result.

We are more interested in the co-integration of the time series in question. Co-integration is a property of two of more time series variables and measures the consistency of the spread between these variables. Co-integrated time variables have a relatively stationary difference in the long term and are revert to this mean

1.3 Stock Selection

To avoid multiple comparison bias, we decided to compare stocks from the resources industry to identify our pairs. Our reasoning behind this was that the resources industry has a strong underlying economic factor which ensured that their prices moved together- namely the underlying commodity price.

After comparing the resource industry stocks through a co-integration matrix on a small time scale, we then retested co-integration on a larger time scale to make ensure evidence of longer term co-integration.

One can replicate the co-integration matrix using the following code snippet, where the pval variable represents the p values associated with the two stocks.

```
import seaborn
pmatrix = seaborn.heatmap(pvalues, xticklabels=res, yticklabels=res, cmap='RdYlGn_r'
                , mask = (pvalues >= 0.05),
                )
pmatrix.grid(color = 'r')
```

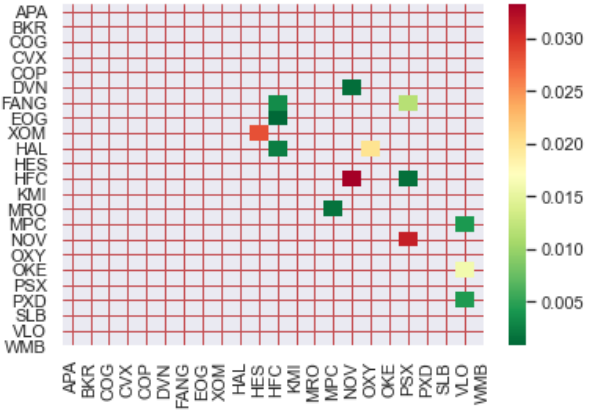

FIGURE 1.1: Co-integration matrix for resources industry

After retesting for longer term co-integration, the likely co-integrated stocks that we identified are listed below:

- HFC and HAL
- HFC and PSX
- NOV and PSX
- VLO and MPC

Chapter 2

Constructing the Algorithm

2.1 Order Sizing

For risk management purposes, we have chosen to make our pairs trading algorithm market neutral. In doing this, our strategy seeks to avoid losses as a result of systematic risk by taking a long position in one stock, and a short position in the other. One option is to take a long position and a short position in an equal number of shares however this approach does not consider how much each individual stock moves relative to the market portfolio. Alternatively, one can utilise beta neutrality instead of absolute position neutrality.

2.1.1 Beta

Beta is a measure of a stock's volatility in relation to the market portfolio. If a stock has a beta of 1, it moves in perfect correlation to the market. Stocks with high beta move greater amounts relative to the market than stocks with low beta and are considered hold more risk. Beta is a component in the Capital Asset Pricing Model and is defined using the following regression model:

$$R_{it} - R_{ft} = \alpha_i + \beta_i(R_{Mt} - R_{it}) + \varepsilon_{it}$$

Where R_{it} is the simple net return of the stock in question, R_{ft} is the risk free rate, and R_{Mt} is the simple net return of the market. Given that our strategy employs US equities, we have used the 10 year Treasury Yield as our risk free rate.

In practice, β is constantly changing, and needs to be dynamically built into the model. As such the algorithm originally calculated beta for the most recent 30 day period prior

to initialisation. Additionally, in recent market conditions R_f has been very close to 0. Due to this, we approximated the risk free rate as 0 and assumed that excess return was equal to simple net return.

We initially implemented beta neutral sizing, but due to differences in time scale between our beta calculation and algorithm, we decided to exclude this feature. The primary issue with our beta calculation was that on the short time scale we were using, comparing the stock's individual excess returns on the market to its return was rather insignificant. Beta is a better proxy for systematic risk at a larger time scale.

As such our final algorithm uses 1 to -1 position sizing.

2.2 Kalman Filter

For our mean reversion strategy we constructed a synthetic spread series based on the following formula.

$$Spread_t = log(A_t) - \beta_0 * log(B_t)$$

Where $Spread_t$ is the value of the spread series at time t, A_t and B_t are the prices of Stocks A and B at time t and and B_0 is the slope of a linear regression of the log of stock B with respect to A. However while we can assume that the long term behaviour of B_0 remains constant, this may not be true for the short term behaviour. This motivates the use of a Kalman Filter, a method of estimating the hedge ratio by updating its best guess based on new data. We can hence use it to produce a spread series that is dynamically updating with new data, as opposed to being trained on past data and remaining static.

A Kalman filter makes the assumption that a hidden variable, in our case the hedge ratio β_0, is related to some observable variable - the stock prices A and B, and each has a normally distributed amount of error associated with it. By providing the following matrix equations to the Kalman Filter, the spread series can be formulated.

$$(\beta_0, \beta_1)_t = I(\beta_0, \beta_1)_{t-1} + w_t$$

$$A_t = \begin{pmatrix} 1 \\ B_t \end{pmatrix} (\beta_0, \beta_1)_t + v_t$$

Where w_t and v_t are from a normally distributed amount of noise, and I is the 2×2 Identity Matrix.

To implement the Kalman Filter we unitised the pykalman python library. Below is our implementation, where x and y are the log of our price series over a 3 day period.

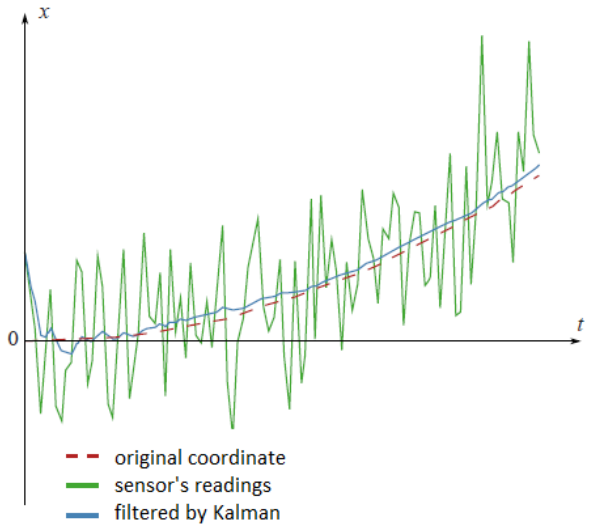

FIGURE 2.1: Example of a Kalman Filter on Some Time Series Data

kf.filer(y) produces a tuple of the average state means (the slope and intercept) and a co-variance which is disregarded.

```
delta = 1e-5
trans_cov = delta / (1 - delta) * np.eye(2)
obs_mat = np.transpose(np.vstack([x, np.ones(x.shape[0])])).reshape(-1, 1, 2)
kf = KalmanFilter(n_dim_obs=1, n_dim_state=2, initial_state_mean=np.ones(2),
                  initial_state_covariance=np.ones((2, 2)),
                  transition_matrices=trans_cov,
                  observation_matrices=obs_mat, observation_covariance=1,
                  transition_covariance=np.eye(2))
state_means, _ = kf.filter(y)
```

LISTING 2.1: Kalman Filter Method Implementation

Here 'delta' represents the desired speed of convergence to the true value - how much we treat new data as the true value. Changing this value had little effect on return.