Cointegration

Joy Tolia

Contents

1 Time Series

1.1 Code Overview

A summary of classes made for timeseries analysis is in Table 1. All code written can be seen in Section 2.1. Included is code in Section 2.1.1 which as an example to get results.

Table 1: Summary of the classes made.

Name	Description
`TimeSeries`	• This is to hold the time series which you pass in. • It will convert Excel dates to Matlab dates and replace NaN and zero values in the time series.
`TimeSeriesCollection`	• This is to hold a set of time series' together. • It will take in multiple time series then find the intersection of all the dates available and adjust the time series' accordingly. • It will also hold the returns of the series and the re-indexed series.
`JCData`	• This holds the important information from the Johansen's procedure.
`BacktestCoint`	• This holds the data from fitting an Ornstein-Uhlenbeck process onto the mean reverting spread obtained from the Johansen's procedure. • This also holds `BacktestData` objects for different strategies and parameters.
`BacktestData`	• This is to hold all the information from backtesting given strategies on the mean reverting spread.

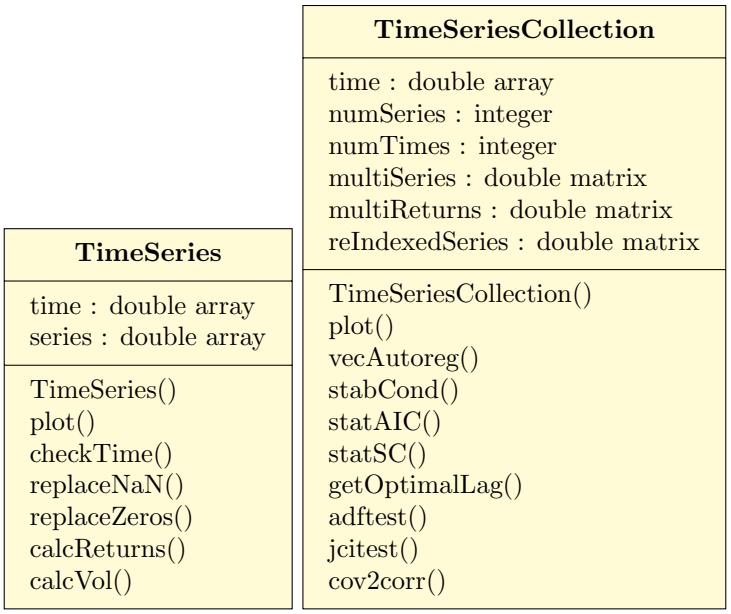


Figure 1: Time series related UML class diagrams.

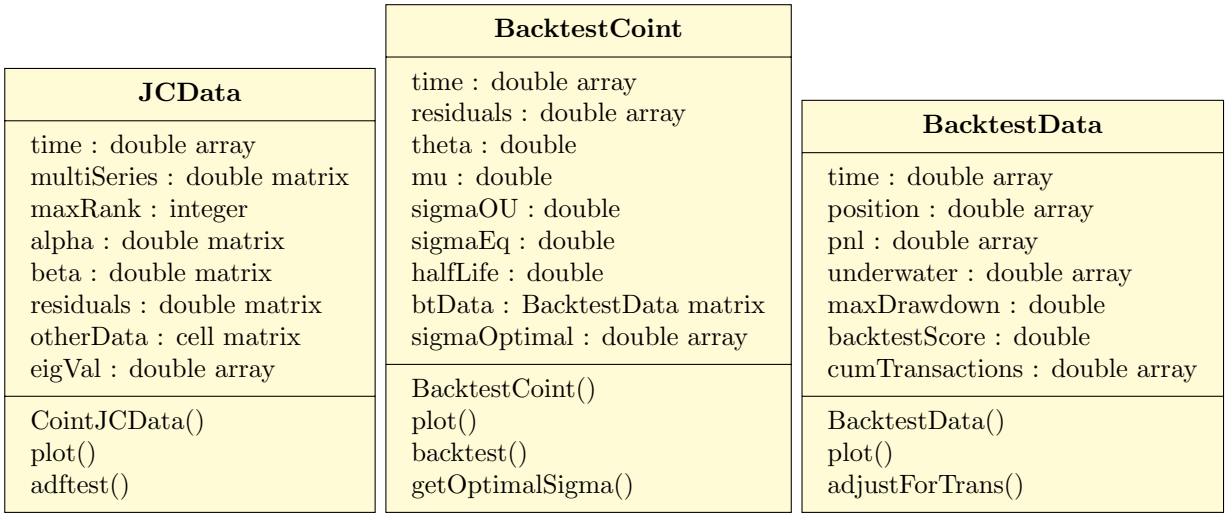
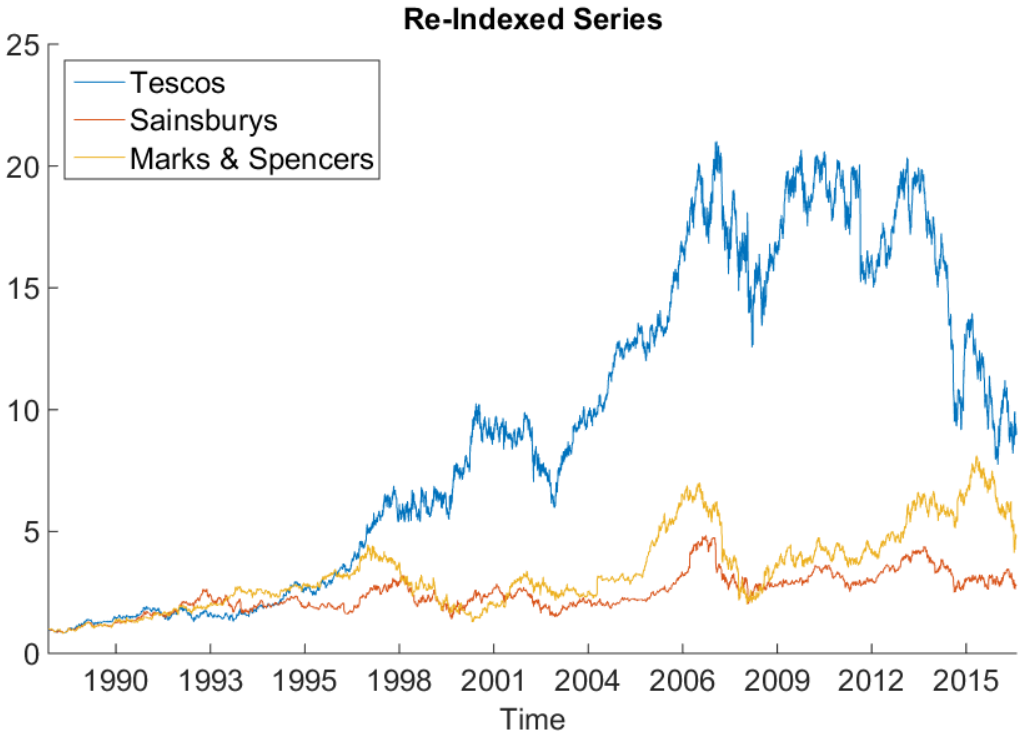


Figure 2: Cointegration and backtesting UML class diagram.

In this section we will look at theory and results at the same time. The results are obtained from three sets of time series which are the stock prices of Tescos, Sainsburys and Marks & Spencers from 01/07/1988 to 20/07/2016. The data is obtained from Yahoo. In Figure 3 we can see the reindexed time series:

Figure 3: Reindexed time series for Tescos, Sainsburys and Marks & Spencers.



1.2 In-depth Learning

1.2.1 Notation

Let us work through how we can do multivariate linear regression. Let us define some notation in the following Table 2:

Table 2: Notation used for vector autoregression.

Description	Type	Dummy Index	Last Index
Number of time series	$\mathbb{N}$	i	n
Number of time points for price levels	$\mathbb{N}$	t	T
Lag	$\mathbb{N}$	j	p
Price level for series i at time t	$\mathbb{R}$	$y_{i,t}$	$y_{n,T}$
Price levels for all series at time t	$\mathbb{R}^{n\times 1}$	$Y_t = (y_{1,t}, \ldots, y_{n,t})'$	Y_T
Return for series i at time t	$\mathbb{R}$	$r_{i,t}$	$r_{i,t}$
Returns for all series at time t	$\mathbb{R}^{n\times 1}$	$R_t = (r_{1,t}, \ldots, r_{n,t})'$	R_T
Regressed beta matrix for a given lag j	$\mathbb{R}^{n\times n}$	β_j	β_p
Regressed beta vector for intercept	$\mathbb{R}^{n\times 1}$	β_0	-
Matrix of ones with x rows and y columns	$\mathbb{R}^{x,y}$	$\mathbb{1}_{x,y}$	-
Residual at time t	$\mathbb{R}^{n\times 1}$	ϵ_t	ϵ_T
Difference in daily price levels for series i at time t	$\mathbb{R}$	$\Delta y_{i,t} = y_{i,t} - y_{i,t-1}$	-
Difference in daily price levels for all series at time t	$\mathbb{R}^{n\times 1}$	$\Delta Y_t = (\Delta y_{1,t}, \ldots, \Delta y_{n,t})'$	-

1.2.2 Vector Autoregression

We would the the following regression:

$$R_t = \beta_0 + \sum_{j=1}^{p} \beta_j R_{t-j} + \epsilon_t \tag{1}$$

As we would like to do this in one go, we can turn the sum in Equation 1 into a matrix multiplication where:

$$\beta = (\beta_0, \beta_1, \ldots, \beta_p) \in \mathbb{R}^{n\times(np+1)}, \quad \hat{R}_t = (1, R_{t-1}, \ldots, R_{t-p})' \in \mathbb{R}^{(np+1)\times 1} \tag{2}$$

Let us denote $R = (R_p, \ldots, R_T) \in \mathbb{R}^{n\times(T-1-p)}$ as the matrix containing returns from time p to T for each series. Let $\epsilon = (\epsilon_p, \ldots, \epsilon_T) \in \mathbb{R}^{n\times(T-1-p)}$ as the matrix containing the residuals from time p to T for each series. Finally we denote $Z = \left(\hat{R}_p, \ldots, \hat{R}_T\right) \in \mathbb{R}^{(np+1)\times(T-1-p)}$. Then we have the following matrix equation:

$$R = \beta Z + \epsilon \tag{3}$$

We know to minimise ϵ by varying β, we get that the following $\hat{\beta}$ minimised the residuals:

$$\hat{\beta} = RZ'\left(ZZ'\right)^{-1} \tag{4}$$

And we have the residuals are given by the following:

$$\hat{\epsilon} = Y - \hat{\beta}Z \tag{5}$$

All the code for the vector autoregression can be seen in Section 2.1.3.1. Now we look at the results from the vector autoregression with lag of $p = 3$ on the returns of the time series shown in Figure 3. We get the correlation matrix seen in Table 3. Clearly, we see that the returns are well correlated between Tescos and Sainsburys. Whereas for Marks & Spencers with others are positively correlated but not as much. This makes sense as Marks & Spencers is in a slightly different market compared to others.

Table 3: Correlations between returns of Tescos, Sainsburys and Marks & Spencers. The code implementation can be seen in Sections 2.1.3 and 2.1.3.1.

	Tescos	Sainsburys	Marks & Spencers
Tescos	100%	51%	35%
Sainsburys	51%	100%	35%
Marks & Spencers	35%	35%	100%

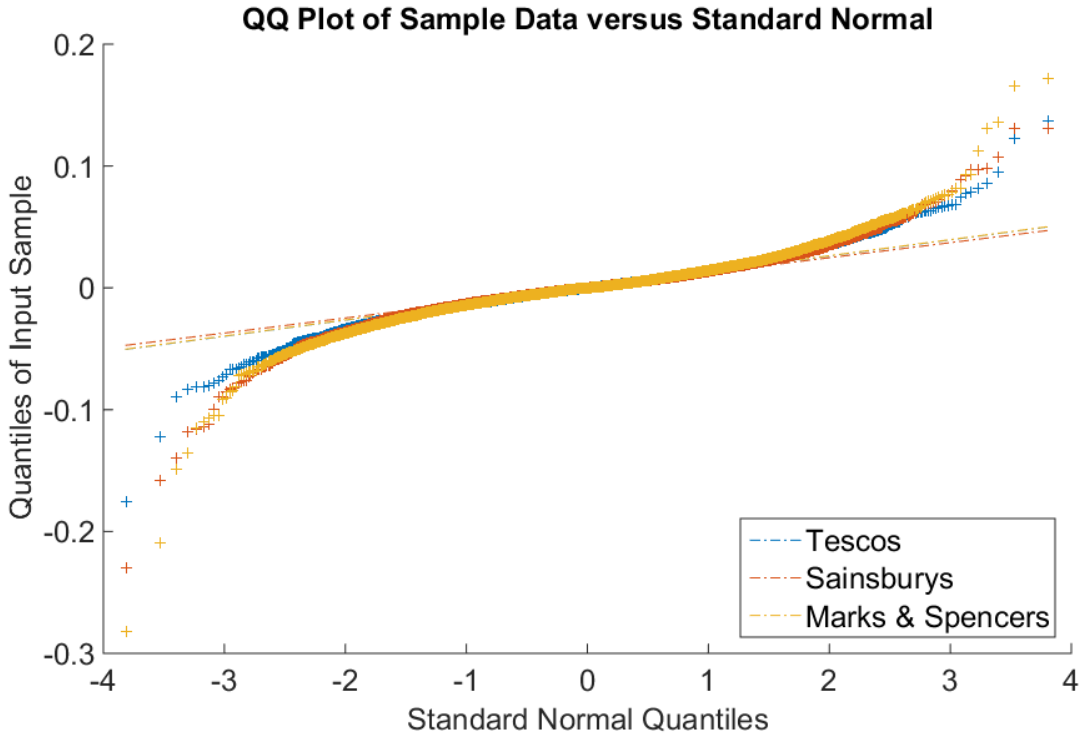
Let us now study the error statistics obtained from the residuals shown in Table 4. Getting an RMSE of around 2% return is quite a big forecasting error.

Table 4: Error statistics from the residuals from vector autoregression where the implemented code can be seen in Section 2.1.3.1.

	ME	RMSE	MAE	MPE	MAPE	MASE
Tescos	0	0.0166	0.0119	NaN	NaN	0.6987
Sainsburys	0	0.0171	0.0117	NaN	NaN	0.6976
Marks & Spencers	0	0.0184	0.0125	NaN	NaN	0.6951

Finally, we look at a Q-Q plot to see how the distributions of the residuals compare to the normal distribution. We see this in Figure 4. It is clear from the Q-Q plot that the distributions of the residuals have much fatter tails than the normal distribution. Hence this method of forecasting the returns does not seem to give the best results.

Figure 4: Q-Q plot of the residuals from vector autoregression.



1.2.3 Optimal Lag Selection

We build the residual covariance matrix using the following formula:

$$\hat{\Sigma} = \frac{1}{T'} \times \hat{\epsilon}\hat{\epsilon}' \tag{6}$$

Where we use $T' = T - 1 - j$ is the number of observations of residuals we have for lag j. We can use the Akaike information criterion (AIC) or the Schwartz criterion (SC) also known as Bayesian information criterion to get the optimal lag:

$$\text{AIC} = \log\left(\left|\hat{\Sigma}\right|\right) + \frac{2k'}{T'} \tag{7}$$

$$\text{SC} = \log\left(\left|\hat{\Sigma}\right|\right) + \frac{k' \log\left(T'\right)}{T'} \tag{8}$$

Where $k' = n \times (nj + 1)$ for lag j. We want to test the AIC and SC values for different lags and choose the lowest value for the optimal lag. This is because we would like to have the smallest residuals, when we have small residuals the determinant of the covariance matrix is close to zero. This would make the AIC and SC values lower hence we are looking for the lowest values. The k' term penalises for a larger lag.

The code implementation for the covariance matrix can be seen in Section 2.1.3.1. For the AIC and SC tests, see Sections 2.1.3.4 and 2.1.3.5 respectively.

Finally the code to get the optimal lag can get seen in Section 2.1.3.2. Here we input a range consisting of minimum and maximum lag that we want to test between. Then we get the AIC and SC statistics as well as checking for stability which is in the next section and deleting unstable lags. Finally, we get the minimum of the AIC and SC statistics, giving priority to the AIC test.

Below, in Table 5, we can see the AIC and SC statistics for lags between 1 and 10. With the lowest values in red. As we are giving priority to the AIC statistic, we get that a lag of 3 is the most optimal.

Table 5: AIC and SC statistics for lags between 1 and 10.

Lag	AIC	SC
1	-24.77400	-24.76266
2	-24.78035	-24.76050
3	-24.78285	-24.75450
4	-24.78114	-24.74427
5	-24.77961	-24.73423
6	-24.77907	-24.72517
7	-24.77778	-24.71537
8	-24.77680	-24.70588
9	-24.77580	-24.69635
10	-24.77443	-24.68646

1.2.4 Stability Condition

Given Equation 1, we need that β_j^k for $j = 1, \ldots, p$ tends to the zero matrix for the equation to be stable. We can check this by checking all the eigenvalues of β_j have an absolute value strictly less than one. This is because of the diagonal decomposition $\beta_j = VDV^{-1}$ where:

$$D = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{pmatrix} \tag{9}$$

Therefore, if $|\lambda_i| < 1$ then we know that elements in D^k will not diverge as $k \to \infty$. We have that $\beta_j^k = VD^kV^{-1}$, therefore if elements of D^k do not diverge then elements of β_j^k will not diverge.

Hence, we have make a stability condition that all the eigenvalues of the matrices β_j for $j = 1, \ldots, p$ have their absolute value less than one for us to have a stable model which does not diverge.

The code for checking the stability condition is in Section 2.1.3.3. Here we get the eigenvalues of the matrix by using the built in `eig` function provided in Matlab. Then we check if each eigenvalue has an absolute value of less than one. Finally, for the time series' we are working with, we get that for all lags between 1 and 30 that they all give us stable matrices.

1.2.5 Augmented Dickey-Fuller Test

This test is to see if the a time series is stationary. A time series is stationary if its statistical properties such as mean, variance and autocorrelation are independent of time constants. Assuming the daily differences of a time series $\Delta y_{i,t}$ are stationary then we can do the following regression:

$$\Delta y_{i,t} = \phi_i y_{i,t-1} + \sum_{j=1}^{p} \phi_{i,j} \Delta y_{i,t-j} + \epsilon_{i,t} \tag{10}$$

By regressing $y_{i,t-1}, \Delta y_{i,t-1}, \ldots, \Delta y_{i,t-p}$ on $\Delta y_{i,t}$, we get the values for $\phi_i, \phi_{i,1}, \ldots, \phi_{i,p}$. Assuming all the differences in Equation 10 are stationary then if $y_{i,t-1}$ is not stationary then the value for ϕ_i will be insignificant or close to zero. Our null hypothesis is that $y_{i,t}$ has a unit root or is non stationary. If ϕ_i is not insignificant then we reject the null hypothesis in favour of the alternative (stationary) model. We use the Dickey-Fuller distribution to check for the significance of ϕ_i.

We use the Matlab in built function `adftest` for doing this test, however we implement a function in `TimeSeriesCollection` shown in Section 2.1.3. This function calls the in built matlab function for the ADF test multiple times to be able to give us the output for each series contained within the object. We get that in terms of price levels, all the the series accepted the null hypothesis of having a unit root implying that they are all non stationary.

Having done the ADF tests on our time series, we find that non of them reject the null hypothesis of having a unit root implying that they are all non stationary time series. Then using the inbuilt Matlab function we can specify the model `'ARD'` to check for drift and we still don't get a rejection of the null hypothesis implying that the time series' are non stationary.

1.2.6 Johansen's Procedure

This is a generalisation of the augmented Dickey-Fuller test. A pair of time series are cointegrated if they are integrated series of order 1 and there exists a linear combination of the time series' such that it is a stationary time series. Suppose we have the following equation:

$$\Delta Y_t = \Phi D_t + \Pi Y_{t-1} + \sum_{j=1}^{p} \Gamma_j \Delta Y_{t-j} + \epsilon_t \tag{11}$$

In Equation 11, $\Delta Y_t, \Delta Y_{t-1}, \ldots, \Delta Y_{t-p}$ are all assumed to be stationary. The only term that could have non stationary terms is Y_{t-1}. Therefore, as with the augmented Dickey-Fuller test, we want to study the term in front of Y_{t-1} which is Π. This means that ΠY_{t-1} must have cointegrating relations if they exist.

$\Pi \in \mathbb{R}^{n \times n}$, we would like to study this matrix. If Y_t was stationary or did not a unit root then Π would be non-singular or have a full rank. However, if Y_t was non stationary or had a unit root then Π would be non singular or have rank $r < n$, there would be two options in this case:

1. $r = 0$, Π has null rank which means it is a zero matrix and there are no cointegrating relations. This is equivalent of ϕ_i being insignificant in the augmented Dickey-Fuller test

2. $0 < r < n$, this means that there are r cointegrating relations which make the linear combinations of the series' stationary. This is what we are looking for.

Given Π has rank r, then it can be written as $\Pi = \alpha\beta'$ where $\alpha, \beta \in \mathbb{R}^{n\times r}$ and rank of both α and β are r.

The Johansen's procedure is a set of iterative hypothesis tests where the null hypothesis is that rank of Π is $r_0 = r$ and the alternative is that the rank $r_0 > r$ where $r = 0, \ldots, n-1$.

Therefore, we start with the null hypothesis that the rank $r_0 = 0$, if we do not reject the null hypothesis then the rank of Π is 0 and there are no cointegrating relations. However, if we do reject the null hypothesis then we know that the rank is greater than 0 which means there is at least one cointegrating relation. The next hypothesis has the null hypothesis that the rank $r_0 = 1$ and the alternative is that $r_0 > 1$ so here if we accept the null then we know the rank of Π is 1 and if we reject the null then the rank of Π is strictly greater than one. We keep doing this until we test the last hypothesis with null that the rank of Π is $n-1$.

Now that we have an overview for the Johansen's procedure. Let us see how each hypothesis test works. In each test, we would like to test for the rank of the matrix Π. We can do the diagonalisation decomposition to get $\Pi = VDV^{-1}$, where D has a form shown in Equation 9. It can be shown that the rank of Π is equal to the rank of D. The rank of D is the number of non-zero eigenvalues. Therefore, the problem comes down to testing if the eigenvalues are too close to zero. The eigenvalues have a certain property that make their values between 0 and 1.

Suppose we order the eigenvalues as follows; $1 > \lambda_1 > \cdots > \lambda_n > 0$. Each hypothesis test checks if the remaining eigenvalues are close enough to zero. There are two types of tests, the trace eigenvalue test and the maximum eigenvalue test, given we are testing for rank r_0:

β is formed from the eigenvectors corresponding to the largest r eigenvalues.

$$\text{Trace Statistic} = -T \sum_{i=r_0+1}^{n} \log(1-\lambda_i) \tag{12}$$

$$\text{Max Eigenvalue Statistic} = -T \log(1-\lambda_{r_0+1}) \tag{13}$$

Given the test statistics in Equations 12 and 13, we compare them against critical values to accept or reject the null hypothesis. For example if we look at the max eigenvalue statistic then if λ_{r_0+1} is close to zero then the test statistic is close to zero, but as $\lambda_{r_0+1} \to 1$ the test statistic tends to infinity. Therefore, to reject the null hypothesis, we want a large test statistic but to accept the null hypothesis we want a test statistic close to zero.

β is formed from the eigenvectors corresponding to the biggest r eigenvalues found above. Then we can get the residuals $\epsilon = \beta' Y \in \mathbb{R}^{r\times T}$. We now have r stationary time series that we can backtest.

All the code for the Johansen's procedure can be found in the constructor of `JCData` in Section 2.1.4. In this constructor, we run the in built Matlab function for the Johansen's procedure called `jcitest`

and do both the trace and the maximum eigenvalue tests. From this, we choose the highest rank given out of the two tests so we can backtest every cointegrating relations available and store all the data from the tests. In Tables 6 and 7, we can see the data.

Table 6: Data from the Johansen's procedure using the trace test.

Rank	Null rejected	Test statistic	Critical value	Probability of statistic	Eigenvalue
0	1	31.3635	29.7976	0.0328	0.0033
1	0	7.4230	15.4948	0.5702	7.2149×10^{-4}
2	0	2.1556	3.8415	0.1426	2.9533×10^{-4}

Table 7: Data from the Johansen's procedure using the maximum eigenvalue test.

Rank	Null rejected	Test statistic	Critical value	Probability of statistic	Eigenvalue
0	1	23.9406	21.1323	0.0196	0.0033
1	0	5.2673	14.2644	0.7215	7.2149×10^{-4}
2	0	2.1556	3.8415	0.1426	2.9533×10^{-4}

Let us draw some conclusions from the results in Tables 6 and 7 for the Johansen's procedure.

- In both tests the default significance level of $5\% = 0.05$ was used.
- We can see that the null hypothesis for the rank of 0 was rejected in both tests meaning that the rank was at least 1. However, the null hypothesis for the rank of 1 was accepted in both tests meaning that there is one cointegrating relation between the three time series.
- It is clear why the hypotheses were accepted or rejected for each rank by comparing the test statistics and the critical values. The null hypothesis was rejected when the test statistic was greater than the critical value.
- We can also use the probability of the test statistic for check the hypotheses, where the null was rejected when the probability was less than the significance level of 0.05.
- Finally, we can see the eigenvalues clearly that that the first eigenvalue is much bigger than the second two which gives the best intuition that the matrix has rank 1 due to the diagonalisation.
- It is interesting to note that the two tests have the same results for the last rank. This is because when there is only one eigenvalue left to test, both tests coincide in terms of their methods.

1.3 Cointegration and Statistical Arbitrage

Once we have our residual time series ϵ_t from the Johansen's procedure, we would like to fit the Ornstein-Uhlenbeck process which is shown in Equation 14 where dW_t is a Brownian motion increment.

$$d\epsilon_t = -\theta\left(\epsilon_t - \mu\right)dt + \sigma_{\mathrm{OU}}dW_t \tag{14}$$

We can solve the SDE in Equation 14 by doing the following:

$$\begin{aligned}
d\epsilon_t &= -\theta\left(\epsilon_t - \mu\right)dt + \sigma_{\mathrm{OU}}dW_t \\
&\Updownarrow \\
d\left(\epsilon_t e^{\theta t}\right) &= \theta\mu e^{\theta t}dt + \sigma_{\mathrm{OU}}e^{\theta t}dW_t \\
&\Updownarrow \\
\epsilon_{t+\tau}e^{\theta(t+\tau)} - \epsilon_t e^{\theta t} &= \theta\mu\int_t^{t+\tau} e^{\theta s}ds + \sigma_{OU}\int_t^{t+\tau} e^{\theta s}dW_s \\
&\Updownarrow \\
\epsilon_{t+\tau} &= \epsilon_t e^{-\theta\tau} + \mu\left(1 - e^{-\theta\tau}\right) + \sigma_{OU}\int_t^{t+\tau} e^{-\theta(t+\tau-s)}dW_s
\end{aligned}$$

We have the Equation 15 shown below by writing the random term as $r_{t,\tau}$:

$$\epsilon_{t+\tau} = \epsilon_t e^{-\theta\tau} + \mu\left(1 - e^{-\theta\tau}\right) + r_{t,\tau} \tag{15}$$

Therefore, running a regression on ϵ_t against itself with a lag of 1 day or $\tau = 1/252$, we can find the constants A and B shown in Equation 16:

$$\epsilon_{t+\tau} = A + B\epsilon_t + r_{t,\tau} \tag{16}$$

Equating coefficients in Equations 15 and 16, we find θ and μ:

$$\theta = -\frac{\log\left(B\right)}{\tau}, \quad \mu = \frac{A}{1-B} \tag{17}$$

Now, we would like to find a formula for σ_{OU}. Let us find the expectation and variance of $\epsilon_{t,\tau}$.

$$\begin{aligned}
\mathbb{E}\left[\epsilon_{t+\tau}|\epsilon_t\right] &= \mathbb{E}\left[\epsilon_t|\epsilon_t\right]e^{-\theta\tau} + \mu\left(1 - e^{-\theta\tau}\right) + \underbrace{\sigma_{OU}\mathbb{E}\left[\int_t^{t+\tau} e^{-\theta(t+\tau-s)}dW_s\middle|\epsilon_t\right]}_{=0} \\
&= \epsilon_t e^{-\theta\tau} + \mu\left(1 - e^{-\theta\tau}\right)
\end{aligned}$$

$$\begin{aligned}
\mathbb{V}\left[\epsilon_{t+\tau}|\epsilon_t\right] &= \mathbb{E}\left[\left(\epsilon_{t+\tau} - \mathbb{E}\left[\epsilon_{t+\tau}|\epsilon_t\right]\right)^2\middle|\epsilon_t\right] \\
&= \mathbb{E}\left[\left(\sigma_{OU}\int_t^{t+\tau} e^{-\theta(t+\tau-s)}dW_s\right)^2\right] \\
&= \sigma_{\mathrm{OU}}^2\mathbb{E}\left[\int_t^{t+\tau} e^{-2\theta(t+\tau-s)}ds\right] \quad \text{(By Ito isometry)} \\
&= \frac{\sigma_{\mathrm{OU}}^2}{2\theta}\left(1 - e^{-2\theta\tau}\right)
\end{aligned}$$

Therefore, we get formula in Equation 18 for calculating σ_{OU}:

$$\sigma_{\text{OU}} = \sqrt{\frac{2\theta \mathbb{V}\left[\epsilon_{t+\tau} | \epsilon_t\right]}{1 - e^{-2\theta\tau}}} \tag{18}$$

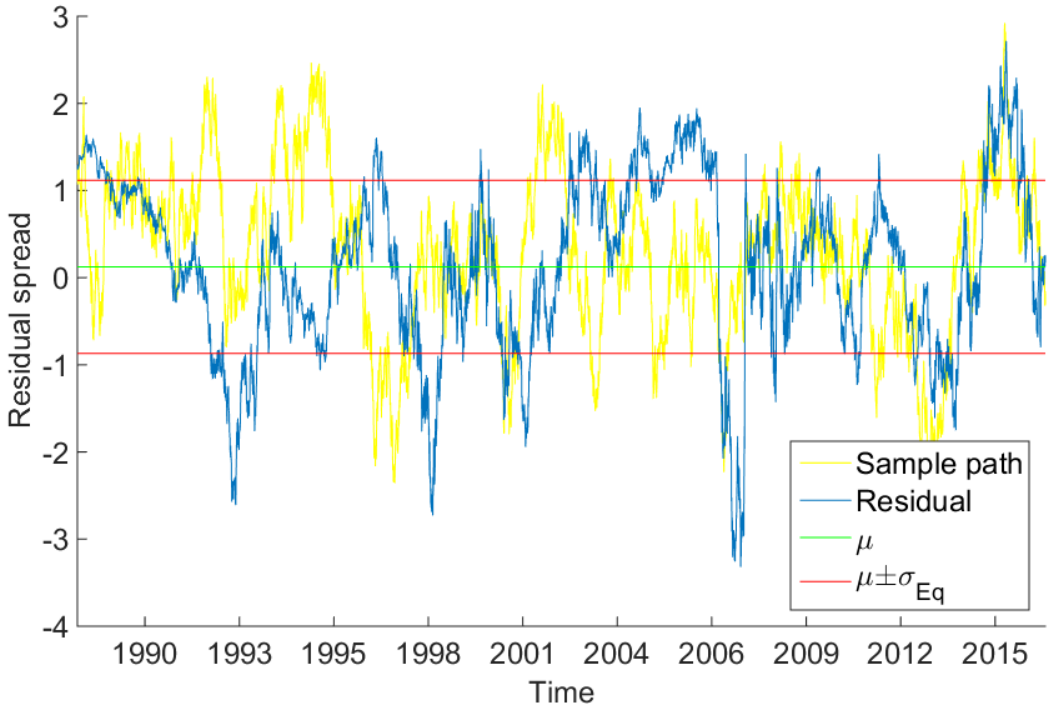
We have two extra parameters; σ_{Eq} which will be used as signals for entry and exit for our trading strategy and also the half life $\tilde{\tau}$ which have the formulas shown in Equation 19:

$$\sigma_{\text{Eq}} \approx \frac{\sigma_{\text{OU}}}{\sqrt{2\theta}}, \quad \tilde{\tau} \propto \frac{\log(2)}{\theta} \tag{19}$$

We do all these calculations in the constructor of `BacktestCoint` in Section 2.1.5 where we also use the built in Matlab function `fitlm` to fit a linear regression model. In Figure 5, we can see the residual with its mean μ and entry/exit barrier $\mu \pm \sigma_{\text{Eq}}$. From the figure, we can see that the residual is clearly mean reverting however, it does not necessarily look symmetric as you can see it goes far more negative in terms of size then it does positive. But if we look at the time that is spent outside $\mu \pm \sigma_{\text{Eq}}$ then it seems about equal. We can also see a sample plot in yellow driven by Equation 20 where ϕ is a standard normal sample.

$$\epsilon_{t+\tau} = \epsilon_t e^{-\theta\tau} + \mu\left(1 - e^{-\theta\tau}\right) + \sigma_{\text{OU}}\phi\sqrt{\frac{1 - e^{-2\theta\tau}}{2\theta}} \tag{20}$$

Figure 5: Plot of the residuals with its mean, standard deviation barriers and a sample path.



We get the parameters shown in Table 8. The most useful parameter for our intuition is the half line $\tilde{\tau}$ which says that it takes under a year to go from $\mu \pm \sigma_{\text{Eq}}$ to μ. However from Figure 5, it

seems that a lot of the times, it takes longer than the half life to get back to equilibrium, this is important as we need to know the approximate or expected time we are going to hold the position.

Table 8: Parameters from fitting an Ornstein-Uhlenbeck process to the residuals.

θ	μ	σ_{OU}	σ_{Eq}	$\tilde{\tau}$
1.4934	0.1240	1.7157	0.9928	0.4641

Given this data and having have a way to score each strategy depending on the return and risk, we can do an optimisation to find the optimal entry/exit levels. This can be seen in Section 2.1.5.1 and 2.1.5.2. To optimise our entry/exit levels, we make an objective function which we can adjust to what we want to maximise or minimise. In this case we are trying to maximise the end P&L divided by the maximum drawdown. Then we use the Matlab built in function `fminsearch` to do the optimisation with the initial point as σ_{Eq} to find the optimal entry/exit levels or equivalently the optimal σ.

1.4 Backtesting

Once we have our residuals, we would like to make a strategy that would have the most returns and the smallest maximum drawdown. We have made a few different strategies, all have their own unique selling points. All strategies have one part in common which is to buy low and sell high and vice versa. For this part of the project we look at the code in Section 2.1.6. This is a class which does all the backtesting and produces plots. Let us now look at the different strategies for trading the residuals.

With each strategy we work out the position held of the residual time series usually in the multiples of 1, with the code for this in Section 2.1.6.2. From there we can calculate the P&L by doing dot product on the position array and the daily difference of the residual time series. From there we can calculate the underwater plot and the maximum drawdown from the P&L array, with the code for this in Section 2.1.6.3. For the underwater plot, we look at the past highest P&L and take away the current P&L, to calculate the maximum drawdown we get the maximum of the underwater values. All of this is done in the constructor of `BacktestData` in Section 2.1.6.

We evaluate each strategy using a backtesting score which is shown in Equation 21:

$$\text{Backtest Score} = \frac{\text{End P\&L}}{\text{Maximum Drawdown}} \tag{21}$$

1.4.1 Strategy 1

In this strategy, we take positions at $\mu \pm \sigma$ and close the positions once we reach the equilibrium μ. We detail the strategy in Algorithm 1. The code for this strategy can be found in Section 2.1.6.2.

Algorithm 1 Strategy 1

If current position is 0:
    If residual level is $> \mu + \sigma$:
        Take a short position of -1.
    Else if residual level is $< \mu - \sigma$:
        Take a long position of +1.
    Else:
        Keep a 0 position.
Else if current position is -1:
    If residual level is $< \mu - \sigma$:
        Close the short position to 0.
    Else:
        Keep the position at -1.
Else if current position is 1:
    If residual level is $> \mu + \sigma$:
        Close the long position to 0.
    Else:
        Keep the position at +1.

We can now see backtest this strategy using σ_{Eq} and also calculate the optimal σ. From this, we obtain Figures 6 and 7 respectively. The red line which represents position takes a value of μ is position is 0, value of $\mu + \sigma$ if the position is is +1 and a value of $\mu - \sigma$ is the position is -1.

Figure 6: Backtesting results from strategy 1 with σ_{Eq}.

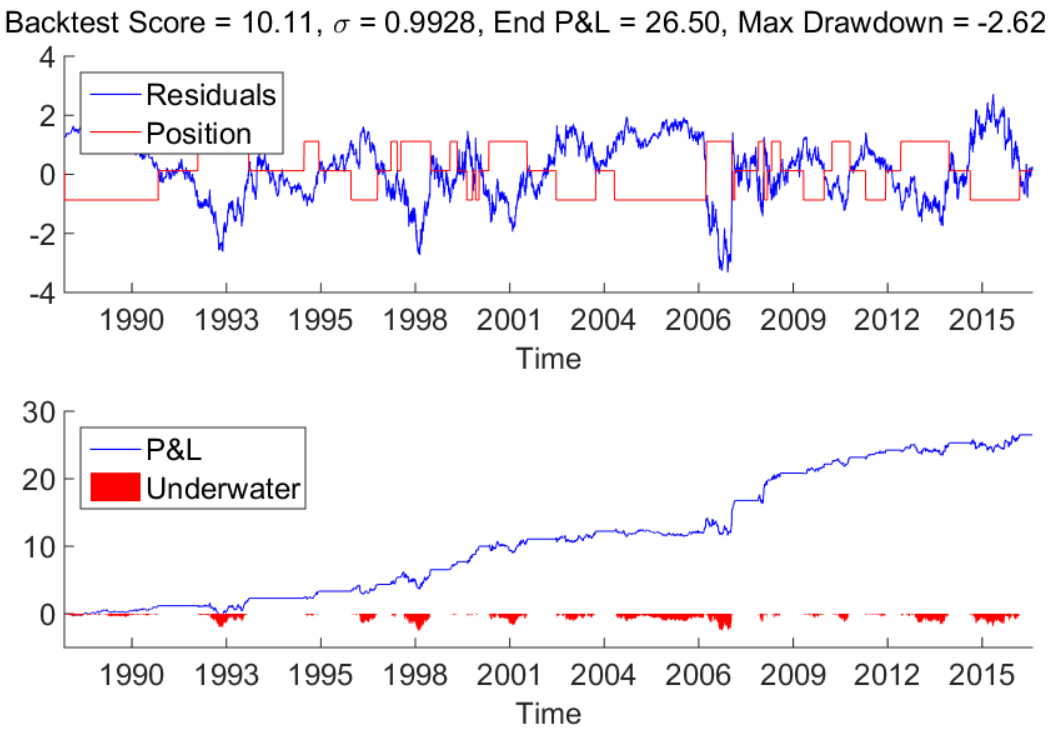
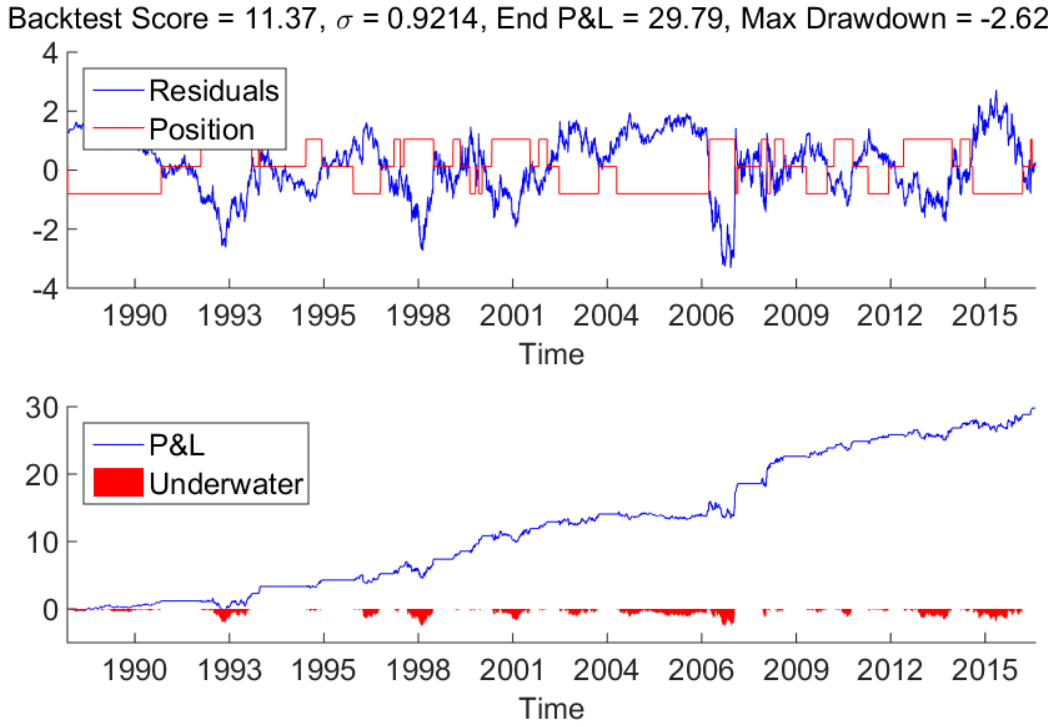


Figure 7: Backtesting results from strategy 1 with optimal σ.



Let us make some remarks for Strategy 1:

- From Figures 6 and 7, it is clear that optimising σ makes a big difference as the maximum drawdown is the same for both σ but the P&L increases by about 10%.
- In this strategy, the optimal σ is lower than σ_{Eq}.
- One of the issues with this strategy is that there are long time periods, for example 1991-1992 and 1994-1995, where no position is held.
- Not getting any returns for years at a time could be a problem when managing money unless we are running multiple of these types of strategies in the hope for diversification.
- Having a maximum drawdown of around -2.62 is around 10% of our end P&L which is quite high.

1.4.2 Strategy 2

This is a similar strategy for the first one. In this strategy, we take positions at $\mu \pm \sigma$ and close the positions at the opposite side $\mu \mp \sigma$. We detail the strategy in Algorithm 2. The code for this strategy can be found in Section 2.1.6.2.

Algorithm 2 Strategy 2

If current position is 0:
    If residual level is $> \mu + \sigma$:
        Take a short position of -1.
    Else if residual level is $< \mu - \sigma$:
        Take a long position of +1.
    Else:
        Keep a 0 position.
Else if current position is -1:
    If residual level is $< \mu - \sigma$:
        Close the short position to 0.
    Else:
        Keep the position at -1.
Else if current position is 1:
    If residual level is $> \mu + \sigma$:
        Close the long position 0.
    Else:
        Keep the position at +1.

We can now see backtest this strategy using σ_{Eq} and also calculate the optimal σ. From this, we obtain Figures 8 and 9 respectively. The red line which represents position takes a value of μ is position is 0, value of $\mu + \sigma$ if the position is is +1 and a value of $\mu - \sigma$ is the position is -1.

Figure 8: Backtesting results from strategy 2 with σ_{Eq}.

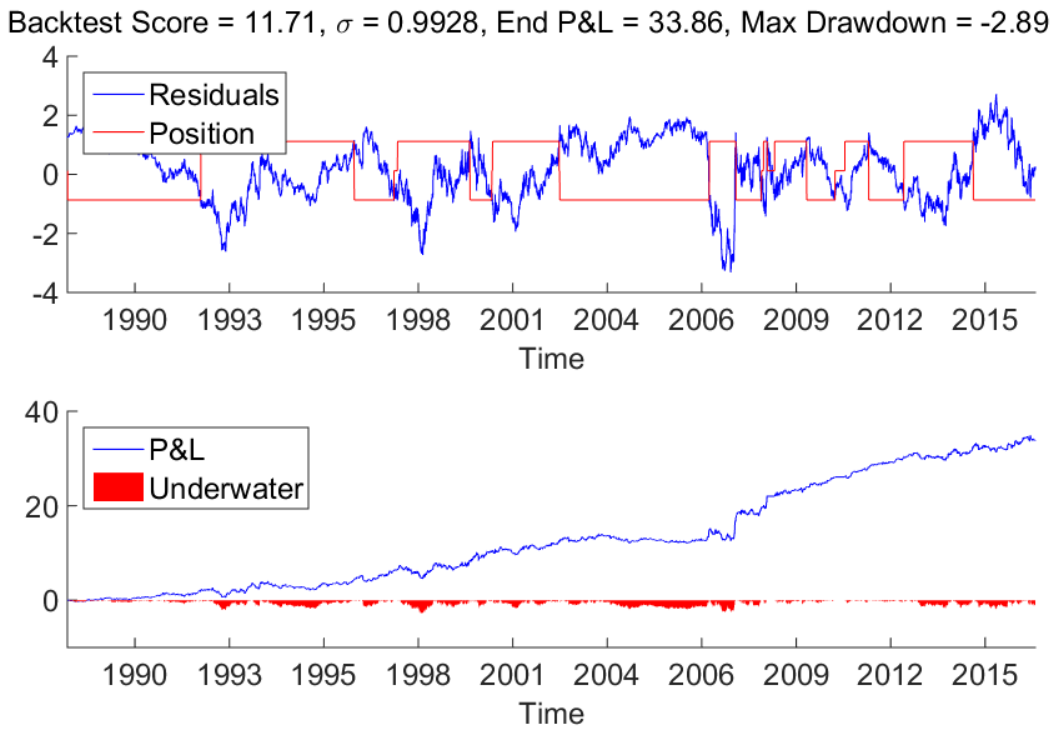
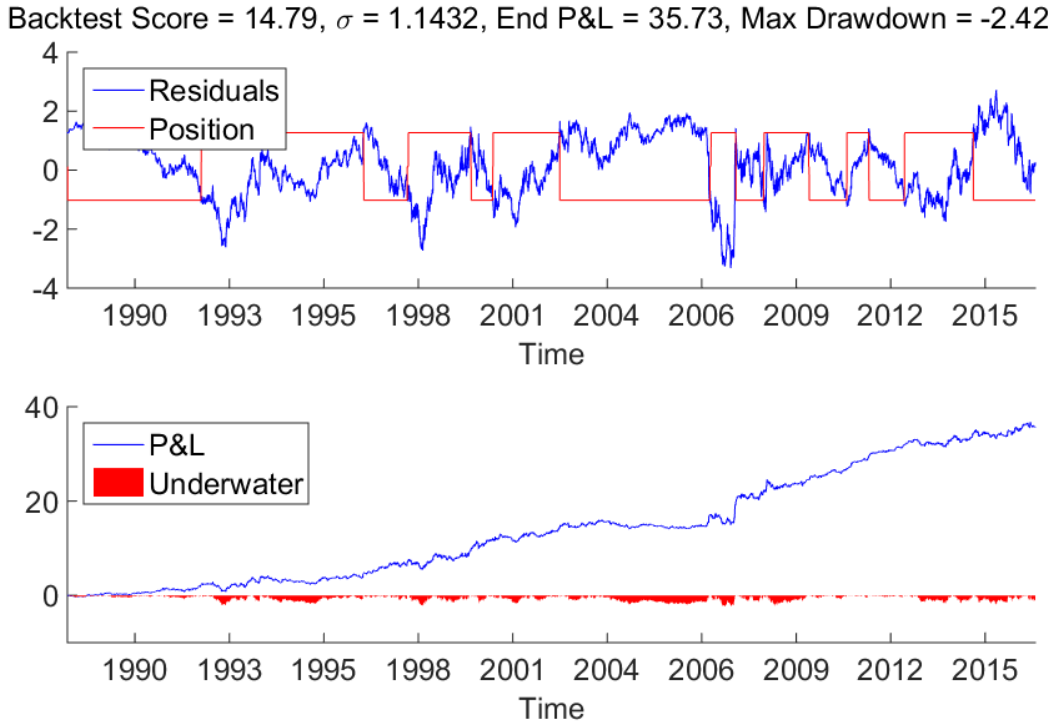


Figure 9: Backtesting results from strategy 2 with optimal σ.



Let us make some remarks for Strategy 2:

- From Figures 8 and 9, it is clear that optimising σ makes a big difference as the maximum drawdown is decreased and the P&L is increased.
- This strategy is different to strategy one where because the position is closed near to where a new position will normally be entered into, there are hardly any times where position is kept at 0.
- However, this means that there are times where we hold the same position for over 4 years, for example 2002-2006 where we are underwater. When sometimes, we would have closed the position at equilibrium and locked in some P&L.
- It is clear that mean reverting strategies such as these make P&L when there are large swings such as in the time period 2007-2011.
- Comparing Strategies 1 and 2; Strategy 2 has done better in the backtest scores for both σ's by both increasing the end P&L and reducing the maximum drawdown.

1.4.3 Strategy 3

This is a very aggresive strategy. In this strategy, we take positions at $\mu \pm \sigma$ then we take a bigger position if the residual level goes against us and close the positions at the opposite side $\mu \mp \sigma$. We detail the strategy in Algorithm 3. The code for this strategy can be found in Section 2.1.6.2.

Algorithm 3 Strategy 3

Set next position to current position, however:
If current position is 0:
    If residual level is $> \mu + \sigma$:
        Take a short position of -1.
    Else if residual level is $< \mu - \sigma$:
        Take a long position of $+1$.
Else if current position is < 0:
    If residual level is $< \mu - \sigma$:
        Close the short position to 0.
    Else if residual level is $> \mu + (i \times \sigma)$:
        Increase the short position to $-i$.
Else if current position is > 0:
    If residual level is $> \mu + \sigma$:
        Close the long position to 0.
    Else if residual level is $< \mu - (i \times \sigma)$:
        Increase the long position to $+i$.

We can now see backtest this strategy using σ_{Eq} and also calculate the optimal σ. From this, we obtain Figures 10 and 11 respectively. The red line which represents position takes a value of μ is position is 0, value of $\mu + (i \times \sigma)$ if the position is $+i$ and a value of $\mu - (i \times \sigma)$ is the position is $-i$.

Figure 10: Backtesting results from strategy 3 with σ_{Eq}.

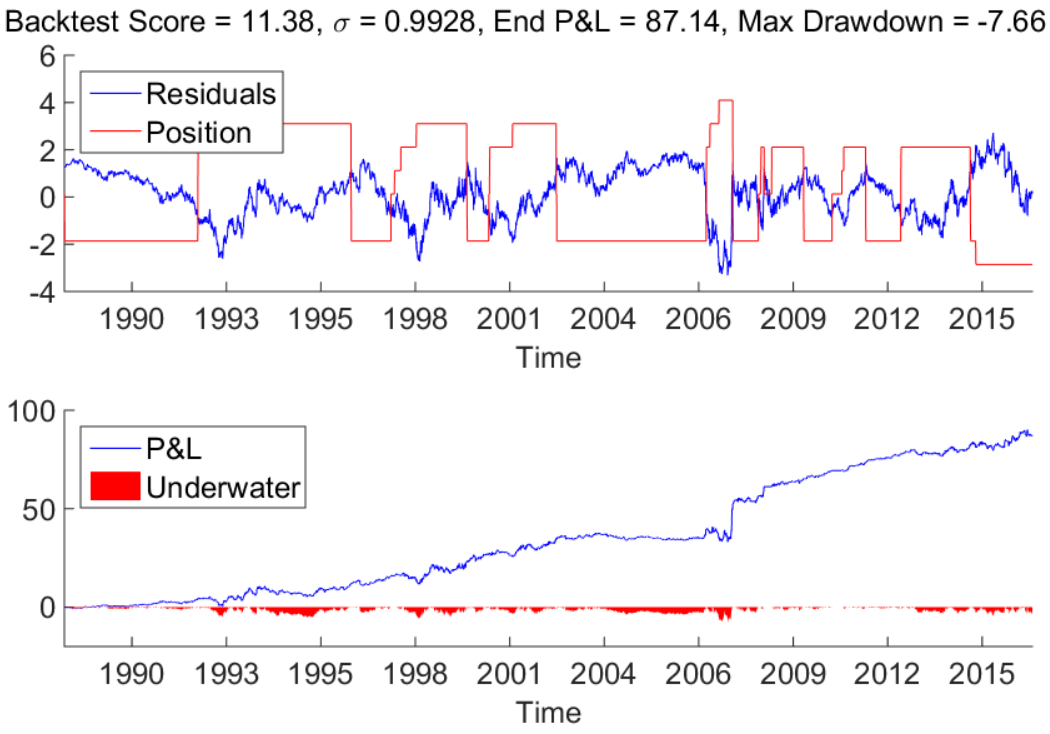
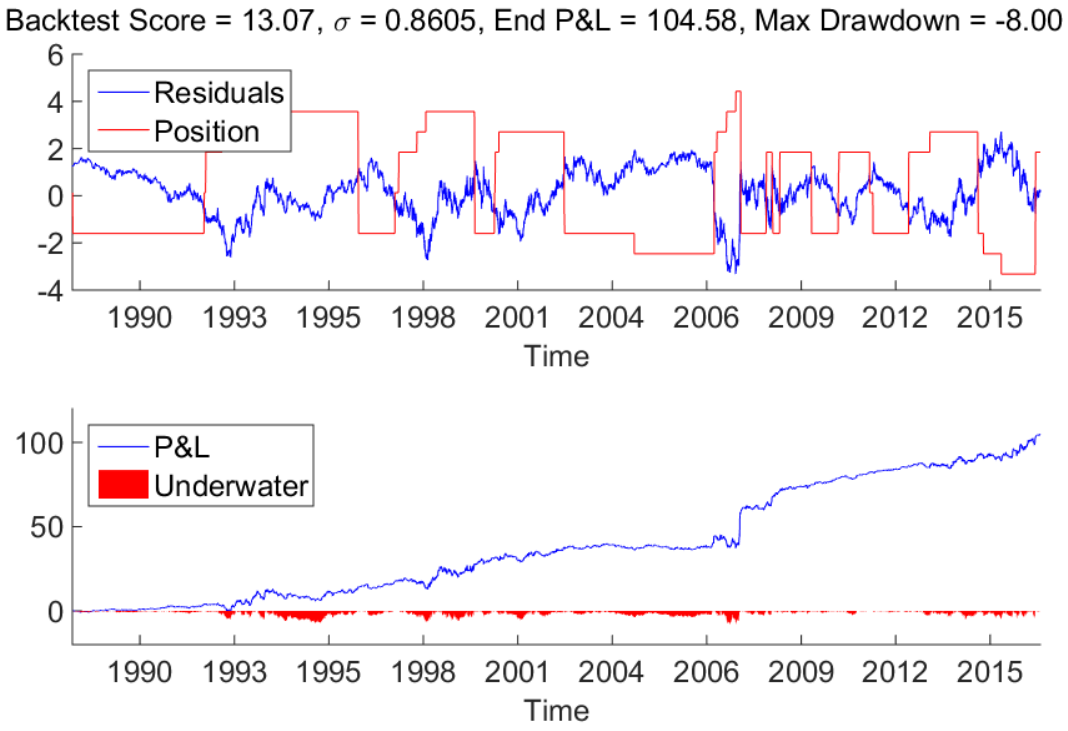


Figure 11: Backtesting results from strategy 3 with optimal σ.



Let us make some remarks for Strategy 3:

- From Figures 10 and 11, it is clear that optimising σ makes a difference as the P&L is increased by 20%, however the drawdown also increased slightly.
- This strategy is similar to Strategy 2 as we will hardly have 0 position but this strategy is more aggressive, taking larger positions when the residual level goes against the strategy.
- This strategy, like the other two has a similar problem that when volatility is low, it does not do as well and there can be large time periods when this is the case, as seen between 2002-2006.
- Comparing strategies; Strategy 3 has done better in the backtest scores for both σ's by both increasing the end P&L than strategy. Hoever, compared to Strategy 2, Strategy 3 is still not as good when it comes to the backtest scores.
- The reason for this strategy was to leverage up on the times when there is high volatility and large swings in the hope that it would beat the times of low volatility. The end P&L is more than double of Strategy 2, however, the maximum drawdown is more than triple of Strategy 2. This shows that it is similar to the effect of leveraging up any normal strategy.
- Therefore, it is important to look at the ratio of return and risk to be able to compare like for like.

1.4.4 Transaction Costs Analysis

It is important to have an idea of transactions costs that we would be facing when we are back-testing our strategies. We will take a simple approach of getting charged a flat fee proportional to the change in the absolute size of our position.

As part of the class `BacktestData`, we have a function called `adjustForTrans`, the code for which can be seen in Section 2.1.6.1.

In Figure 12, we can see how transaction costs affects our P&L for Strategy 2 with optimal σ.

Figure 12: Effect of transaction costs on P&L for Strategy 2 with optimal σ.

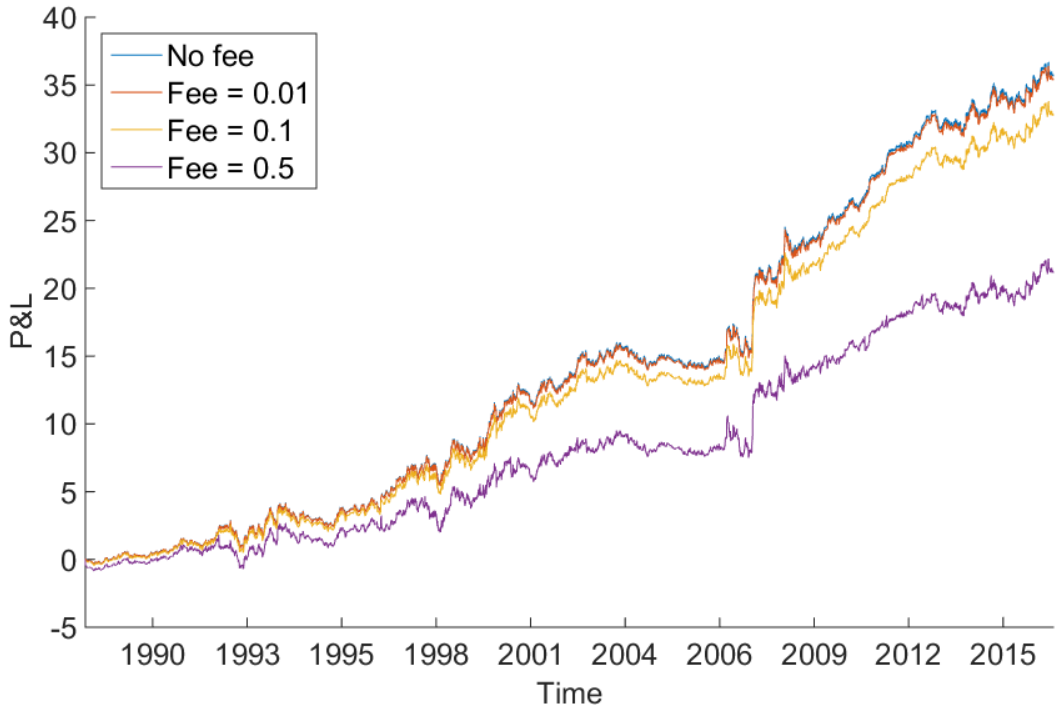


Table 9: Effect of transaction costs on backtest scores for Strategy 2 with optimal σ.

No Fee	1%	10%	50%
14.79	14.67	13.59	6.78

From Figure 12 and Table 9, it is clear that transaction costs have a big effect on our P&L and on our back test scores which more than halve big 50% flat fees on the size of our positions. Transaction costs are unlikely to be so big however it is good to know the extent to which our P&L gets affected by transaction costs. We now do the same but for Strategy 1.

Figure 13: Effect of transaction costs on P&L for Strategy 1 with optimal σ.

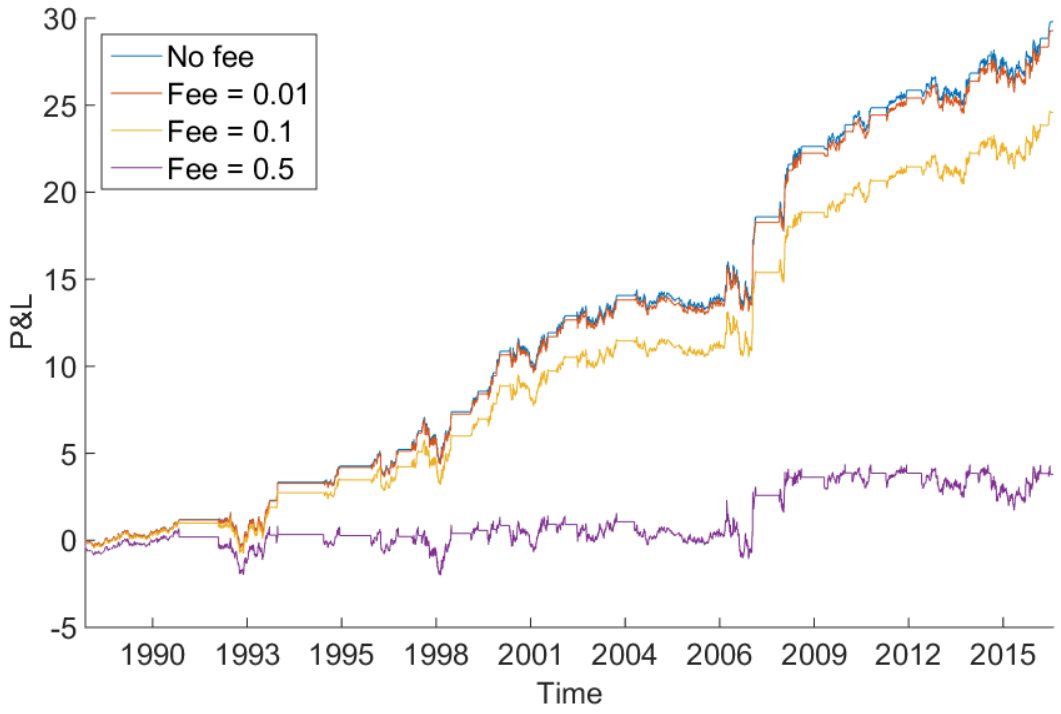


Table 10: Effect of transaction costs on backtest scores for Strategy 1 with optimal σ.

No Fee	1%	10%	50%
11.37	11.17	9.39	1.12

From Figure 13 and Table 10, it looks like Strategy 1 can get affected largely by transaction costs due to its slow gain in P&L. It is interesting as this takes positions than Strategy 2 but the losses from the transaction costs seem have a greater effect than the fewer changes in positions. We can see that with 50% fees, the P&L is close to zero and more often negative than positive. The only reason P&L is in green at the end is because of the large swings around 2007.

1.5 Further Considerations

There are many parts of the project I would like to work further on and hope to do in the future.

- When doing the backtesting, I would want to split the data into two parts, one for testing for cointegration and fitting the Ornstein-Uhlenbeck process then the other part of the data (would be the larger part) to be able to backtest on. This means that we can see if our hypothesised strategy works in the out of sample or 'unknown' data.
- In terms of testing for cointegration, I would want to make the code flexible enough so that I pass in many time series at once then the code would check for cointegration on all the

possible combinations of the time series. Then we can backtest strategies on multiple time series.

- I would like to look at diversification of the the strategies so when there are large periods of low volatility in our strategy, there would be other strategies working on high volatility.
- Understanding costs for taking on positions and the capital required would be very interesting as there would be costs repo costs on the short positions and the rate we would have to pay to be able to borrow money.
- Finding factors for what drives the mean reverting spread would also be interesting and would have something to do with the volatility.
- I would like to calculate the α against a benchmark returns and other time series returns to see if the strategy has information on the market or it is plain luck driving the P&L.
- I would also like to come up with more strategies which could take advantage of periods of the low volatility adding further to the P&L.

2 Appendix

2.1 Time Series Code

2.1.1 testScript

```
%% Load data------------------------------------------------------------------
load('spmData.mat');

%% TimeSeries Initialisation---------------------------------------------------
% Make time series
timeSeries(1,1) = TimeSeries(tescos);
timeSeries(2,1) = TimeSeries(sainsburys);
timeSeries(3,1) = TimeSeries(mands);

%% TimeSeriesCollection Initialisation-----------------------------------------
% Make time series collection
spm = TimeSeriesCollection(timeSeries);
spm.plot();
% Do the ADF tests for all three models and check for determinisitc trend
[h,pValue,stat,cValue,reg] = spm.adftest('model',{'AR';'ARD'});

%% Vector Autoregresson--------------------------------------------------------
% Get optimal lag
p = spm.getOptimalLag(1,30);
% Run vector autoregression
[BHat,epsHat,SigmaHat,errorStats] = spm.vecAutoreg(p);
% Get correlation matrix from covariance matrix
corrHat = spm.cov2corr(SigmaHat);
% Q-Q plot
figHandle = figure;
qqplot(epsHat')
legend('Tescos','Sainsburys','Marks & Spencers','location','southeast')
changeFig(figHandle)

%% Johansen's Test-------------------------------------------------------------
% Run Johansen's test from TimeSeriesCollection which outputs JCData object
cointData = spm.jcitest('display','off');

%% Backtesting-----------------------------------------------------------------
% Make BacktestCoint objects which in turn hold BacktestData objects when
% backtest() is called
for i = 1:cointData.maxRank
    bt(1,i) = BacktestCoint(cointData.time,cointData.residuals(:,i));
    bt(1,i) = bt(1,i).backtest();
end

%% Transaction Costs Analysis--------------------------------------------------
btdata = bt(1,1).btData{2,1};
btdata = btdata.adjustForTrans([0.01 0.1 0.5]);
```

2.1.2 TimeSeries

```
classdef TimeSeries
    % TimeSeries object to hold a single time series and be able to produce
    % different results and properties of that time series

    properties
        time        % Array of times
        series      % Array of prices
    end % properties

    methods
        %%---------------------------------------------------------------------
        % Constructor-----------------------------------------------------------
        function obj = TimeSeries(varargin)

            % If non empty constructor
            if nargin > 0

                if nargin > 2
                    error('Too many input arguments.');
                elseif nargin == 2
                    % Data validation
                    if size(varargin{1},2) ~= 1 || size(varargin{2},2) ~=1
                        error('Input arrays are not column vectors');
                    end
                    if size(varargin{1},1) ~= size(varargin{2},1)
                        error('Number of rows both arrays are different.');
                    end
                    % Put the arrays together
                    try
                        varargin{1} = [varargin{1} varargin{2}];
                    catch
                        error('Input arrays are not both numeric');
                    end
                    varargin{2} = [];
                elseif nargin == 1
                    % Data validation
                   if size(varargin{1},2) ~= 2
                       error('Input array does not have two columns.');
                   end
                end % if-else

                % Data validation
                if ~isnumeric(varargin{1})
                    error('Input array or arrays are not numeric');
                end
                varargin{1}(:,1) = obj.checkTime(varargin{1}(:,1));

                % Sort time
                [sortedTime,sortedIndices] = sort(varargin{1}(:,1));
                % Set time
                obj.time = sortedTime;
                sortedSeries = varargin{1}(sortedIndices,2);
```

```matlab
                obj.series = obj.replaceNaN(sortedTime,sortedSeries);
                obj.series = obj.replaceZeros(sortedTime,obj.series);
            end

        end % Constructor

        %%-----------------------------------------------------------------
        % Plotting function-----------------------------------------------
        function plot(varargin)
            if nargin == 1 || (nargin == 2 && strcmpi(varargin{2},'levels'))
                figHandle = figure;
                plot(varargin{1}.time,varargin{1}.series);
                title('Time Series Levels');
                ylabel('Levels');
            elseif nargin == 2 && strcmpi(varargin{2},'returns')
                figHandle = figure;
                plot(varargin{1}.time(2:end),...
                    log(varargin{1}.series(2:end)./...
                    varargin{1}.series(1:end-1)));
                title('Time Series Returns');
                ylabel('Returns');
            else
                error('Input arguments are not right');
            end % if-else
            xlabel('Time');
            yLimits = ylim;
            axis([varargin{1}.time(1),varargin{1}.time(end),...
                yLimits(1),yLimits(2)]);
            changeFig(figHandle);
            datetick('x','yyyy','keeplimits','keepticks')
        end % plot

        %%-----------------------------------------------------------------
        % Calculate volatility--------------------------------------------
        function returns = calcReturns(obj)
            % Calculate log return
            returns = log(obj.series(2:end)./obj.series(1:end-1));
        end % calcVol

        %%-----------------------------------------------------------------
        % Calculate volatility--------------------------------------------
        function volTimeSeries = calcVol(obj,N)
            % Calculate log return
            logReturn = obj.calcReturns();
            vol = zeros(size(logReturn,1)-N+1,1);
            % Loop through to calculate volatility
            for i = 1:size(vol,1);
                vol(i) = sqrt(252)*std(logReturn(i:i+N-1));
            end
            % Make time series
            volTimeSeries = TimeSeries();
            volTimeSeries.time = obj.time(N+1:end);
            volTimeSeries.series = vol;
        end % calcVol

    end % methos
```

```
109
110     methods (Access = private)
111         %%-------------------------------------------------------------------
112         % Function to change to Matlab dates and handle NaN values----------
113         function outputTime = checkTime(obj,inputTime)
114             % Check if excel dates or matlab dates
115             if inputTime(1) < 100e3
116                 inputTime = x2mdate(inputTime);
117             end
118             % Deal with NaN values
119             outputTime = obj.replaceNaN((1:size(inputTime,1))',inputTime);
120         end % checkTime
121
122         %%-------------------------------------------------------------------
123         % Function to handle NaN values--------------------------------------
124         function outputYAxis = replaceNaN(~,inputXAxis,inputYAxis)
125             if any(isnan(inputYAxis))
126                 notNaNIndices = ~isnan(inputYAxis);
127                 outputYAxis = interp1(inputXAxis(notNaNIndices),...
128                 inputYAxis(notNaNIndices),inputXAxis);
129             else
130                 outputYAxis = inputYAxis;
131             end
132         end % replaceNaN
133
134         %%-------------------------------------------------------------------
135         % Function to handle zero values-------------------------------------
136         function outputYAxis = replaceZeros(~,inputXAxis,inputYAxis)
137             if any(inputYAxis==0)
138                 notZeroIndices = inputYAxis~=0 ;
139                 outputYAxis = interp1(inputXAxis(notZeroIndices),...
140                     inputYAxis(notZeroIndices),inputXAxis);
141             else
142                 outputYAxis = inputYAxis;
143             end
144         end % replaceNaN
145
146     end % private methods
147
148 end % classdef
```

2.1.3 TimeSeriesCollection

```
1  classdef TimeSeriesCollection
2      % TimeSeriesCollection object to hold multiple time series with the
3      % same time array associated with each series
4
5      properties
6          time            % Array of times which are associated to all the series
7          numSeries       % Number of different series
8          numTimes        % Number of time points
9          multiSeries     % All the series stored together
10         multiReturns    % All the returns stored together
```

```
11          reIndexedSeries % Re-indexed series starting from one
12      end % properties
13
14      methods
15          %%-------------------------------------------------------------------------
16          % Constructor-------------------------------------------------------------
17          function obj = TimeSeriesCollection(varargin)
18
19              % Cannot have an empty constructor
20              if nargin == 0
21                  error('Need at least one TimeSeries as input.');
22              end
23
24              % Check if all inputs are TimeSeries
25              counter = 0;
26              for i = 1:nargin
27                  if ~isa(varargin{i},'TimeSeries')
28                      error('All inputs need to be TimeSeries.');
29                  end
30                  % Turn into column vector
31                  varargin{i} = varargin{i}(:);
32                  % Add to the counter to calculate number of series
33                  counter = counter + size(varargin{i},1);
34              end
35
36              % Set numSeries
37              obj.numSeries = counter;
38
39              % Store all the times and series'
40              allTime = cell(1,obj.numSeries);
41              allSeries = cell(1,obj.numSeries);
42              counter = 1;
43              for i = 1:nargin
44                  for j = 1:size(varargin{i},1)
45                      allTime{counter} = varargin{i}(j).time;
46                      allSeries{counter} = varargin{i}(j).series;
47                      counter = counter + 1;
48                  end
49              end
50
51              % Intersect times
52              intersectTime = allTime{1,1};
53              if obj.numSeries > 1
54                  for i = 2:obj.numSeries
55                      intersectTime = intersect(intersectTime,...
56                          allTime{1,i},'stable');
57                  end
58              end
59
60              % Set time
61              obj.time = intersectTime;
62              obj.numTimes = size(intersectTime,1);
63
64              % Get the right values from series
65              finalSeries = zeros(size(obj.time,1),obj.numSeries);
66              for i = 1:obj.numSeries
```

```
67                  % Get indices from intersection
68                  [~,~,tempIndices] = intersect(obj.time,allTime{i});
69                  % Get the corresponding series values
70                  finalSeries(:,i) = allSeries{i}(tempIndices,1);
71              end
72              % Set multiSeries
73              obj.multiSeries = finalSeries;
74              % Set multiReturns
75              obj.multiReturns = log(finalSeries(2:end,:)...
76                  ./finalSeries(1:end-1,:));
77              obj.reIndexedSeries = [ones(1,obj.numSeries);...
78                  cumprod(exp(obj.multiReturns))];
79
80          end % Constructor
81
82          %%-------------------------------------------------------------------
83          % Function to plot-------------------------------------------------------
84          function plot(varargin)
85              if nargin == 1
86                  figHandle = figure;
87                  hold on;
88                  for i = 1:varargin{1}.numSeries
89                      plot(varargin{1}.time,varargin{1}.reIndexedSeries(:,i));
90                  end
91                  hold off;
92                  title('Re-Indexed Series');
93                  xlabel('Time');
94                  yLimits = ylim;
95                  axis([varargin{1}.time(1),varargin{1}.time(end),...
96                      yLimits(1),yLimits(2)]);
97                  changeFig(figHandle);
98                  datetick('x','yyyy','keeplimits','keepticks')
99              else
100                 error('No input argument required.');
101             end % if-else
102         end % plot
103
104         %%-------------------------------------------------------------------
105         % Function for vector auto regression with lag p-----------------------
106         [BHat,epsHat,SigmaHat,errorStats] = vecAutoreg(obj,p)
107
108         %%-------------------------------------------------------------------
109         % Stability condition----------------------------------------------------
110         stable = stabCond(obj,BHat)
111
112         %%-------------------------------------------------------------------
113         % Akaike information criterion-------------------------------------------
114         statistic = statAIC(obj,SigmaHat,p)
115
116         %%-------------------------------------------------------------------
117         % Schwartz criterion or Bayesian information criterion-----------------
118         statistic = statSC(obj,SigmaHat,p)
119
120         %%-------------------------------------------------------------------
121         % Function to find optimal lag-------------------------------------------
122         p = getOptimalLag(obj,minLag,maxLag)
```

```
123
124         %%-------------------------------------------------------------------
125         % Function to do the Augmented Dickey Fuller test-------------------
126         function [h,pValue,stat,cValue,reg] = adftest(obj,varargin)
127
128             % Initialise variables
129             h = cell(1,obj.numSeries);
130             pValue = cell(1,obj.numSeries);
131             stat = cell(1,obj.numSeries);
132             cValue = cell(1,obj.numSeries);
133             reg = cell(1,obj.numSeries);
134             % Loop through each series to run adftest
135             for i = 1:obj.numSeries
136                 [h{i},pValue{i},stat{i},cValue{i},reg{i}] = adftest(...
137                     obj.multiSeries(:,i),varargin{:});
138             end
139
140         end % adftest
141
142         %%-------------------------------------------------------------------
143         % Function to run Johansen's test------------------------------------
144         function cointData = jcitest(obj,varargin)
145             % Make cointJCData object and run Johansen's test
146             cointData = JCData(obj.time,obj.multiSeries,varargin{:});
147
148         end % jcitest
149
150     end % methods
151
152     methods(Static)
153         %%-------------------------------------------------------------------
154         % Function to turn covariance matrix into a correlation matrix-------
155         function corr = cov2corr(Sigma)
156             corr = Sigma./kron(sqrt(diag(Sigma)),sqrt(diag(Sigma))');
157         end % cov2corr
158
159     end % static methods
160
161 end % classdef
```

2.1.3.1 vecAutoReg

```
1  function [BHat,epsHat,SigmaHat,errorStats] = vecAutoreg(obj,p)
2  % Function for vector auto regression with lag p
3
4      % Build matrix Y which are (p+1)th to end returns and has
5      % numSeries rows
6      R = obj.multiReturns(p+1:end,:)';
7      % Initialise matrix Z with top row as ones, has p*numSeries + 1
8      % rows and number of returns - p columns
9      Z = ones(1+p*obj.numSeries,(obj.numTimes-1)-p);
10     for i = 1:p
11         % Starting with times p to end -1 shift back one to get to
```

```
12          % 1 to end - p and store going downwards
13          Z(2+(i-1)*obj.numSeries:1+i*obj.numSeries,:) =...
14              obj.multiReturns(p-i+1:end-i,:)';
15      end
16
17      % Calculate matrix BHat
18      BHat = R*Z'/(Z*Z');
19      % Calculate matrix eps_hat
20      epsHat = R - BHat*Z;
21      % Calculate covariance matrix covHat
22      SigmaHat = epsHat*epsHat'/(size(epsHat,2));
23
24      errorNames = {'MeanError','RootMeanSquaredError',...
25          'MeanAbsoluteError','MeanPercentageError',...
26          'MeanAbsolutePercentageError','MeanAbsoluteScaledError'};
27
28      % Error statistics
29      % Mean error
30      ME = mean(epsHat,2);
31      % Root mean squared error
32      RMSE = sqrt(mean(epsHat.^2,2));
33      % Mean absolute error
34      MAE = mean(abs(epsHat),2);
35      percError = epsHat./R;
36      % Mean percentage error
37      MPE = nanmean(percError,2);
38      % Mean absolute percentage error
39      MAPE = mean(abs(percError),2);
40      % Mean absolute scaled error
41      MASE = bsxfun(@rdivide,MAE,mean(abs(diff(R'))')');
42
43      errorStats = table(ME,RMSE,MAE,MPE,MAPE,MASE,...
44          'VariableNames',errorNames);
45  end % vecAutoreg
```

2.1.3.2 getOptimalLag

```
1   function p = getOptimalLag(obj,minLag,maxLag)
2   % Function to find optimal lag
3       % Error if minLag is greater then maxLag
4       if minLag > maxLag
5           error('Minimum lag is greater than maximum lag.');
6       end
7       % Initialise variables
8       allLag = (minLag:maxLag)';
9       stats = [allLag zeros(size(allLag,1),2)];
10      stability = zeros(size(allLag,1),1);
11      % Loop through differest lags to get stability and statistics
12      for i = allLag'
13          % Get the covariance matix from vector autoregression
14          [BHat,~,SigmaHat] = vecAutoreg(obj,i);
15          % Get statistics
16          stats(i,2) = obj.statAIC(SigmaHat,i);
```

```
17          stats(i,3) = obj.statSC(SigmaHat,i);
18          % Get stability
19          stability(i) = obj.stabCond(BHat);
20      end
21      % Delete instable lags
22      stats(stability==0,:) = [];
23      % Get the minimum statistic
24      [~,lagAICIndex] = min(stats(:,2));
25      [~,lagSCIndex] = min(stats(:,3));
26      % Ouput the right lag
27      if lagAICIndex == lagSCIndex
28          p = stats(lagAICIndex,1);
29      else
30          % Give priority to the AIC test
31          if stats(lagAICIndex,2) < stats(lagSCIndex,3)
32              p = stats(lagAICIndex,1);
33          else
34              p = stats(lagSCIndex,1);
35          end
36      end % if-else
37  end % findOptimalLag
```

2.1.3.3 stabCond

```
1   function stable = stabCond(obj,BHat)
2   % Stability condition
3       % Get p
4       p = (size(BHat,2)-1)/obj.numSeries;
5       % Initialise stability to 1
6       stable = 1;
7       % Check for each BHat^p the absolute values of the eigenvalues
8       for i = 1:p
9           % Get the eigenvalues
10          eValues = ...
11              eig(BHat(:,2+(i-1)*obj.numSeries:1+i*obj.numSeries));
12          % Check if absolute value of any eValues > 1
13          if any(abs(eValues) > 1)
14              stable = 0;
15              disp([num2str(i) '-th lag has unstable eigenvalues']);
16          end %if
17      end
18  end % stabCond
```

2.1.3.4 statAIC

```
1   function statistic = statAIC(obj,SigmaHat,p)
2   % Akaike information criterion
3       % log(|SigmaHat|) + 2*(n*(n*p+1))/T
4       T = (obj.numTimes-1) - p;
5       statistic = log(det(SigmaHat)) +...
```

```
6           2*(obj.numSeries*(obj.numSeries*p + 1))/T;
7   end % statAIC
```

2.1.3.5 statSC

```
1   function statistic = statSC(obj,SigmaHat,p)
2   % Schwartz criterion or Bayesian information criterion
3       % log(|SigmaHat|) + (n*(n*p+1))/T*log(T)
4       T = (obj.numTimes-1) - p;
5       statistic = log(det(SigmaHat)) +...
6           (obj.numSeries*(obj.numSeries*p + 1))...
7           /T*(log(T));
8   end % statSC
```

2.1.4 JCData

```
1   classdef JCData
2       % cointJCData holds all the necessary data from the Johansen's test
3
4       properties
5           time            % Time
6           multiSeries     % Holds all the time series including ones
7           maxRank         % Maximum rank
8           alpha           % Cointegration alpha
9           beta            % Cointegration weights
10          residuals       % residualss from the cointegrated weights
11          otherData       % rank, h, pval, stat, cval, eigval
12          eigVal          % Eigenvalue
13      end % properties
14
15      methods
16          %%-------------------------------------------------------------------
17          % Constructor-------------------------------------------------------
18          function obj = JCData(time,multiSeries,varargin)
19              % Initialise time series variables
20              obj.time = time;
21              numTimes = size(multiSeries,1);
22              numSeries = size(multiSeries,2);
23              obj.multiSeries = [ones(numTimes,1) multiSeries];
24
25              % Run the Johansen's test for both trace and maxeig
26              [h,pValue,stat,cValue,mles] = jcitest(...
27                  multiSeries,'test',{'trace','maxeig'},varargin{:});
28
29              % Get the ranks of matrix
30              traceRank = sum(h{1,:});
31              maxEigRank = sum(h{2,:});
32
33              % Display results
34              disp(['Trace test: The null hypothesis for r = '...
```

```
            num2str(traceRank) ' was not rejected.']);
        disp(['Max eigenvalue test: The null hypothesis for r = '...
            num2str(maxEigRank) ' was not rejected.']);

        % Get the maximum rank out of the two tests
        obj.maxRank = max(traceRank,maxEigRank);

        % Get alpha and beta where beta includes the intercept
        obj.alpha = mles{1,obj.maxRank+1}.paramVals.A;
        obj.beta = [mles{1,obj.maxRank+1}.paramVals.c0';...
            mles{1,obj.maxRank+1}.paramVals.B];

        % Get residuals
        obj.residuals = obj.multiSeries*obj.beta;

        % Get the eigenvalues
        obj.eigVal = zeros(numSeries,1);
        for i = 1:numSeries
            obj.eigVal(i) = mles{1,i}.eigVal;
        end

        % Get other data
        obj.otherData = cell(2,2);
        obj.otherData{1,1} = 'Trace test';
        obj.otherData{2,1} = 'Max eigenvalue test';
        dataNames = {'rank','nullRejected','pValue','stat',...
            'cValue','eigVal'};
        for i = 1:2
            obj.otherData{i,2} = table((0:numSeries-1)',h{i,:}',...
                pValue{i,:}',stat{i,:}',cValue{i,:}',obj.eigVal,...
                'variableNames',dataNames);
        end

        % Plot
        obj.plot;

    end % Constructor

    %%-------------------------------------------------------------------
    % Plotting function-------------------------------------------------
    function plot(obj)

        figHandle = figure;
        plot(obj.time,obj.residuals);
        title('Cointegration residuals');
        xlabel('Time');
        yLimits = ylim;
        axis([obj.time(1),obj.time(end),yLimits(1),yLimits(2)]);
        changeFig(figHandle);
        datetick('x','yyyy','keeplimits','keepticks')

    end % plot

    %%-------------------------------------------------------------------
    % Function to do the Augmented Dickey Fuller test------------------
    function [h,pValue,stat,cValue,reg] = adftest(obj,varargin)
```

```
91
92              id = 'econ:adftest:LeftTailStatTooSmall';
93              warning('off',id);
94
95              numSeries = size(obj.residuals,2);
96
97              % If no series as input then run adftest on every series
98              if nargin == 1
99                  % Initialise variables
100                 h = cell(1,numSeries);
101                 pValue = cell(1,numSeries);
102                 stat = cell(1,numSeries);
103                 cValue = cell(1,numSeries);
104                 reg = cell(1,numSeries);
105                 % Loop through each series to run adftest
106                 for i = 1:numSeries
107                     [h{i},pValue{i},stat{i},cValue{i},reg{i}] = adftest(...
108                         obj.residuals(:,i));
109                 end
110
111             % If a series is given to run adftest on
112             elseif nargin == 2
113                 % Data validation
114                 if varargin{1} < 1 || varargin{1} > obj.numSeries
115                     error('Input series is not available.');
116                 end
117                 % Run adftest on that series
118                 [h,pValue,stat,cValue,reg] = adftest(...
119                     obj.residuals(:,varargin{1}));
120
121             % Otherwise too many inputs
122             else
123                 warning('on',id);
124                 error('Too many input arguments.');
125             end
126             warning('on',id);
127
128         end % adftest
129
130     end % methods
131
132 end % classdef
```

2.1.5 BacktestCoint

```
1  classdef BacktestCoint
2      % BacktestCoint class to hold the stationary time series and do
3      % backtesting and calculating P&L
4
5      properties
6          time                % Times corresponding to time series
7          residuals           % Time series of residuals
8          theta               % theta from OU process
```

```
        mu                  % mu from OU process
        sigmaOU             % sigma from OU process
        sigmaEq             % sigma from equilibrium for entry/exit
        halfLife            % half life tau calculated from theta
        btData              % Backtesting data
        sigmaOptimal        % Optimal sigma (bounds for entry/exit)

    end % properties

    methods
        %%-------------------------------------------------------------------
        % Constructor-------------------------------------------------------
        function obj = BacktestCoint(time,residuals)
            obj.time = time;
            % Set the residuals
            obj.residuals = residuals;
            % Regress on the residuals with one lag
            mdl = fitlm(obj.residuals(1:end-1),obj.residuals(2:end));
            A = mdl.Coefficients.Estimate(1);
            B = mdl.Coefficients.Estimate(2);
            % Set the OU process constants, sigmaEq and halfLife
            tau = 1/252;
            obj.theta = -log(B)/tau;
            obj.mu = A/(1-B);
            obj.sigmaOU = sqrt(2*obj.theta*var(mdl.Residuals.Raw)...
                /(1-exp(-2*obj.theta*tau)));
            obj.sigmaEq = obj.sigmaOU/sqrt(2*obj.theta);
            obj.halfLife = log(2)/obj.theta;
        end % Constructor

        %%-------------------------------------------------------------------
        % Plotting----------------------------------------------------------
        function plot(obj)
            % Sample path
            tau = 1/252;
            samplePath = obj.sigmaOU*sqrt((1-exp(-2*obj.theta*tau))...
                /(2*obj.theta))*randn(size(obj.residuals));
            samplePath(1) = obj.residuals(1);
            for i = 2:size(samplePath,1)
                samplePath(i) = samplePath(i-1)*exp(-obj.theta*tau) + ...
                    obj.mu*(1-exp(-obj.theta*tau)) + samplePath(i);
            end

            % Plotting
            figHandle = figure;
            hold on;
            plot(obj.time,samplePath,'y');
            plot(obj.time,obj.residuals);
            plot(obj.time,obj.mu*ones(size(obj.residuals,1),1),'g');
            plot(obj.time,...
                (obj.mu+obj.sigmaEq)*ones(size(obj.residuals,1),1),'r');
            plot(obj.time,...
                (obj.mu-obj.sigmaEq)*ones(size(obj.residuals,1),1),'r');
            hold off;
            legend('Sample path','Residual','\mu','\mu\pm\sigma_{Eq}'...
                ,'location','southeast');
```

```
            xlabel('Time');
            ylabel('Residual spread');
            yLimits = ylim;
            changeFig(figHandle);
            axis([obj.time(1),obj.time(end),yLimits(1),yLimits(2)]);
            datetick('x','yyyy','keeplimits','keepticks')

        end % plot

        %%-------------------------------------------------------------------
        % Function to backtest-----------------------------------------------
        function obj = backtest(obj,varargin)

            numStrats = 3;
            tempCell = cell(2,numStrats);
            rowNames = {'sigmaEq';'optimalSigma'};
            variableNames = cell(1,numStrats);
            obj.sigmaOptimal = zeros(1,numStrats);
            for i = 1:3
                tempCell{1,i} = BacktestData(obj.sigmaEq,...
                    obj.mu,obj.time,obj.residuals,i);
                obj.sigmaOptimal(1,i) = obj.getOptimalSigma(i);
                tempCell{2,i} = BacktestData(obj.sigmaOptimal(1,i),...
                    obj.mu,obj.time,obj.residuals,i);
                variableNames{1,i} = ['Strategy' num2str(i)];
            end

            obj.btData = cell2table(tempCell,'rowNames',rowNames,...
                'variableNames',variableNames);

        end % backtest

        %%-------------------------------------------------------------------
        % Function to get the optimal sigma----------------------------------
        sigma = getOptimalSigma(obj,varargin)

    end % methods

end % classdef
```

2.1.5.1 getOptimalSigma

```
function sigma = getOptimalSigma(obj,varargin)
% Function to get the optimal sigma

    % Make the objective function
    objFunc = @(sigma) optimiseSigma(sigma,obj.mu,obj.residuals,varargin{:});
    % Get the optimal sigma
    sigma = fminsearch(objFunc,obj.sigmaEq);

end % getOptimalSigma
```

2.1.5.2 optimiseSigma

```
function optimiseVal = optimiseSigma(sigma,mu,residuals,varargin)
% Function to optimise sigma
    positionArray = getPosition(sigma,mu,residuals,varargin{:});

    % pnl array
    pnlArray = cumsum(diff(residuals).*positionArray);

    [~,maxDrawdown] = getUnderwater(pnlArray);

    % Optimise value of profit at the end and max drawdown
    optimiseVal = pnlArray(end)/maxDrawdown;

end % optimiseSigma
```

2.1.6 BacktestData

```
classdef BacktestData
    % Backtest data container

    properties
        time                % Time
        position            % Time series of position on the spread
        pnl                 % P&L from the position
        underwater          % Underwater time series
        maxDrawdown         % Maximum drawdown
        backtestScore       % End pnl/maxdrawdown
        cumTransactions     % Cumulative transactions
        pnlIncTrans         % P&L including transaction costs
    end % properties

    methods
        %%-------------------------------------------------------------------
        % Constructor--------------------------------------------------------
        function obj = BacktestData(sigma,mu,time,residuals,strat)
            obj.time = time(2:end);
            % Get position and P&L
            obj.position = getPosition(sigma,mu,residuals,strat);
            obj.pnl = cumsum(diff(residuals).*obj.position);

            % Get underwater and max drawdown
            [obj.underwater,obj.maxDrawdown] = getUnderwater(obj.pnl);

            % Get backtestScore
            obj.backtestScore = -obj.pnl(end)/obj.maxDrawdown;

            % Get cumTransactions
            obj.cumTransactions = zeros(size(obj.position));
            tempArray = obj.position(2:end) ~= obj.position(1:end-1);
            obj.cumTransactions = [0;cumsum(tempArray)];
```

```
34
35              disp(['Backtesting score: ' num2str(obj.backtestScore)]);
36
37              % Plot
38              obj.plot(sigma,mu,residuals);
39
40          end % Constructor
41
42          %%-------------------------------------------------------------------------
43          % Plotter-----------------------------------------------------------------
44          function plot(obj,sigma,mu,residuals)
45
46              figHandle = figure;
47              % Title
48              annotation('textbox',[0 0.9 1 0.1],'String',...
49                  ['Backtest Score = ' num2str(obj.backtestScore,'%.2f')...
50                  ', \sigma = ' num2str(sigma,'%.4f') ', End P&L = '...
51                  num2str(obj.pnl(end),'%.2f') ', Max Drawdown = ' ...
52                  num2str(obj.maxDrawdown,'%.2f')],...
53                  'EdgeColor','none','HorizontalAlignment','center');
54              % Plot the residuals and the position
55              subplot(2,1,1);
56              hold on;
57              plot(obj.time,residuals(2:end),'b');
58              plot(obj.time,mu+sigma*obj.position,'r');
59              hold off;
60              legend('Residuals','Position','location','northwest');
61              xlabel('Time');
62              yLimits = ylim;
63              changeFig(figHandle);
64              axis([obj.time(1),obj.time(end),yLimits(1),yLimits(2)]);
65              datetick('x','yyyy','keeplimits','keepticks')
66
67              % Plot the P&L and the underwater chart
68              subplot(2,1,2);
69              hold on;
70              plot(obj.time,obj.pnl,'b');
71              fillHandle = fill([obj.time;flipud(obj.time)],...
72                  [obj.underwater;zeros(size(obj.time))],'r');
73              set(fillHandle,'EdgeColor','None');
74              hold off;
75              legend('P&L','Underwater','location','northwest');
76              xlabel('Time');
77              yLimits = ylim;
78              changeFig(figHandle);
79              axis([obj.time(1),obj.time(end),yLimits(1),yLimits(2)]);
80              datetick('x','yyyy','keeplimits','keepticks')
81
82          end % plot
83
84          %%-------------------------------------------------------------------------
85          % Transaction Costs Analysis----------------------------------------------
86          obj = adjustForTrans(obj,flatFee)
87
88
89      end % methods
```

```
90
91 end % classdef
```

2.1.6.1 adjustForTrans

```
1  function obj = adjustForTrans(obj,flatFee)
2  % Function to analyse transaction costs
3
4      % Orientate the flatFee array
5      if size(flatFee,1) ~= 1
6          flatFee = flatFee';
7      end
8
9      % Calculate transaction costs
10     adjFactor = -kron(abs(diff(obj.position)),flatFee);
11     % Calculate the P&L including transaction costs
12     obj.pnlIncTrans = [zeros(1,size(flatFee,2));cumsum(bsxfun(...
13         @plus,adjFactor,diff(obj.pnl)))];
14
15     % Display no fee backtest score
16     disp(['No fee - Backtest score = ' num2str(obj.backtestScore)]);
17
18     % For loop to display backtest scores
19     for i = 1:size(flatFee,2)
20         % Get max drawdown
21         [~,tempMaxDrawdown] = getUnderwater(obj.pnlIncTrans(:,i));
22         % Display backtest score for each fee
23         disp(['Fee = ' num2str(flatFee(1,i)) ' - Backtest score = '...
24             num2str(-obj.pnlIncTrans(end,i)/tempMaxDrawdown)]);
25     end
26
27     % Plotting
28     figHandle = figure;
29     hold on;
30     plot(obj.time,obj.pnl);
31     legendStr = cell(1,size(obj.pnlIncTrans+1,2));
32     legendStr{1} = 'No fee';
33     for i = 1:size(obj.pnlIncTrans,2)
34         plot(obj.time,obj.pnlIncTrans(:,i));
35         legendStr{i+1} = ['Fee = ' num2str(flatFee(1,i))];
36     end
37     hold off;
38     legend(legendStr,'location','northwest');
39     xlabel('Time');
40     ylabel('P&L');
41     yLimits = ylim;
42     changeFig(figHandle);
43     axis([obj.time(1),obj.time(end),yLimits(1),yLimits(2)]);
44     datetick('x','yyyy','keeplimits','keepticks')
45
46 end % adjustForTrans
```

2.1.6.2 getPosition

```
function positionArray = getPosition(sigma,mu,residuals,varargin)
% Function to get the positions for different stategies

    % If no input strategy then default
    if nargin == 3
        strategy = 1;
    elseif nargin == 4
        strategy = varargin{1};
    elseif nargin > 4
        error('Too many input arguments.');
    end

    % Calculate position array depending on strategy
    switch strategy
        % Strategy one
        case 1
            % long short position array
            positionArray = zeros(size(residuals,1)-1,1);
            for i = 2:size(positionArray,1)
                if positionArray(i-1) == 0
                    if residuals(i) < mu - sigma
                        positionArray(i) = 1;
                    elseif residuals(i) > mu + sigma
                        positionArray(i) = -1;
                    else
                        positionArray(i) = positionArray(i-1);
                    end
                elseif positionArray(i-1) == 1
                    if residuals(i) > mu
                        positionArray(i) = 0;
                    else
                        positionArray(i) = positionArray(i-1);
                    end
                elseif positionArray(i-1) == -1
                    if residuals(i) < mu
                        positionArray(i) = 0;
                    else
                        positionArray(i) = positionArray(i-1);
                    end
                end
            end

        % Strategy 2
        case 2
            % long short position array hold from +sigma to -sigma and vice
            % versa
            positionArray = zeros(size(residuals,1)-1,1);
            for i = 2:size(positionArray,1)
                if positionArray(i-1) == 0
                    if residuals(i) < mu - sigma
                        positionArray(i) = 1;
                    elseif residuals(i) > mu + sigma
```

```
53                          positionArray(i) = -1;
54                      else
55                          positionArray(i) = positionArray(i-1);
56                      end
57                  elseif positionArray(i-1) == 1
58                      if residuals(i) > mu + sigma
59                          positionArray(i) = 0;
60                      else
61                          positionArray(i) = positionArray(i-1);
62                      end
63                  elseif positionArray(i-1) == -1
64                      if residuals(i) < mu - sigma
65                          positionArray(i) = 0;
66                      else
67                          positionArray(i) = positionArray(i-1);
68                      end
69                  end
70              end
71
72          % Strategy 3
73          case 3
74              % long short position array hold from +sigma to -sigma and vice
75              % versa and aggresive
76              positionArray = zeros(size(residuals,1)-1,1);
77              for i = 2:size(positionArray,1)
78                  if positionArray(i-1) == 0
79                      if residuals(i) < mu - sigma
80                          positionArray(i) = 1;
81                      elseif residuals(i) > mu + sigma
82                          positionArray(i) = -1;
83                      else
84                          positionArray(i) = 0;
85                      end
86                  elseif positionArray(i-1) > 0
87                      if residuals(i) > mu + sigma
88                          positionArray(i) = 0;
89                      elseif residuals(i) < mu + -positionArray(i-1)*sigma
90                          positionArray(i) = positionArray(i-1) + 1;
91                      else
92                          positionArray(i) = positionArray(i-1);
93                      end
94                  elseif positionArray(i-1) < 0
95                      if residuals(i) < mu - sigma
96                          positionArray(i) = 0;
97                      elseif residuals(i) > mu + -positionArray(i-1)*sigma
98                          positionArray(i) = positionArray(i-1) - 1;
99                      else
100                         positionArray(i) = positionArray(i-1);
101                     end
102                 end
103             end
104
105     end % switch
106
107 end % getPosition
```

2.1.6.3 getUnderwater

```
function [ underwater, maxDrawdown ] = getUnderwater( pnlArray )
% Function to get underwater and maximum drawdown

    % Get the high watermark
    highWatermark = cummax(pnlArray);
    % Calculate the underwater array
    underwater = min(pnlArray-highWatermark,0);
    % Calculate the maximum drawdown
    maxDrawdown = min(underwater);

end % getUnderwater
```