Ornstein Uhlenbeck Process Statistical Arbitrage on Cointegrated Equity Pairs and Application in Quantitative Finance

by Filip Toth

A Mathematical Exploration for IB Math: Applications & Interpretations HL

Contents

1 Brownian Motion and the Wiener Process

Consider a stochastic process, where per each iteration, we add a Δ to the previous value, this Δ is distributed by a normal distribution with a *mu* of zero, and these are independent stochastic events this is called the Wiener process. In discrete time, we can define the Wiener process with the following equation

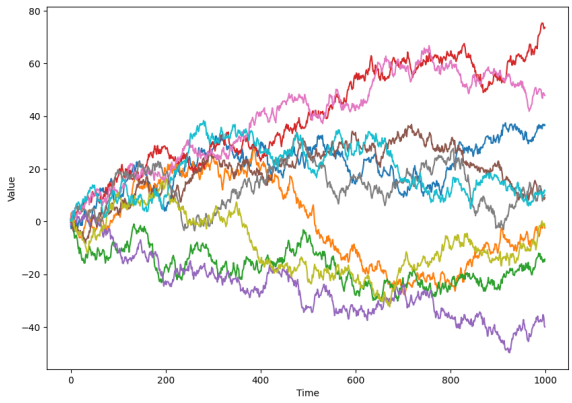
$$W_{t+1} = W_t + \mathcal{N}(0, \sigma^2)$$

Where W_{t+1} is the next value of the process (under discrete time), $\mathcal{N}(0, \sigma^2)$ is a normally distributed drift variable with the variance σ^2.

Intuitively, the expected value, i.e. the limit of this process as it approaches infinity, should be zero, since the normal distribution is symmetric. (maybe prove this? Using a limits on the $\mathbb{E}[W]$ and the normal dist)

The spread of the Wiener process grows at a rate of $\sigma_t = \sigma\sqrt{t}$, due to each timestep adding additional compounded uncertainty.

Here's an example of a Wiener process with $W_0 = 0$ and $\sigma^2 = 1$:



In the diagram, we can clearly observe that the the variance of the Wiener process gets larger as the time progresses.

The wiener process is crucial and has many financial applications, for example, equity prices under the assumption of the efficient market hypothesis follow a purely-stochastic Wiener process. This although disregards stochastic drift, which applies directional pressure on equity

prices, i.e., the Wiener process is market-agnostic, and applies to equity which do not follow the upward trend of the market due to the positive risk-free rate. Non-market agnostic processes typically leverage geometric brownian motion.

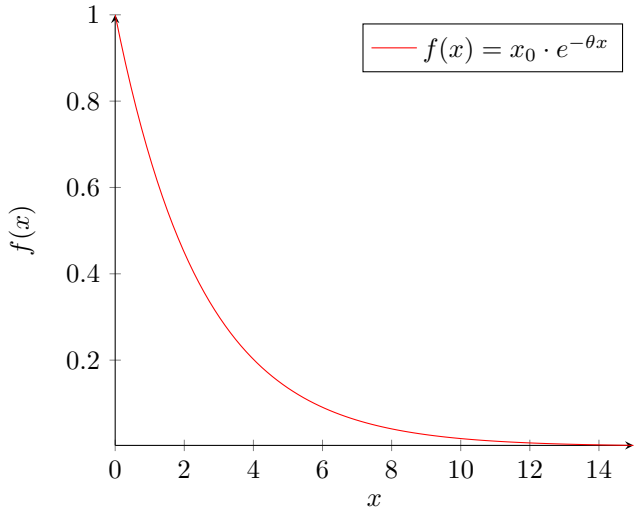
2 Ornstein-Uhlenbeck Process

2.1 Deriving a Mean Reversion Mechanism

But what if we don't want the variance of the process to get larger as time progresses? We call these processes mean-reverting, as in they tend to revert their values back towards the mean μ. This has numerous financial applications (not to mention its applications outside of finance), for example: statstical arbitrage, interest rate models, heston model, and mean reverting equities.

So how do we implement mean reversion? We can start by thinking about what mean reversion actually is? Mean reversion is the tendency of a variable (or usually a stochastic process) to revert back to its long-term mean. If we consider the roots of brownian motion - physics -, we can consider a mean-reverting term to be similar to a force pushing a particle back towards its mean; note that this doesn't happen instantaneuously, it happens over a period of time, where the steadyness of this decay is dictated by the rate of mean-reversion, denoted as θ.

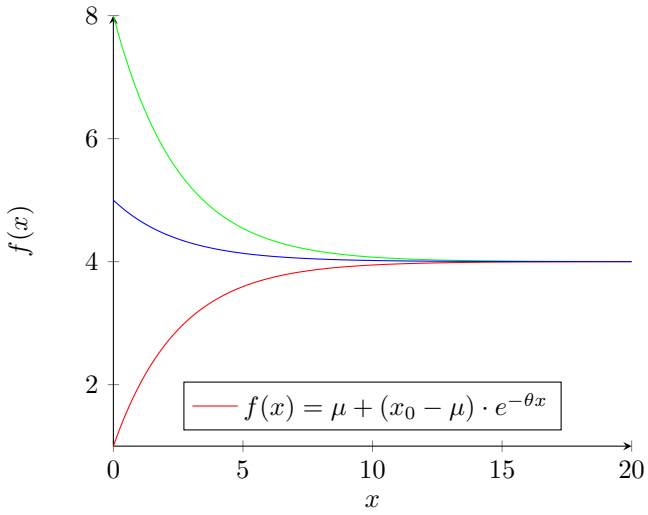
We know that an exponential function with a negative coefficient applied to the exponent will produce a function that will slowly decay.



But this is not exactly mean reverting, we need to adjust the function to not only have the tendency to approach zero, but rather, the function must approach the mean. And the function must be applied from any starting point x_0. Considering that $f(0) = x_0$, we can adjust the function and introduce a base term: $f(x) = x_0 e^{-\theta x}$, this ensures the initial value is x_0, since $e^0 = 1$. Now we have to ensure that the function converges at the mean, not at zero. Initially, we can try to add a the mean as a constant term, such that $f(x) = \mu + x_0 e^{-\theta x}$, but when graphing this function, we notice that we break our first condition - the initial value of the function $f(0)$ is not equal to x_0. We have to adjust for this in the coefficient of the exponential term. We notice that the difference between x_0 and $f(0)$ is equal to the mean, thus modifying the function to the following will solve our problem:

$$f(x) = \mu + (x_0 - \theta)e^{-\theta x}$$

The function produces the following graph, assuming $\mu = 4$, $\theta = 0.4$ and varying the initial value x_0 to be $x_0 \in \{8, 5, -1\}$ for each of the separate lines respectivelly:



2.2 Differentiating for the Mean-reverting Force

We can now think about creating a stochastic process that will take the Wiener process as a basis, but add mean-reverting properties in order to ensure the variance stays within a certain range.

Just adding a term for the brownian motion will not be enough, this would be misinterpreting the mean-reverting effect and would be mostly useless for our purposses, thus we can't do something like:

$$X_t = \mu + (X_0 - \mu) \cdot e^{-\theta t} + \sigma W_t$$

We must now calculate the tug or the actual amount by which the function changes per unit t. This will be necessary for our stochastic process, as they are inherently defined in differential form (in descrete time), where the next value depends on the previous value and we simply add a modifying term, e.g. $M_{t+1} = M_t + \sigma$. We can do this by taking a simple derivative of the function. This derivative can then be thought of as the per-unit t amount of tug applied for each value above or below the mean.

$$\frac{d}{dt}\left(\mu + (X_0 - \mu) \cdot e^{-\theta t}\right)$$
$$\frac{d}{dt}\mu + \frac{d}{dt}\left(X_0 - \mu\right) e^{-\theta t}$$

We know that the derivative of the constant μ is zero, thus we can eliminate the term. We are left with the derivative of a product, for which we can use the product rule of differentiation: $\frac{d}{dt} x \cdot y = x'y + xy'$. In our case, the two terms are $(X_0 - \mu)$ and $e^{-\theta t}$. The first term is a constant that does not chage with respect to t, thus it will be zero. The second term can be differentiated using the chain rule.

We need to define an outer and an inner function in order to apply the chain rule, the inner function will be e^x and the outer function will be $-\theta t$. The $\frac{d}{dx} e^x = e^x$ is a standard differential. And the $\frac{d}{dx} - \theta x$ will simply collapse down to $-\theta$, since it's a constant that is applied to our differentiating variable. The chain rule states:

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$
$$\frac{d}{dt} e^{-\theta x} = e^{-\theta x} \cdot (-\theta) = -\theta e^{-\theta x}$$

We can now revisit the prudct rule and apply it, we know that the derivative of our first term is zero, thus we can ignore the first term of the product rule. Only leaving the following: $\frac{d}{dt}x \cdot y = xy'$, we have already calculated $y' = -\theta e^{-\theta x}$ in the previous step. Thus we can Just multiply $-\theta e^{-\theta x}$ with $(X_0 - \mu)$, and we get the following result for our whole derivative:

$$\frac{d}{dt}\left(\mu + (X_0 - \mu) \cdot e^{-\theta t}\right) = (-\theta e^{-\theta x}) \cdot (X_0 - \mu)$$

We can rearrange this to:

$$\frac{d}{dt}\left(\mu + (X_0 - \mu) \cdot e^{-\theta t}\right) = -\theta(X_0 - \mu)e^{-\theta t}$$

This differential desribes the mean-reverting tug applied to our brownian motion model.

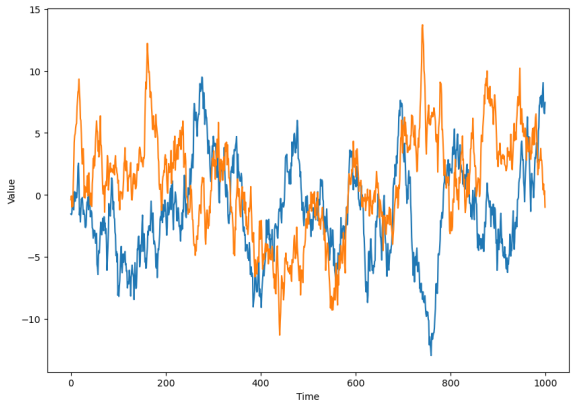
2.3 Full Naive Derivation of the OU Process

We can now try to perform a few experiments on a naively-formulated OU (Ornstein Uhlenbeck) process. We can begin by simply adding a brownian motion to our stochastic process, and assuming (naive step) $X_0 = X_t$, since at each point, the differential will get us tug, this will mean that our t is now zero. We are left with the following stochastic process:

$$X_{t+1} = -\theta(X_t - \mu)e^{-\theta \cdot 0} + W_t$$

$$X_{t+1} = -\theta(X_t - \mu)e + \mathcal{N}(0, \sigma^2)$$

We can not create stochastic paths along the process and they should be mean reverting. A stochastic path (more formally called a sample path of a stochastic process) is simply a possible outcome of running the process, esentially a random path that conforms to the process.



Here we can see that the stochastic process is much more contained within a range, and is thus mean-reverting.

2.4 Formal Derivation of the OU Process

The Ornstein Uhlenbeck process is formally defined as a stochastic differential equation (SDE). An SDE is just a differential equation with a stochastic term.

We begin with adding a brownian motion component. A standard wiener process in its differential form is: $dX_t = dW_t$, this is pretty straight forward if you think about it, the change in X_t is just the change in W_t, this being a stochastic component, dW_t will always be different, and for the sake of simplicity, we can visualize this as if the term follows the normal distribution $dW_t \sim \mathcal{N}(0, \sigma^2)$, this isn't entire true because the variance of the normal distribution itself depends on dt, but we will get to that later.

We now add the drift term (which we derived in a previous subchapter) $\theta(\mu - X_t)$ (we derived it as $-\theta(X_t - \mu)$, but these are functionally equivalent), we can say that this drift happens - continuously - over time t, thus can be written as $\theta(\mu - X_t)dt$, recall that dt here represents an infinitesmal (very small) change in time t, thus the whole term represents the change in the process value X_t as we change time t by an infinitesmal amount and can thus be written as $\frac{dX_t}{dt} = \theta(\mu - X_t)$, but this is just basic differential equations. We end up with:

$$dX_t = \theta(\mu - X_t)dt + dW_t$$

We also add a variable σ to our process, not to be confused with the standard deviation, in our context, θ represents the intensity of the brownian motion (stochastic) component. We end up with the final formal definition of the OU process:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

3 Long-term Equilibrium Cointegration in Equity Markets

We perform the **Two Step Engle-Granger Test** for cointegration of time series to get the cointegration coefficient. This method is widely accepted in quantitative finance and econometrics.

But what is cointegration? Cointegration represents the level to which a linear combination of two (or more) time series $X_t = \beta Y_t$ follows a stationary time series $I(0)$. Or in other words, whether the linear combination has a zero long-term mean. We can take any linear combination, such as the linear regression of X_t on Y_t, in the next sub-chapter, we form this regression and add an error term, the error term is esentially formed from a linear combination of the two time series. A high cointegration coefficient means that the two time series move in tandem with each other and that deviations in their spread revert to the long-term mean (since the mean of the spread is zero), divergence of spreads resulting in a zero long-term mean spread is impossible because the function is continuous.

3.1 Static Regression of Equity Time Series

First, we need to determine the residuals $\hat{\epsilon}_t$ (similar to the error) of a regression between the two time series of the prices of the equities we're analyzing. We define a linear regression $Y_t = \alpha + \beta X_t + \epsilon_t$. Where Y_t and X_t are the t-th elements of the time series of the equity prices of X and Y. And where α is the intercept term (which is the mean level of Y_t where X_t is zero) and where β is mean slope coefficient, i.e. the change in the value of Y_t per unit X_t. We define

the following intuitively or from the definition of the linear regression:

$$\beta = \frac{\sum_{i=0}^{n}(X_i - \mu_X)\cdot(Y_i - \mu_Y)}{\sum_{x\in X}(x-\mu_X)^2}$$

$$\alpha = \mu_Y - \beta \cdot \mu_X$$

Now we calculate the residuals $\hat{\epsilon}_t$ using the formula $\hat{\epsilon}_t = Y_t - (\alpha + \beta X_t)$, which we obtain from rearranging the terms of the linear regression equation with resduals. We can interpret these residuals as the difference (for each timestep t) between the actual value X_t and the predicted value - the value of the regression solution for the parameter t.

3.2 Dickey Fuller Test

The Dickey Fuller Test or (DF Test for short) is a statistical test that tests for the stationarity of a time series, or in other words, it tests for if the order of integration of the time series is $I(0)$. The non-augmented version of the Dickey Fuller Test is only applicable for an auto-regressive AR(1) process. Equity spreads are widely considered to be AR(1) processes. We can define an AR(1) time series as $y_t = \alpha + \phi y_{t-1} + \epsilon_t$, where α is the constant shift term, ϕy_{t-1} is the autoregressive term where ϕ is a coefficient denoting how strongly the previous value of the time series affects the current value at time t, and ϵ_t (not to be confused with the residuals from our regression model) denotes a random error term, which IS assumed to be white noise $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. Now we can define the hypotheses for the DF Test.

$$H_0 : \phi = 1$$

$$H_1 : \phi < 1$$

Where the null hypothesis denotes that the process is non-stationary and the alternative hypothesis indicates that the process is stationary. We could simply use a t-test to see if a $\hat{\phi}$ estimator is statistically significant, and based on that reject or fail to reject the null hypothesis of the DF Test. But the t-test is inapplicable in models where the central limit theorem doesn't

hold true (because the ϵ values of the process are not independent and identically distributed), thus we need to use the DF test. First, we transform our original time series expression in as a differencing equation. Note that we also rewrite the $y_t - y_{t-1}$ as Δy_t.

$$y_t = \alpha + \phi y_{t-1} + \epsilon_t$$

$$y_t - y_{t-1} = \alpha + \phi y_{t-1} - y_{t-1} + \epsilon_t$$

$$\Delta y_t = \alpha + y_{t-1}(\phi - 1) + \epsilon_t$$

For the sake of simplicity, we introduce δ instead of $(\phi - 1)$. We can simplify our express to $\Delta y_t = \alpha + \delta y_{t-1} + \epsilon_t$, which is the standard form you'll find in most literature on the DF Test. Our hypotheses remain and can be rewritten in terms of δ instead of ϕ.

$$H_0 : \delta = 0$$

$$H_1 : \delta < 0$$

We apply an Ordinary Least Squares linear regression to find the estimator $\hat{\delta}$, this is the p-value for our test and we compare it against our hypotheses. But in our case, we this won't be rigorous enough since we didn't account for autocorrelation (In our case, this means that previous values have some sort of effect on the current value y_t in our time series) properties of financial time series, to remedy this and correctly account for autocorrelation, we use the Augmented Dickey Fuller Test or ADF for short.

3.3 Augmented Dickey Fuller Test

This is an extension of the Dickey Fuller Test that correctly accounts for autocorrelation between consecutive values of the time series and is also extended to the autoregressive AR(p) process; it tests whether a time series of the order of integration $I(p)$ is stationary and for the presence of a unit root (which is a prerequisite of non-stationarity). In the ADF test, our hypotheses stay

the same.

$$H_0 : \delta = 0 \rightarrow \text{non-stationary, unit root}$$

$$H_1 : \delta < 0 \rightarrow \text{stationary, no unit root}$$

We need to add lags to our expression to account for autocorrelation. We can model autocorrelation as some coefficient applied on some previous value of the time series also being accounted in the value of the time series at t. Considering we add two lags to account for autocorrelation between the current value y_t and values y_{t-1} and y_{t-2}, our expression will intuitively become $\Delta y_t = \alpha + y_{t-1}(\phi - 1) + \beta_1(\Delta y_{t-1}) + \beta_2(\Delta y_{t-2}) + \epsilon_t$. Where the betas β_1 are our coefficients onf the lagged differences. We also define p to be the number of lags we include, this value is automatically optimized in various statistical application programming interfaces (APIs) and one can use various Bayesian estimation methods to get the optimal p value, or one can perform a t-test (or an F-test when testing for the significance of multiple variables) and keep adding lagged differences (thus increasing p) until the addition is no longer statistically significant. Our expression can be generalized in the following form:

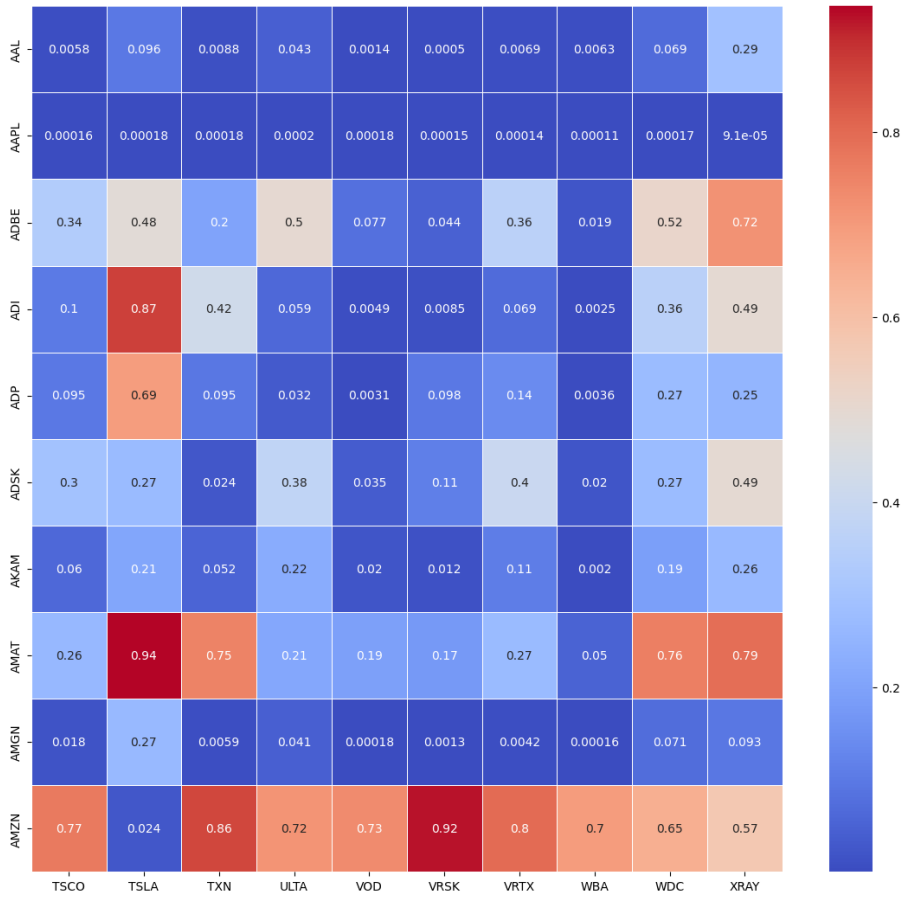
$$\Delta y_t = \alpha\delta y_{t-1} + \sum_{i=1}^{p} \beta_i \Delta y_{t-1} + \epsilon_t$$

Now that we have this equation, we can do an Ordinary Least Squares regression (in its multiple linear regression form, since we have multiple regressors) to find $\alpha, \delta, \beta_1, \beta_2, \ldots, \beta_p$, we then use the δ value as the final p-value for the Augmented Dickey Fuller Test.

3.4 Performing the Two-Step Engle Granger Test on Equity Prices

We now perform the two-step engle granger test on 78 mature stocks from the selected from NADSAQ-100 index from Janurary 1st 2014 to January 1st 2024. We pair every stock with every other stock resulting in $78^2 = 6084$ pairs, for each of these pairs we perform an OLS regression and get the residual values (as described in subsection 1) and then perform the Augmented

Dickey Fuller Test using the 'statsmodels' library. We extract the p-values of the test and end up with the following heatmap:



Only 100 pairs are displayed as the complete 78x78 heatmap is too large to present here.

We now select 10 of the most cointegrated (lowest p-values) equity pairs to further examine, the ten most cointegrated pairs were, in descending order of cointegration, we observe that most of the equity pairs with very heavy cointegration have AAPL (Apple Inc.) as the base stock, this is interesting as AAPL has heavily outperformed even the growth-heavy NASDAQ, thus this should create a larger spread between most other stocks, suggesting limitations of our test.

Stock 1 Ticker	Stock 2 Ticker	ADF P-Value
AAPL	XRAY	9.104296439449678e-05
AAPL	WBA	0.00011254543577701014
AAPL	VRTX	0.00014280647279306093
AAPL	VRSK	0.00015429781044445213
AMGN	WBA	0.00016469559752946386
AAPL	TSCO	0.00016489122300362627
AAPL	WDC	0.0001652537550668745
AAPL	TSLA	0.00017722866444703876
AMGN	VOD	0.00018081206670114813
AAPL	TXN	0.00018223283943511726

For reference, we can plot the residuals of the initial regression of a select heavily cointegrated pair and a non-cointegrated pair to better visualize the idea of cointegration.

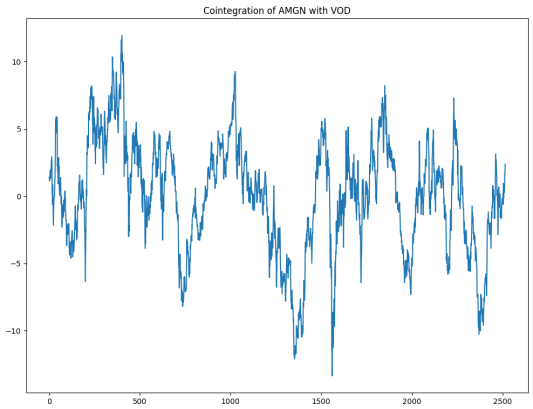


Figure 1: AMGN, VOD, p-value: 0.000181

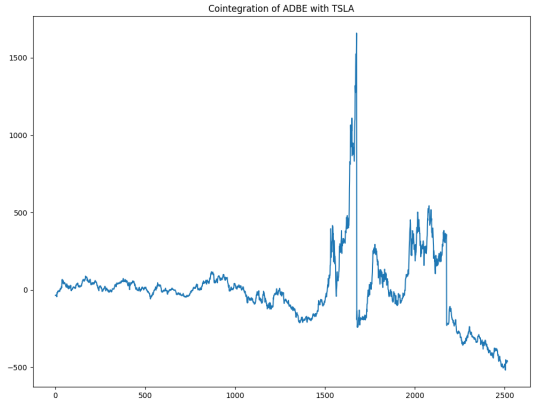


Figure 2: ADBE, TSLA, p-value: 0.484251

3.5 Engle Granger Residuals vs Equity Spreads

If we plot the residuals of AMGN and VOD, we can observe that the resultant time series clearly follows an OU process (or a time-discrete AR(1) time series), as is also indicated by the test we performed, but if we plot the equity spreads (values of stock 1 minues values of stock 2), we

observe a different pattern, indicating that our model is still incomplete and that OLS residuals might not give us the full picture. Let's examine the difference between the graph of the residuals and the graph of the spreads:

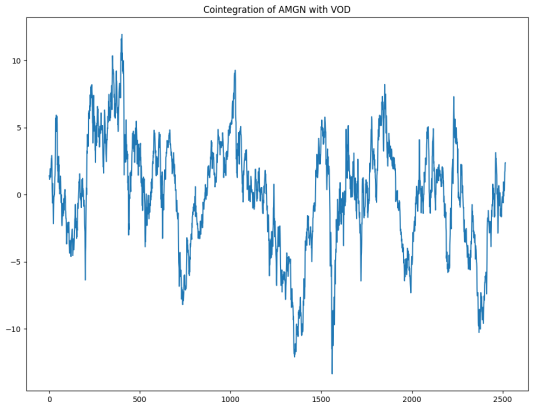


Figure 3: AMGN-VOD cointegration regression residuals

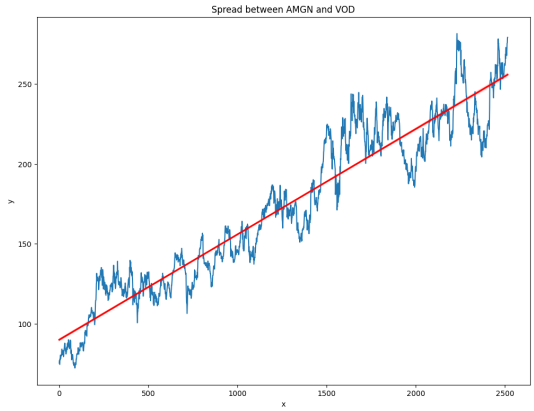


Figure 4: AMGN-VOD spread with line of best fit

We can see that the cointegration esentially disregards deterministic drift in the spreads (deterministic drift is esentially the non-stochastic trend). This makes sense if you think about our test methodology, we defined an OLS linear regression between two stock price time series and then we used to residuals to perform the stationarity test, our graph just shows the residuals at each time step. A residual is just the difference between the what the regression model predicts (using a standard linear equation model $y = \beta x + \alpha$) and the actual value of the time series.

Our Ornstein Uhlenbeck model resembles the residuals and not the actual spreads, it doesn't yet account for deterministic drift.

4 Fitting the Ornstein Uhlenbeck Process to our Data

There are two ways you can calibrate an Ornstein Uhlenbeck process for a particular dataset. Either using a simple OLS regression for the OU parameters (θ, σ, μ), or you can use a much more rigorous technique that yields better results, called: Maximum Likelyhood Estimation (MLE), while if this weren't a Math IA, I would use MLE (due to there already being pre-

existing implementation and it being fairly easy to understand, just requires basic multivariable calculus), we do not have the capacity to explain MLE in this IA, additionally, OLS parameter estimation is frequently used in Quant Finance, despite it's obvious limitations. The first step is obtaining a formal solution to the OU stochastic differential equation (SDE).

4.1 Formal Analytical Solution to the OU SDE

This might not be in the final IA as I've found a way to do this with just the discrete-time solution, making this section obsolete. Might still have to use it when optimizing the tresholds (discussed later)

We start with the OU SDE we derived in a previously, and we expand the deterministic drift term.

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

$$dX_t = \theta\mu dt - \theta X_t dt + \sigma dW_t$$

Looking at the deterministic part of our SDE $dX_t = \theta\mu dt - \theta X_t dt$, we notice that we need to simplify this ordinary differential equation (ODE) and find an integrating factor. An integrating dactor is given by the following: $I(x) = e^{\int P(x)\,dx}$, where $P(x)$ is defined by the standard form of a first-order ODE: $\frac{dx}{dy} + P(x)y = Q(x)$, which we can fit to our deterministic ODE:

$$\frac{dX_t}{dt} = \theta\mu - \theta X_t$$
$$\frac{dX_t}{dt} + \theta X_t = \theta\mu$$

where $P(x) = \theta$ and $Q(x) = \theta\mu$, we can now use the above-mentioned formula to find the integrating factor, $I(t) = e^{\int P(t)\,dx}$ (notice that we switch to t since dt is our differential and not dx). Subce θ is a constant, we can easily evaluate the integral $\int \theta\,dt = \theta t + C$, thus our integrating factor becomes: $I(t) = e^{\theta t}$, notice that we deliberately ignore the constant term C.

Now, going back to our SDE, we can apply our integrating factor.

$$dX_t = \theta\mu dt - \theta X_t dt + \sigma dW_t$$

$$e^{\theta t} dX_t = e^{\theta t}\theta\mu dt - e^{\theta t}\theta X_t dt + e^{\theta t}\sigma dW_t$$

We now notice that we can use the product rule of differentiation to simplify the above SDE. If we rearrange the SDE, we notice that $e^{\theta t}dX_t + e^{\theta t}\theta X_t dt$ conforms to the product rule, that states $(u \cdot v)' = u \cdot v' + u' \cdot v$. We aim to express our two terms as a single stochastic derivative, we can't simply use a regular derivative since X_t is a stochastic process, thus we introduce the Itô's derivative, which is a derivative on a stochastic process, and in our case $d(\cdot)$ represents an infinitesmal change in the stochasric process changes with respect to time t, this can be expanded with the use of the Itô's Lemma. Going back to solving our SDE, we work backwards through the application of the product rule and see that $u = e^{\theta t}$ and $v = X_t$, our terms will simplify to:

$$d\left(e^{\theta t} \cdot X_t\right) = e^{\theta t} dX_t + e^{\theta t}\theta X_t dt$$

And we can apply this to the SDE, like so:

$$e^{\theta t} dX_t = e^{\theta t}\theta\mu dt - e^{\theta t}\theta X_t dt + e^{\theta t}\sigma dW_t$$

$$e^{\theta t} dX_t + e^{\theta t}\theta X_t dt = e^{\theta t}\theta\mu dt + e^{\theta t}\sigma dW_t$$

$$d\left(e^{\theta t} \cdot X_t\right) = e^{\theta t}\theta\mu dt + e^{\theta t}\sigma dW_t$$

Now we can integrate to find the general solution to the linear stochastic differential equation. Note that for our stochastic components, we can't perform the regular Riemann Integral, but rather, we have to use the Itô Integral. We define an Itô Integral as an integral on a stochastic process, the result of the Itô Integral is another stochastic process, that esentially defines the "sum" of the process up until a certain time t, it can again, be solved by application of the Itô's

Lemma. We define a general Itô Integral as:

$$\int_0^t f(s)dX_s$$

Where $f(s)$ is a deterministic process, which is generated from a filtration $\mathcal{F}(s)$, which is just a collection of all known information about the process up until a time s. You can think of $f(s)$ as just a deterministic process that describes how the stochastic process behaves over time s. And dX_t is, of course, is an infinitesmal change in the stochastic process X_t. Let's take our brownian motion term as an example, $\int_0^t e^{\theta t}\sigma dW_s$, our deterministic process generated by the filtration $\mathcal{F}(s)$ would be $f(s) = e^{\theta t}\sigma$, note that this is constant and not dependent on time s. We can now applt the Itô Integral on the left-hand side of the SDE and on the stochastic Wiener Process term.

$$d\left(e^{\theta t}\cdot X_t\right) = e^{\theta t}\theta\mu dt + e^{\theta t}\sigma dW_t$$
$$\int_0^t d\left(e^{\theta s}\cdot X_s\right) = \int_0^t e^{\theta s}\theta\mu\, ds + \int_0^t e^{\theta s}\sigma dW_s$$

We can now evaluate each of these integrals. We begin by evaluating the deterministic integral in the middle of the SDE. We notice that $\theta\mu$ is a constants term, thus a part of it can be factored out of the integral. We also recall the rule for integrating exponental functions $\int e^{ax}\,dx = \frac{1}{a}e^{ax} + C$ and apply it.

$$\int_0^t e^{\theta s}\theta\mu\, ds = \mu\int_0^t e^{\theta s}\theta\, ds = \mu\Big|_0^t \theta\frac{1}{\theta}e^{\theta t} = \mu\Big|_0^t e^{\theta t}$$

Next we evaluate the integral from 0 to time s, using the standard procedure for evaluating definite integrals.

$$\mu\Big|_0^t e^{\theta t} = \mu\left(e^{\theta t} - e^{\theta\cdot 0}\right) = \mu\left(e^{\theta t} - 1\right)$$

Next, we need to solve two of our Itô Integrals. Let's solve the left-hand side of our SDE first.

We will apply the fundemental theorem of calculus adapted to Itô integrals, which states that for a well-behaved process Y_t, the Itô Integral of the differential of that process dY_t is described by: $\int_0^t dY_s = Y_t - Y_0$, we recoggnizee that $Y_s = e^{\theta s} X_s$, thus we can apply it to our left-hand side:

$$\int_0^t d\left(e^{\theta s} \cdot X_s\right) = e^{\theta t} X_t - e^{\theta \cdot 0} X_0 = e^{\theta t} X_t - X_0$$

We cannot further simplify our stochastic integral over the Wiener Process, thus we leave it as is - which is fine, in our algorithmic trading strategy this will be fairly simple to compute and correctly account for. We combine everything and get the following:

$$\int_0^t d\left(e^{\theta s} \cdot X_s\right) = \int_0^t e^{\theta s} \theta \mu \, ds + \int_0^t e^{\theta s} \sigma dW_s$$
$$e^{\theta t} X_t - X_0 = \mu(e^{\theta t} - 1) + \int_0^t e^{\theta s} \sigma dW_s$$

Next, we express the term X_t which represents the value of the Ornstein Uhlenbeck process at time t, which is the final solution to the SDE.

$$e^{\theta t} X_t = X_0 + \mu(e^{\theta t} - 1) + \int_0^t e^{\theta s} \sigma dW_s$$
$$X_t = \frac{1}{e^{\theta t}} X_0 + \frac{1}{e^{\theta t}} \mu(e^{\theta t} - 1) + \frac{1}{e^{\theta t}} \int_0^t e^{\theta s} \sigma dW_s$$

We can now simplify each of these terms. $\frac{1}{e^{\theta t}} X_0$ will become $X_0 e^{-\theta t}$. We continue simplifying, $\frac{1}{e^{\theta t}} \mu(e^{\theta t} - 1) = \frac{1}{e^{\theta t}} \left(\mu e^{\theta t} - \mu\right) = \left(\mu - \mu e^{-\theta t}\right) = \mu\left(1 - e^{-\theta t}\right)$. We recall the rule about factoring constants out of integrals, we can also do the reverse and factor a constant into an integral, while this is contraproductive most of the time, it can be used to simplify certain terms. Thus, $\frac{1}{e^{\theta t}} \int_0^t e^{\theta s} \sigma dW_s = \int_0^t \frac{1}{e^{\theta t}} e^{\theta s} \sigma dW_s = \int_0^t e^{-\theta t} e^{\theta s} \sigma dW_s = \int_0^t e^{\theta s - \theta t} \sigma dW_s = \int_0^t e^{-\theta(t-s)} \sigma dW_s$. Now that all of our terms have been simplified, we can combine them to form the final formal solution

to the stochastic differential equation governing the Ornstein Uhlenbeck Process.

$$X_t = X_0 e^{-\theta t} + \mu \left(1 - e^{-\theta t}\right) + \int_0^t e^{-\theta(t-s)} \sigma dW_s$$

4.2 Adjusting OU Deterministic Drift for Trend

Recall the graph of the sample paths generated by the Ornstein Uhlenbeck process, you can notice that the mean is static, the process always reverts back to a constant mean. But Looking at the equity spreads between most of our cointegration pairs, we notice that the spread is actually growing, while still adhering to the Ornstein Uhlenbeck Process, we know this because we checked for the stationarity of regression residuals of the spreads. What if we simply introduce a function of time representing a changing mean? We can represent this with the following function $\mu(t)$, and we can directly substitute it to the SDE, but we still have to do some work to properly find a general solution.

$$dX_t = \theta \left(\mu(t) - X_t\right) dt + \sigma dW_t$$

We can't use the general solution we found in the previous subchapter directly, because we assumed that μ is a constant, which is not true. Previously, we defined an integral for finding the general solution to the deterministic part of the SDE and we simplified it. We can no longer do this since we would like to keep things general and assume that $\mu(t)$ is some function and we don't have any further information about it. Thus, we can change our integral over the deterministic term:

$$\int_0^t e^{\theta s} \theta \mu \, ds \Longrightarrow \int_0^t e^{\theta s} \theta \mu(s) \, ds$$

Recall that we also had to divide by our integrating factor $e^{-\theta t}$, we also have to apply the operation here.

$$e^{-\theta t}\int_0^t e^{\theta s}\theta\mu(s)\,ds = \int_0^t e^{\theta s}e^{-\theta t}\theta\mu(s)\,ds = \int_0^t e^{\theta(s-t)}\theta\mu(s)\,ds$$

We can now substitute this new integral in place of the deterministic part of the general solution to the SDE.

$$X_t = X_0e^{-\theta t} + \int_0^t e^{\theta(s-t)}\theta\mu(s)\,ds + \int_0^t e^{-\theta(t-s)}\sigma dW_s$$

4.3 Time Discretization Using the Euler Maruyama Method

We can define any Itô Process uisng the following generalization of a stochastic differenial equation:

$$dX_t = f(t, X_t)dt + g(t, X_t)dW_t$$

Where $f(t, X_t)$ represents the deterministic drift term and $g(t, X_t)$ represents the stochastic diffusion term, which you can think of as a modifier on the brownian motion. Looking at the SDE describing the Ornstein Uhlenbeck Process with a non-constant time-dependent mean $dX_t = \theta\left(\mu(t) - X_t\right)dt + \sigma dW_t$, we notice that we can formulate a $f(t, X_t)$ function and a $g(t, X_t)$ function such thate the SDE conforms to the before-mentioned generalization.

$$f(t, X_t) = \theta\left(\mu(t) - X_t\right)$$

$$g(t, X_t) = \sigma$$

Applying the Euler Maruyama Method for discretization, we get the following time-discrete approximation of the OU SDE:

$$X_{t+1} = X_t + \tau f(t, X_t) + g(t, X_t)\Delta W_n$$

$$X_{t+1} = X_t + \tau\theta\left(\mu(t) - X_t\right) + \sigma\Delta W_n$$

We define τ to be a small increment (esentially the scale of our time series X_t), we also define ΔW_t as an increment of the Wiener process over a discrete time-increment interval $[t, t+1]$, such that $\Delta W_t = W_{t-1} + W_t \sim \mathcal{N}(0, \tau)$. In our model, we consider $\tau = 1$, so we can disregard it. We end up with the following discretization:

$$X_{t+1} = X_t + \theta\left(\mu(t) - X_t\right) + \sigma\Delta W_n$$

4.4 Calibrating OU Process Parameters using OLS

We start by defining a time series X_t, which contains the underlying dataset we're trying to fit the OU process to, in our case, this would be the spread between two equities. We need to estimate the parameters θ - mean reverting force, $\mu(t)$ - time-dependent mean function, and σ - volatility of the noise.

We can easily estimate the mean function $\hat{\mu}(t) = \beta t + \alpha$ by running a linear regression on our dataset. The slope of the result of this regression is β and the intercept is α.

Next, to estimate the other parameters, we rearrange our time-discrete solution in terms of the changes between consecutive elements of the time series.

$$X_{t+1} = X_t + \theta\left(\mu(t) - X_t\right) + \sigma\Delta W_n$$

$$X_{t+1} - X_t = \theta\left(\mu(t) - X_t\right) + \sigma\Delta W_n$$

$$\Delta X_t = \theta\left(\mu(t) - X_t\right) + \sigma\Delta W_n$$

We need to fit our equation to the form $Y_t = \beta X_t + \alpha + \epsilon_t$, where ϵ_t represents the residuals of the

OLS, this is the standard form of an OLS reression which we can then feed into our algorithm. We start by preprocessing the time series X_t, to a singular form that will match the term $\theta X_t'$, we define the time series $X_t' = \mu(t) - X_t$, we will use this new transformed time series in place of X_t in our OLS regression because it matches up with the regression equation form. Thus, we defne a regression $Y_t = \beta X_t' + \epsilon_t$, where intuitively $Y_t = \Delta X_t$. Now, the only parameter left is σ, which we obtain by calculating the variance of the residuals, $\sigma^2 = \text{Var}(\epsilon_t)$, as I've mentioned previously ϵ_t is the result of the Wiener process, thus getting the variance of this time series will correspond to the variance of the Wiener process.

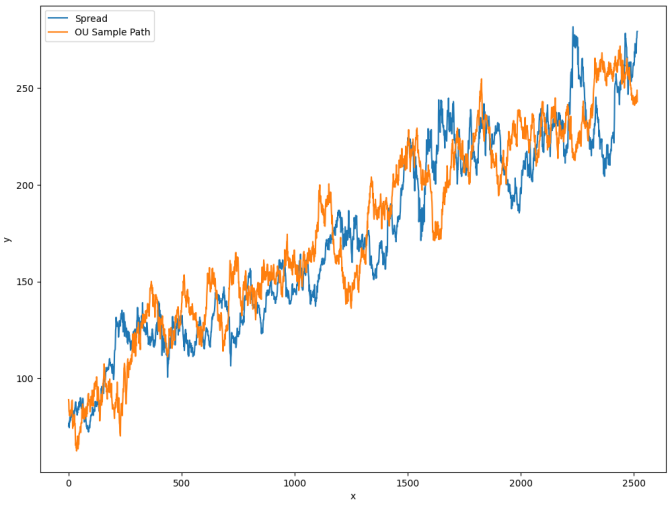
After performing the OLS (through the 'statsmodels' package) and calibrating the OU process on the time series of spreads between 'NASDAQ: AMGN' and 'NASDAQ: VOD', we get the following parameters:

$$mu(t) = 0.06594474095969069t + 89.97341936600293$$

$$\theta = 0.017667438720532103$$

$$\sigma = 2.8934948104163873$$

We can now plot a sample path along our calibrated Ornstein Uhlenbeck Process (with the above parameters) and the spreads between 'AMGN' and 'VOD', and visually confirm that they line up quite nicely:



5 Unoptimized OU-based Trading Strategy

Since we have a calibrated mean-reverting process, we can now leverage this to create an algorithmic trading strategy capturing the mean-reverting mechanism of the spread between two equities - we will look at 'AMGN' and 'VOD' for now, but this can be extended to any pair of equities (and even multiple equities through a more advanced straregy). The idea is pretty simple, bet that the spread between them will rever to its long-term mean, but how exactly do we do that?

5.1 A Bit of Background: Long Short

How exactly do we capture the spread between two equities? Consider a spread between two stocks $A_t - B_t$, if the spread increases, this means that stock A has outperformed stock B, but if the spread decreases, stock B has outperformed stock A. Thus, to bet that the spread will increase, we open a long position on stock A and open a short position on stock B, conversely, if we want to bet on the spread decreasing, we open a long position on stock B and open a short position on stock A. But what is a long and short position? A long position esentially means holding a stock, i.e. the classic way most people think about investing, you hold a stock expecting the price to increase, after which you exit your position. When short-selling a stock however, we aim to profit from a decrease in share price. The process usually goes something like this: you borrow a stock (usually from a broker), then you sell it on the open market, after the price falls, you buy back the stock and get to keep the difference (since the price has fallen), note that this is a heavily oversimplified explanation of the shorting process. It's also worthwhile to mention that shorting a security poses a lot more risk than longing a security, since it technically has unlimited downside (since the stock price can theoretically rise to infinity).

5.2 Defining our Straregy

We will backtest one simple strategy that leverage the Ornstein Uhlenbeck Process. The strategy focuses on calculating the mean reverting force $\theta(\mu(t) - X_t)$, which acts as a proxy for how far

the current spread has deviated from the long-term mean, this suggests higher pressure to revert back to its mean. Thus, when $\theta(\mu(t) - X_t)$ is over a predefined treshold, we fire a signal. Then, if the current spread is above the mean, we bet on the spread reverting, thus open a short position on A and open a long position on B. If the current spread is below the mean, we open a long position on B and a short position on A. When the price of over the treshold again (just in the opposite direction), we liquidate our long and short positions. In both of these strategies, we use a long-short position to isolate the market effect, thus become market neutral, i.e. our position will not be affected by the directional moves of the market. This is called statistical arbitrage and is one of the few strategies to generate returns that are uncorrelated to the market.

5.3 Backtest with Unoptimized Tresholds

We look at the prices of 'AMGN' and 'VOD' between January 1st 2014 and January 1st 2024. We define a training dataset (trainset) and a testing dataset (testset), with the trainset consiting of 1450 trading days and the testset of the remaining 1066 trading days of the 10 year period. The trainset will be used to fit the Ornstein Uhlenbeck Process and estimate process parameters and the testset will be used to actually perform the backtest and see how our strategy performs. We begin by fitting an OU process on our trainset, thus getting parameters such as the mean function $\mu(t) = \beta x + \alpha$, θ, and the Wiener variance σ^2. For our trainset, we find out that:

$$\mu(t) = 0.05933273399681959t + 92.3886877399938$$

$$\theta = 0.022943374462635356$$

$$\sigma = 2.3433270992078366$$

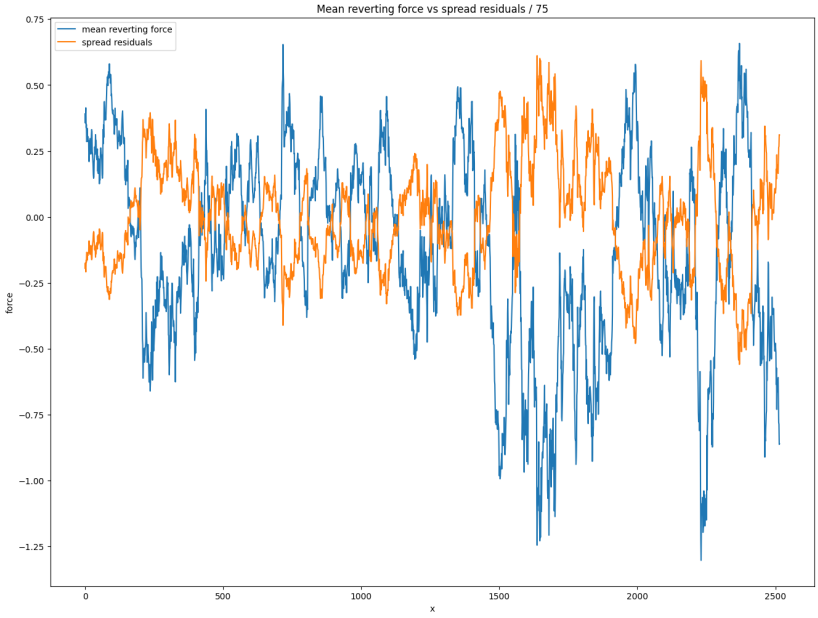
Now let's talk about how we actually perform the backtest. We iterate over our testset (go over each value of the time series sequentially, $X_t, X_{t+1}, \dots$), at each of these steps we calculate the mean-reverting force $F = \theta(\mu(t) - X_t)$. The mean reverting force is set greater than some treshold τ_{up} or lower than some treshold τ_{down}, we fire a signal. This just means that we execute our trade, if $F > \tau_{\text{up}}$, we open a long position on the spread and liquidate any short positions

on the spread. If $-F > \tau_{\text{down}}$, we open a short position on the spread and liquidate any long positions on the spread. We can write this down in the followin algorithm:

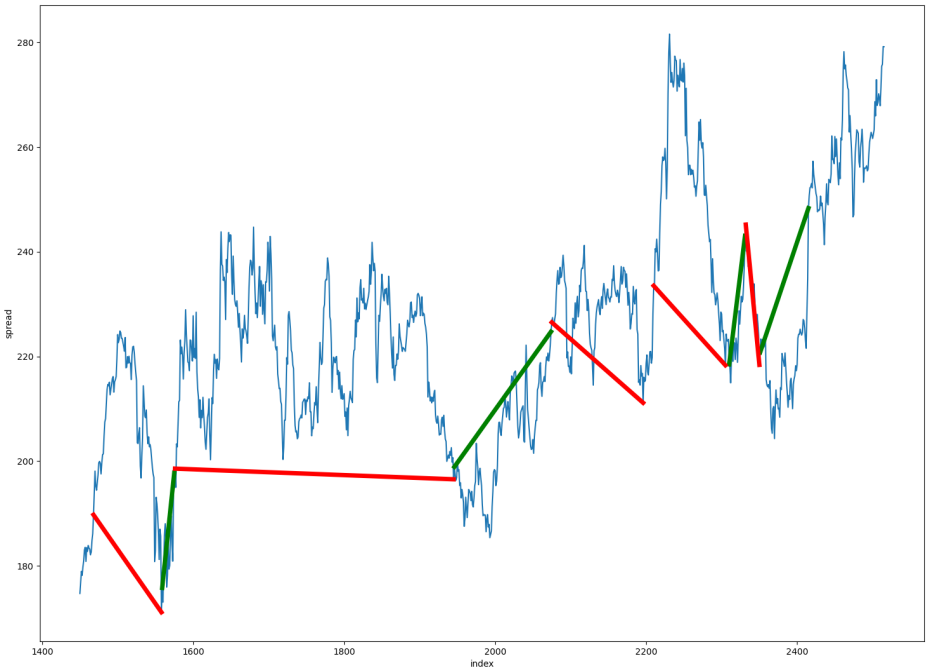
Algorithm 1 Backtesting algorithm

```
for X_t in testset do
    if F > τ_up then
        exit_short()
        open_long()
    else if −F > τ_down then
        exit_long()
        open_short()
    end if
end for
```

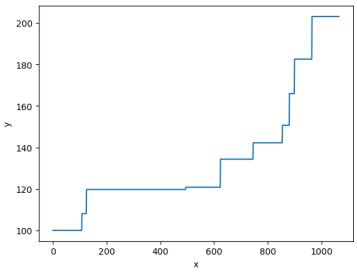
For reference, let's graph the mean reverting force $\theta(\mu(t) - X_t)$ against the quity spreads, we should see an inversely-proportional relationship, such that when the spread gets greater, the mean reverting force should decrease (into the negative) and when the spread gets into the negative, the mean reverting force should increase.



Based on this graph, we can estimate the resholds for our algorithm. As an example, I set $\tau_{\text{up}} = \tau_{\text{down}} = 0.2$, such that it would hit most of the peaks. running our backtesting algorithm, we can now plot our positions, where a red line indicates shorting the spread and a green line indicates longing the spread.



We can also plot the total portfolio value of our strategy over time (starting with a porfolio value $p = 100$).



In the end, we observe that our final portfolio value is around 203.10593563249694, which indicates a 203.10% increase in portfolio value over a time period of 1066 trading days. Given 252 trading days in a given year, we've had our portfolio for 4.23 years. Let's calulate our CAGR (Compound average growth rate).

$$\text{CAGR} = \left(\frac{V_{\text{final}}}{V_{\text{begin}}}\right)^{\frac{1}{t}} - 1 = \left(\frac{203.10}{100}\right)^{\frac{1}{4.23}} - 1 = 18.23\%$$

We can see that our CAGR is around 18.23%, which is absolutely top-of-the-line even when compared to some of the best hedge-fund standards. This is even more impressive considering that this is a statistical arbitrage strategy that isolates the market effect, thus is esentially

"recession-proof". For reference, the S&P-500 returned an average CAGR of just 11.24% over the same time period, and experienced a very strong rally in late 2020, the S&P usually averages around 8% per year.

6 Optimizations and Model Improvements

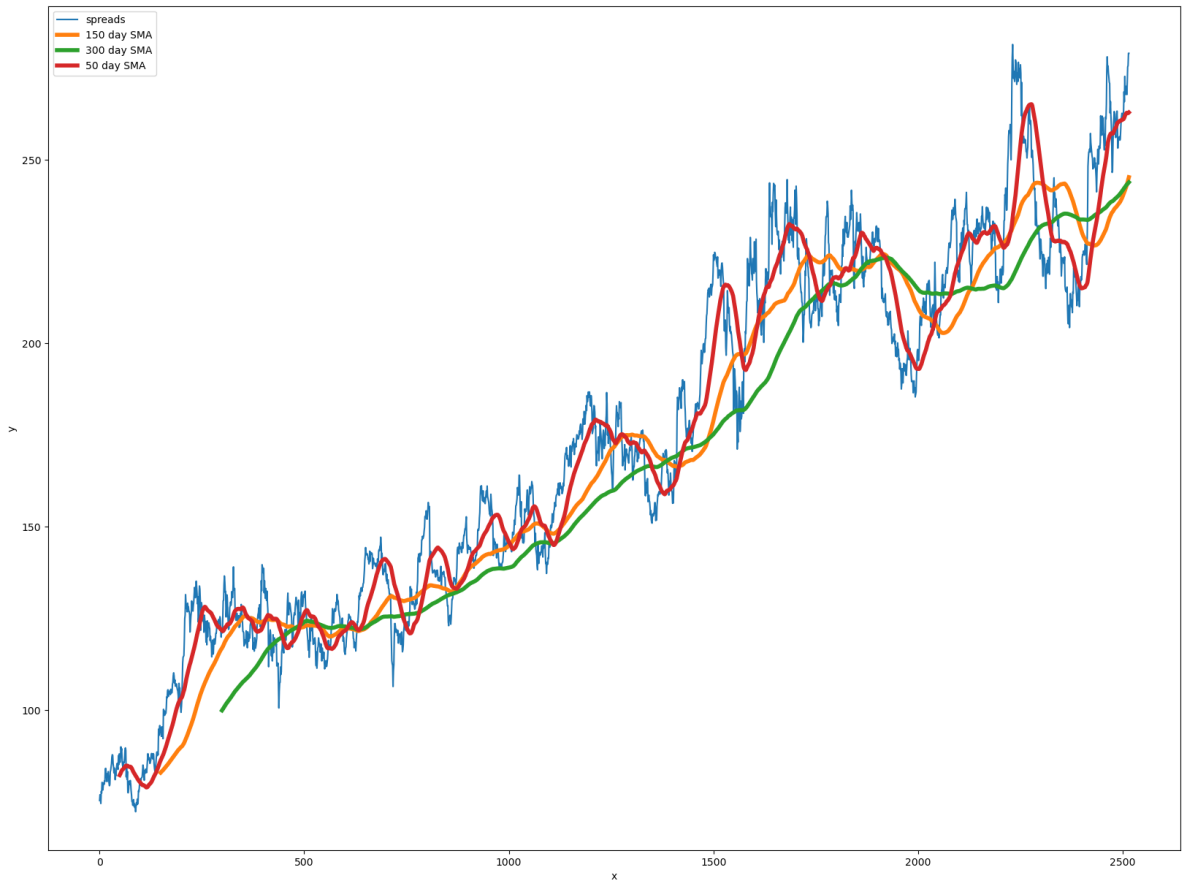
We notice that our algorithm, while generating some pretty impressive returns, does not fully capture potential value, mostly due to us using a pre-set criterion for the mean-reverting force to fire signals. Originally, this section was reserved for optimizing this value using basic mathematical non-linear programming methods. This would have likely required the use a whole lot more theory, while delivering an algorithm that is less flexible than what we are proposing next, and would be way out of line of the IB mathematics curriculum. The only upside to this is that these optimization problems are very common in quantitative finance and should be in every Quant Researcher's toolbox. Rather, in this section, we will propose a series of adjustments to the model that will make it more flexible, usable, and generate higher returns.

6.1 Simple Moving Average

We currently compute the mean function $\mu(t)$ by running a simple OLS regression and then using the line of best fit as an estimator for the mean. This makes us unable to adjust to dynamically changing base values of the spread oscillations. It's also better to recompute the mean as we go through time in the backtest, and we should have done this anyway in our base model. We define the SMA with the following:

$$\mu(t) = \frac{1}{n} \sum_{i=t-n}^{t} \mathcal{X}_i, \text{ where } |\mathcal{X}| \geq n$$

Where n is the sample size for the SMA. Esentially, the number of time frames we consider looking back.



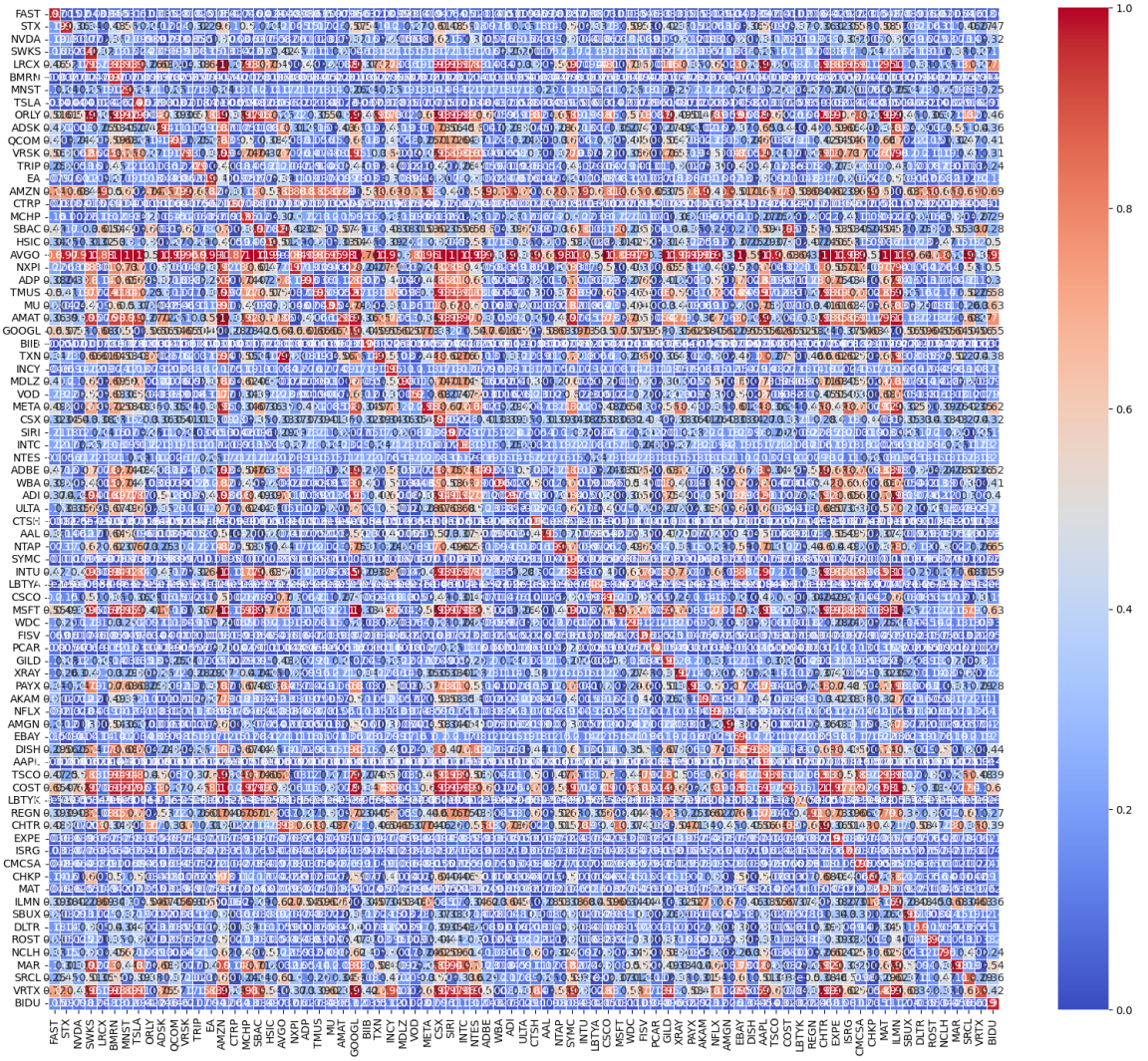
You can see that the 150 day SMA captures the dynamic nature of spreads a little better, while still not being perfect.

6.2 Optimizing Signals for Local Extrema

We intuitively note the fact that the most optimal entry and exit points (signal points) in a stat-arb strategy focusing OU oscillations in equity spreads are local extrema points of the non-continuous price function. Since out pricing date resembles a function that is discrete and not continuous, we cannot use a limit converging to zero to calculate the derivative of the prices. Thus we cannot use a regular derivative (since a requirement for that is having a continuous function), but we can approximate the derivative using a technique called Numerical Differentiation.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x) - f(x-1)}{\Delta x}$$

7 Appendix



Please note that imports might not mach in all of these, since they were used in a jupyter notebook environment.

Code for generating a sample Wiener process:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

NUM_PATHS = 10
SAMPLES = 1000

MU = 0
SIGMA = 1

rng = np.random.default_rng()
```

```
def compute_wiener() -> list[float]:
    value = 0
    wiener = []

    for _ in range(SAMPLES):
        s = rng.normal(MU, SIGMA)
        value += s

        wiener.append(value)

    return wiener


df_data = {}
for i in range(NUM_PATHS):
    path = compute_wiener()
    df_data[f'WPath {i}'] = path

plt.figure(figsize=(10, 7))

data = pd.DataFrame(df_data)
sns.lineplot(data=data)

plt.legend([], [], frameon=False)
plt.show()
```

Code for generating a sample Ornstein Uhlenbeck process:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

NUM_PATHS = 2
SAMPLES = 200
THETA = 0.2
MEAN = 0

MU = 0
SIGMA = 1

rng = np.random.default_rng()

def compute_wiener() -> list[float]:
    value = 0
    wiener = []

    for _ in range(SAMPLES):
        mean_rev_term = -THETA * (value - MEAN)
        s = mean_rev_term + rng.normal(MU, SIGMA)
        value += s

        wiener.append(value)

    return wiener


df_data = {}
for i in range(NUM_PATHS):
    path = compute_wiener()
```

```
    df_data[f'WPath {i}'] = path

data = pd.DataFrame(df_data)

long_form = pd.melt(data.reset_index(), id_vars='index', var_name='Path',
    value_name='Value')
long_form.rename(columns={'index': 'Time'}, inplace=True)

plt.figure(figsize=(10, 7))
sns.lineplot(data=long_form, x='Time', y='Value', hue='Path')

plt.legend([], [], frameon=False)
plt.show()
```

Code for performing an OLS regression on residuals for the two-step engle granger test:

```
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt

ko = pd.read_pickle('../data/ko.pkl')
pep = pd.read_pickle('../data/pep.pkl')

def regression_residuals(s1: pd.DataFrame, s2: pd.DataFrame) -> pd.DataFrame:
    x = s1[0]
    y = s2[0]

    # add a constant term
    x = sm.add_constant(x)

    model = sm.OLS(y, x)
    res = model.fit()

    return res.resid


residuals = regression_residuals(ko, pep)

print(residuals)

sns.lineplot(residuals)
plt.show()
```

Performing the ADF Test:

```
# performs the two step engle granger test on all pairs from the NASDAQ 100

import os
from tqdm.notebook import tqdm
from statsmodels.tsa.stattools import adfuller

DATA_BASE = '../data/nasdaq/'
unprobed_files = os.listdir(DATA_BASE)

# only include mature companies,
# also makes OLS simpler

files = []
for file in unprobed_files:
```

```
    file_path = os.path.join(DATA_BASE, file)
    df = pd.read_pickle(file_path)

    prices = df[0]
    if len(prices) != 2516:
        continue

    files.append(file)

p_values = {}
for i in tqdm(range(len(files)), desc="Running ADF Test"):
    base_file = files[i]
    base_file_path = os.path.join(DATA_BASE, base_file)
    base_df = pd.read_pickle(base_file_path)
    base_ticker = base_file.split('.')[0]

    base_p_values = []
    for j in range(len(files)):
        aux_file = files[j]
        aux_file_path = os.path.join(DATA_BASE, aux_file)
        aux_df = pd.read_pickle(aux_file_path)

        try:
            residuals = regression_residuals(base_df, aux_df)
            adf = adfuller(residuals)
            p_value = adf[1]
        except:
            print(f'error on base: {i}, aux: {j}')
            p_value = 1

        base_p_values.append(p_value)

    p_values[base_ticker] = base_p_values

# show sns heatmap
heatmap_index = list(map(lambda f: f.split('.')[0], files))

heatmap_df = pd.DataFrame(p_values, index=heatmap_index)
heatmap_df.to_pickle('../data/adf.pkl')
```

Code for performing a manually-picked treshold backtest:

```
import os
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt

# We define a backtesting method, that takes
# long-short tresholds and performs a backtest
# this is an oversimplified model that doesn't
# factor in the parameters of the underlying OU
# process and simply trades off arbitrarily
# defined tresholds, this follows an unoptimized
# P/L, which we will later optimize for risk
# appetite or optimality

TRAIN_SET_END = 1450
DATA_BASE = '../data/nasdaq/'
TICKER_BASE = 'AMGN'
```

```
TICKER_AUX = 'VOD'

base_path = os.path.join(DATA_BASE, TICKER_BASE + '.pkl')
aux_path = os.path.join(DATA_BASE, TICKER_AUX + '.pkl')

base = pd.read_pickle(base_path)
aux = pd.read_pickle(aux_path)
spreads = base - aux

def backtest(treshold_down: float, treshold_up: float):
    train_set_spreads = spreads[0:TRAIN_SET_END]

    test_set_spreads = spreads[TRAIN_SET_END:]
    test_set_len = len(test_set_spreads)

    # perform OLS fitting to X_{t} = \beta Y_{t} + \alpha
    x = list(range(len(train_set_spreads)))
    y = train_set_spreads[0]

    x = sm.add_constant(x)
    model = sm.OLS(y, x)
    res = model.fit()

    slope = res.params['x1']
    intercept = res.params['const']

    # perform backtest trading off tresholds
    total_value = 100
    portfolio_values = []

    entered_long = False
    long_entrance_price = 0

    entered_short = False
    short_entrance_price = 0

    enters_long = []
    enters_short = []

    for i in range(test_set_len):
        value = test_set_spreads.iloc[i, 0]

        real_time = TRAIN_SET_END + i
        pred_value = intercept + (slope * real_time)

        if pred_value + treshold_up < value:
            # End long position on spread
            if entered_long:
                entered_long = False
                growth = (value - long_entrance_price) / long_entrance_price
                total_value *= 1 + growth
                long_entrance_price = 0

                print(f'exiting long with growth: {growth * 100}%, at: {value},
                    total_value: {total_value}')

            # short spread
            if not entered_short:
                entered_short = True
                short_entrance_price = value
```

```
                enters_short.append(i)
                print(f'entering short at: {value}')

        elif pred_value + treshold_down > value:
            # End short position on spread
            if entered_short:
                entered_short = False
                growth = (short_entrance_price - value) / value
                total_value *= 1 + growth
                short_entrance_price = 0

                print(f'exiting short with growth: {growth * 100}%, at: {value},
                    total_value: {total_value}')

            # long-spread
            if not entered_long:
                entered_long = True
                long_entrance_price = value

                enters_long.append(i)
                print(f'entering long at: {value}')

        portfolio_values.append(total_value)

    print(f'total startegy value: {total_value}')
    return portfolio_values, enters_long, enters_short, slope, intercept


TRADE_DOWN = -10
TRADE_UP = 6

portfolio, long_entrances, short_entrances, total_slope, total_intercept =
    backtest(TRADE_DOWN, TRADE_UP)
portfolio_df = pd.DataFrame({
    'time': range(len(portfolio)),
    'portfolio_value': portfolio
})

plt.title('Manually-picked Treshold Trade Backtest on AMGN/VOD Spreads')
sns.lineplot(data=portfolio_df, x='time', y='portfolio_value')
plt.show()
```

Code for plotting model tresholds:

```
spreads = spreads[TRAIN_SET_END:]
print(len(spreads))

x = range(len(spreads))
spreads_df = pd.DataFrame({
    'x': x,
    'y': spreads[0]
})

plt.figure(figsize=(19, 14))

ax = sns.lineplot(data=spreads_df, x='x', y='y')

# Add long-short tresholds
y_down = [((p_x + TRAIN_SET_END) * total_slope) + total_intercept + TRADE_DOWN for p_x
```

```
    in x]
y_reg = [((p_x + TRAIN_SET_END) * total_slope) + total_intercept for p_x in x]
y_up = [((p_x + TRAIN_SET_END) * total_slope) + total_intercept + TRADE_UP for p_x in x]

tresholds_df = pd.DataFrame({
    'x': x,
    'y_down': y_down,
    'y_reg': y_reg,
    'y_up': y_up
})

sns.lineplot(data=tresholds_df, x='x', y='y_down', color='green', linewidth=2.5)
sns.lineplot(data=tresholds_df, x='x', y='y_reg', color='blue', linewidth=2.5)
sns.lineplot(data=tresholds_df, x='x', y='y_up', color='red', linewidth=2.5)

# Add trade entrances and exits
long_x = []
long_y = []

short_x = []
short_y = []

for x in long_entrances:
    y = spreads_df.iloc[x, 1]
    long_x.append(x)
    long_y.append(y)

for x in short_entrances:
    y = spreads_df.iloc[x, 1]
    short_x.append(x)
    short_y.append(y)

long_df = pd.DataFrame({
    'x': long_x,
    'y': long_y
})

short_df = pd.DataFrame({
    'x': short_x,
    'y': short_y
})

sns.scatterplot(data=long_df, x='x', y='y', color='green', s=150)
sns.scatterplot(data=short_df, x='x', y='y', color='red', s=150)

ax.set(xlabel='time after trainset', ylabel='common ylabel')
plt.show()
```

Code for getting the mean-reverting force for second backtest:

```
import ou

base_path = os.path.join(DATA_BASE, TICKER_BASE + '.pkl')
aux_path = os.path.join(DATA_BASE, TICKER_AUX + '.pkl')

base = pd.read_pickle(base_path)
aux = pd.read_pickle(aux_path)
spreads = base - aux

trainset = spreads[:TRAIN_SET_END].iloc[:, 0]
```

```
testset = spreads[TRAIN_SET_END:].iloc[:, 0]

print(len(trainset))

mean_coeff, mean_intercept, theta, sigma = ou.calibrate(trainset)

print(f'mu(t) = {mean_coeff}t + {mean_intercept}')
print(f'theta = {theta}')
print(f'sigma = {sigma}')
```

Code for performing the second backtest:

```
data = pd.DataFrame({
    'spread': testset,
    'base': base.iloc[TRAIN_SET_END:, 0],
    'aux': aux.iloc[TRAIN_SET_END:, 0]
})

class Position:
    def __init__(self, short: bool, entry: float, capital: float, entry_index: int) ->
        None:
        self.short = short
        self.entry = entry
        self.capital = capital
        self.entry_index = entry_index

    def liquidate(self, price: float) -> float:
        if not self.short:
            # long
            shares = self.capital / self.entry
            return shares * price
        else:
            # short
            price_delta = self.entry - price
            shares = self.capital / self.entry
            return (price_delta * shares) + self.capital


def alocate_capital_fractionally(cap, base, aux):
    total = base + aux
    num_shares = cap / total
    return num_shares * base, num_shares * aux


def backtest(signal_treshold):
    working_capital = 100

    base_position: Position = None
    aux_position: Position = None

    capital_progression = []
    long_trades = []
    short_trades = []

    for index, rows in data.iterrows():
        spread = rows['spread']
        base = rows['base']
        aux = rows['aux']

        # get mean reverting force at index
```

```
        mean = mu(index)
        force = theta * (mean - spread)

        if force > signal_treshold:
            # bet spread goes up
            # long base
            if base_position == None:
                print(f'entering long at {index}')
                base_cap, aux_cap = alocate_capital_fractionally(working_capital, base,
                    aux)

                base_position = Position(False, base, base_cap, index)
                aux_position = Position(True, aux, aux_cap, index)

            elif base_position.short:
                # liquidate base short, aux long
                print(f'liquidating short at {index}')
                short_trades.append((base_position.entry_index, index))

                working_capital = base_position.liquidate(base)
                working_capital += aux_position.liquidate(aux)

                base_position = None
                aux_position = None

        elif -force > signal_treshold:
            # bet spread goes down
            # short base
            if base_position == None:
                print(f'entering short at {index}')
                base_cap, aux_cap = alocate_capital_fractionally(working_capital, base,
                    aux)

                base_position = Position(True, base, base_cap, index)
                aux_position = Position(False, aux, aux_cap, index)
            elif not base_position.short:
                # liquidate base long, aux short
                print(f'liquidating long at {index}')
                long_trades.append((base_position.entry_index, index))

                working_capital = base_position.liquidate(base)
                working_capital += aux_position.liquidate(aux)

                base_position = None
                aux_position = None

        capital_progression.append(working_capital)

    return capital_progression, working_capital, long_trades, short_trades


capital_progression, capital, long_trades, short_trades = backtest(0.2)
print(f'capital: {capital}')

df = pd.DataFrame({
    'x': range(len(capital_progression)),
    'y': capital_progression
})

sns.lineplot(data=df, x='x', y='y')
```

```
plt.show()
```

Code for plotting the results of the second backtest:

```
# show trade entrances and exits

data['index'] = data.index

plt.figure(figsize=(18, 13))

sns.lineplot(data=data, x='index', y='spread')

for (entry, exit) in long_trades:
    entry_y = data.loc[entry]['spread']
    exit_y = data.loc[exit]['spread']

    plt.plot([entry, exit], [entry_y, exit_y], linewidth=5, color='green')

for (entry, exit) in short_trades:
    entry_y = data.loc[entry]['spread']
    exit_y = data.loc[exit]['spread']

    plt.plot([entry, exit], [entry_y, exit_y], linewidth=5, color='red')

plt.show()
```