CHAPTER 3

Implementing Mean Reversion Strategies

In the previous chapter, we described the statistical tests for determining whether a price series is stationary and therefore suitable for mean reversion trading. This price series may be the market value of a single asset, though it is rare that such stationary assets exist, or it may be the market value of a portfolio of cointegrating assets, such as the familiar long-short stock pair.

In practice, though, we should remember that we don't necessarily need true stationarity or cointegration in order to implement a successful mean reversion strategy: If we are clever, we can capture short-term or seasonal mean reversion, and liquidate our positions before the prices go to their next equilibrium level. (Seasonal mean reversion means that a price series will mean-revert only during specific periods of the day or under specific conditions.) Conversely, not all stationary series will lead to great profits—not if their half-life for mean reversion is 10 years long.

We also described a simple linear mean reversion strategy that simply "scales" into an asset in proportion to its price's deviation from the mean. It is not a very practical strategy due to the constant infinitesimal rebalancing and the demand of unlimited buying power. In this chapter, we discuss a more practical, but still simple, mean reversion strategy—

the Bollinger bands. We describe variations of this technique, including the pros and cons of using multiple entry and exit levels ("scaling-in"), and the use of the Kalman filter to estimate the hedge ratio and mean price. Finally, we highlight the danger data errors pose to mean-reverting strategies.

In presenting the backtests of any strategy in this book, we do not include transaction costs. We sometimes even commit a more egregious error of introducing look-ahead bias by using the same data for parameter optimization (such as finding the best hedge ratio) and for backtest. These are all pitfalls that we warned about in Chapter 1. The only excuse for doing this is that it makes the presentation and source codes simpler to understand. I urge readers to undertake the arduous task of cleaning up such pitfalls when implementing their own backtests of these prototype strategies.

Trading Pairs Using Price Spreads, Log Price Spreads, or Ratios

In constructing a portfolio for mean reversion trading in Chapter 2, we simply used the market value of the "unit" portfolio as the trading signal. This market value or price is just the weighted sums of the constituent price series, where the weights are the hedge ratios we found from linear regression or from the eigenvectors of the Johansen test:

$$y = h_1 y_1 + h_2 y_2 + \cdots + h_n y_n \tag{3.1}$$

y is, by construction, a stationary time series, and the h_i's tell us the number of shares of each constituent stock (assuming we are trading a stock portfolio). In the case of just two stocks, this reduces to a spread familiar to many pair traders:

$$y = y_1 - h y_2. \tag{3.2}$$

(We inserted a minus sign in Equation 3.2 to anticipate the fact that we will usually be long one stock and short another, so that h as defined this way will be positive.) Suppose instead of price series, we find that the log of prices are cointegrating, such that

$$\log(q) = h_1 \log(y_1) + h_2 \log(y_2) + \cdots + h_n \log(y_n) \tag{3.3}$$

is stationary for some set of h's derived from either a regression fit or Johansen's eigenvectors. How do we interpret this equation, since q (for "query") is just a name given to a stationary time series that may or may not be the market value of a portfolio? To find out its properties, let's take its first difference in time:

$$\Delta\log(q) = h_1\Delta\log(y_1) + h_2\Delta\log(y_2) + \cdots + h_n\Delta\log(y_n). \qquad (3.4)$$

Remembering that $\Delta log(x) \equiv log(x(t)) - log(x(t-1)) = log(x(t)/x(t-1)) \approx \Delta x/x$ for small changes in x, the right hand side of Equation 3.4 becomes $h_1\Delta y_1/y_1 + h_2\Delta y_2/y_2 + \cdots + h_n\Delta y_n/y_n$, which is none other than the returns of a portfolio consisting of the n assets with weights h's. But unlike the hedge ratio h's in Equation 3.1 where they referred to the number of shares of each asset, here we can set the market value of each asset to h. So we can interpret q as the market value of a portfolio of assets with prices $y_1, y_2, \ldots, y_n$ and with constant capital weights $h_1, h_2, \ldots, h_n$, together with a cash component implicitly included, and this market value will form a stationary time series. Note that a cash component must be implicitly included in the portfolio q because if the capital weights h's are kept constant, there is no other way that the market value of the portfolio can vary with time. This cash does not show up in Equation 3.4 because its market value, of course, doesn't change from $t - 1$ to t as a result of market movement, but its value will change at t when the trader rebalances the portfolio to maintain the constancy of the capital weights, realizing some of the gains or losses, and adding to or subtracting from the cash balance. So to keep the market value of this portfolio stationary (but not constant!) requires a lot of work for the traders, as they need to constantly rebalance the portfolio, which is necessitated by using the log of prices.

The upshot of all these is that mean reversion trading using price spreads is simpler than using log price spreads, but both can be theoretically justified if both price and log price series are cointegrating. But what about the ratio of prices y_1/y_2 that many traders favor as the signal for a pair? If we look at Equation 3.1 in the case of just two price series, we notice that if $h_1 = -h_2$, then indeed $\log(y_1/y_2)$ or y_1/y_2 is stationary. But this is a special case: We normally don't expect the hedge ratios to be equal in magnitude, or equal to -1 if we normalize them. So the ratio y_1/y_2 does not necessarily form a stationary series. But as one reader mentioned, using ratios may have an advantage when the

underlying pair is not truly cointegrating (http://epchan.blogspot.com/2012/02/ideas-from-psychologist.html?showComment=1329801874131#c3278677864367113894). Suppose price A = $10 and price B = $5 initially, so the ratio is 2. After some time, price A increases to $100 and price B to $50. The spread has gone from $5 to $50, and we will probably find that it is not stationary. But the ratio remains 2, and a mean-reverting strategy that trades based on ratio can be equally effective whether their prices are $10 versus $5 or $100 versus $50. In other words, if your two assets are not really cointegrating but you believe their spread is still mean reverting on a short time frame, then using ratio as an indicator may work better than either price spreads or log price spreads. (This is the same idea as using moving average and standard deviation in our linear mean-reverting strategy.)

There is another good reason to use ratio when a pair is not truly cointegrating. For such pairs, we often need to use a dynamically changing hedge ratio to construct the spread. But we can dispense with this trouble if we use the ratio as a signal in this situation. But does a ratio work better than an adaptive hedge ratio with price (or log price) spreads? I don't know a general answer to this, but we can look at Example 3.1, where we compare the use of price spreads, log price spreads, and ratios in the linear mean reversion strategy involving GLD and USO, the gold and the crude oil exchange-traded funds (ETFs). You will find, in that example at least, price spreads with an adaptive hedge ratio work much better than ratio.

An interesting special case is currency trading. If we trade the currency pair EUR.GBP, we are using ratio because this is exactly equal to trading EUR.USD/GBP.USD. We already demonstrated a simple mean-reverting strategy on trading such currency pairs in Example 2.5 for USD.CAD using ratio as the signal. But about those pairs that have no ready-made cross rates on many brokerages or exchanges, such as MXN.NOK? Should we use the ratio USD.NOK/USD.MXN as the signal, or the spread USD.NOK–USD.MXN instead? Again, because MXN.NOK is not truly stationary, using the ratio MXN.NOK may be more effective. This is true even though we can't directly trade MXN.NOK, and have to trade USD.NOK and USD.MXN instead. (Trading USD.NOK and USD.MXN will generate profit and loss [P&L] denominated in both NOK and MXN. Trading MXN.NOK would have generated P&L denominated only in NOK. So the two methods are not identical.)

Example 3.1: Trading Price Spread, Log Price Spread, and Ratio

We apply the linear mean-reverting strategy from Examples 2.5 and 2.8 to the ETFs GLD and USO. But we try this strategy on the price spread, log price spread, and ratio for comparison.

Some traders believe that when oil prices go up, so do gold prices. The logic is that high oil price drives up inflation, and gold prices are positively correlated with inflation. But you can verify using one of the cointegration tests we studied in Chapter 2 that gold (represented by the ETF GLD) and oil prices (represented by USO) are not, in fact, cointegrated. (We will gloss over the difference between spot oil prices versus oil futures, which actually constitute USO. We will come back to this difference in Chapter 5). Nevertheless, we will see if there is enough short-term mean reversion to make a mean-reverting strategy profitable.

We will first try the price spread as the signal. But we need to dynamically recalculate the hedge ratio every day using a short look-back period (set to near-optimal 20 trading days with the benefit of hindsight) in order to adapt to the changing levels of the ETFs over time. The method we used to calculate the hedge ratio is linear regression, using the *ols* function from the jplv7 package as before. You can, of course, use the first eigenvector from the Johansen test instead.

The MATLAB source code can be downloaded from my website as *PriceSpread.m*. We assume the price series of GLD is contained in the Tx1 array *x*, and that of USO is contained in the Tx1 array *y*. Note that what is usually referred to as the "spread" USO-hedgeRatio*GLD is equal to the price of the unit portfolio, which we denote as *yport* in the program.

```
% lookback period for calculating the dynamically changing
  % hedge ratio
lookback=20;
hedgeRatio=NaN(size(x, 1), 1);
for t=lookback:size(hedgeRatio, 1)
     regression_result=ols(y(t-lookback+1:t), ...
       [x(t-lookback+1:t) ones(lookback, 1)]);
     hedgeRatio(t)=regression_result.beta(1);
```

(*Continued*)

```
end
y2=[x y];
yport=sum([-hedgeRatio ones(size(hedgeRatio))].*y2, 2);
```

Plotting this spread in Figure 3.1 shows that it looks very stationary. We will now see if we can create a profitable linear mean reversion strategy. Once again, the number of units (shares) of the unit portfolio we should own is set to be the negative Z-Score, and the Tx2 *positions* array represents the market value (in dollars) of each of the constituent ETFs we should be invested in.

```
numUnits=-(spread-movingAvg(spread, lookback)) ...
  ./movingStd(spread, lookback);
positions=repmat(numUnits, [1 size(y2, 2)]).*[hedgeRatio ...
  -ones(size(hedgeRatio))].*y2; pnl=sum(lag(positions, ...
  1).*(y2-lag(y2, 1))./lag(y2, 1), 2); % daily P&L of the
  % strategy
ret=pnl./sum(abs(lag(positions, 1)), 2); % return is P&L
  % divided by gross market value of portfolio
```


We obtain an annual percentage rate (APR) of about 10.9 percent and Sharpe ratio of about 0.59 using price spread with a dynamic hedge ratio, even though GLD and USO are by no means cointegrated.

Next, we will see if using log prices will make any difference. The source code for this is in *LogPriceSpread.m*, but we will display here the only two lines that are different from *PriceSpread.m*:

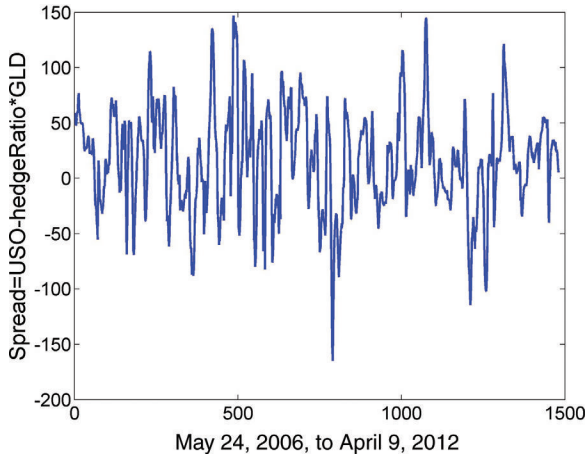

FIGURE 3.1 Spread between USO and GLD Using a Changing Hedge Ratio

```
regression_result=ols(log(y(t-lookback+1:t)), ...
  [log(x(t-lookback+1:t)) ones(lookback, 1)]);
```

and

```
yport=sum([-hedgeRatio ones(size(hedgeRatio))].*log(y2), ...
  2); % The net market value of the portfolio is same as
  % the "spread"
```

The APR of 9 percent and Sharpe ratio of 0.5 are actually lower than the ones using the price spread strategy, and this is before accounting for the extra transactions costs associated with rebalancing the portfolio every day to maintain the capital allocation to each ETF.

Next, we will try using ratio as the signal. In this case, we will also require the long and short side to have the same dollar capital. The source code is in *Ratio.m*. It is interesting to look at a plot of the ratio in Figure 3.2 first.

You can see that the ratio actually doesn't look very stationary at all, compared with either the price spread or adaptive hedge ratio. So it should not surprise us if we find the mean-reverting strategy to perform poorly, with a negative APR.

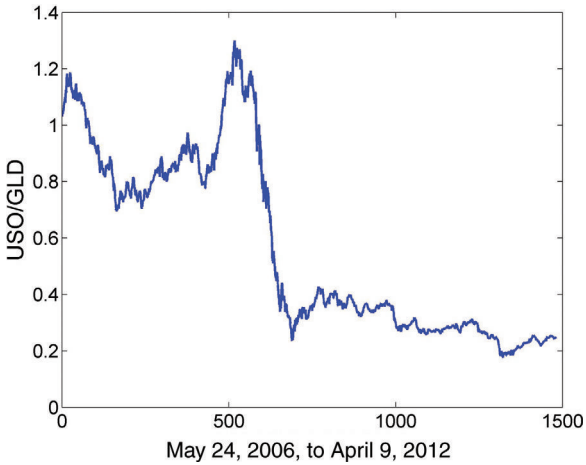

FIGURE 3.2 Ratio = USO/GLD

(*Continued*)

Example 3.1 (*Continued*)

```
lookback=20; % Lookback is set arbitrarily
ratio=y./x;
ratio(1:lookback)=[]; % Removed to have same test set as
  % price spread and log price spread strategies
x(1:lookback)=[];
y(1:lookback)=[];

% Apply a simple linear mean reversion strategy to GLD-USO
numUnits=-(ratio-movingAvg(ratio, lookback))...
  ./movingStd(ratio, lookback); positions=repmat(numUnits, ...
  [1 2]).*[-ones(size(x, 1), 1) ones(size(x, 1), 1)];
  pnl=sum(lag(positions, 1).*([x y]-lag([x y], 1)). ...
  /lag([x y], 1), 2); ret=pnl./sum(abs(lag(positions, 1)), 2);
```

Bollinger Bands

The only mean-reversal strategy I have described so far is the linear strategy: simply scale the number of units invested in a stationary unit portfolio to be proportional to the deviation of the market value (price) of the unit portfolio from a moving average. This simple strategy is chosen because it is virtually parameterless, and therefore least subject to data-snooping bias. While this linear strategy is useful for demonstrating whether mean reversion trading can be profitable for a given portfolio, it is not practical because we don't know beforehand what the maximum capital deployed will be, as there is no limit to the temporary deviation of the price from its average.

For practical trading, we can use the Bollinger band, where we enter into a position only when the price deviates by more than *entryZscore* standard deviations from the mean. *entryZscore* is a free parameter to be optimized in a training set, and both standard deviation and mean are computed within a look-back period, whose length again can be a free parameter to be optimized, or it can be set equal to the half-life of mean reversion. We can exit when the price mean-reverts to *exitZscore* standard deviations from the mean, where $exitZscore < entryZscore$. Note that if $exitZscore = 0$, this means we will exit when the price mean-reverts to the current mean. If $exitZscore = -entryZscore$, we will exit when the price moves beyond the opposite band so as to trigger a trading signal of the opposite sign. At any one time, we can have either zero or one unit (long or short) invested, so it is very easy to allocate capital to this strategy or to manage its risk. If we set the look-back

to a short period, and small *entryZscore* and *exitZscore* magnitude, we will get a shorter holding period and more round trip trades and generally higher profits. We illustrate the Bollinger band technique in Example 3.2 using the pair GLD-USO we discussed above.

Example 3.2: Bollinger Band Mean Reversion Strategy

We traded GLD-USO in Example 3.1 using price spread USO-hedgeRatio*GLD as the signal with a linear mean reversion strategy. Here, we simply switch to a Bollinger band strategy, using the *entryZscore* = 1 and *exitZscore* = 0, with the first part of the program identical to *PriceSpread.m*. The present source code is in *bollinger.m*. Notice that the entry signals *longsEntry* and *shortsEntry* are Tx1 logical arrays, as are the exit signals *longsExit* and *shortsExit*. We initialize the number of units of the unit portfolio on the long side, *numUnitsLong*, a Tx1 array, and then set one of its values to 1 if we have a long entry signal, and to 0 if we have a long exit signal; and vice versa for the number of units on the short side. For those days that do not have any entry or exit signals, we use the *fillMissingData* function to carry forward the previous day's units. (*fillMissingData* starts with the second row of an array, and overwrites any cell's NaN value with the corresponding cell's value in the previous row. It can be downloaded from my website.) Once *numUnitsLong* and *numUnitsShort* are computed, we can combine them to find the net number of units denoted by *numUnits*. The rest of the program is the same as in Example 3.1's *PriceSpread.m*.

```
% Bollinger band strategy
entryZscore=1;
exitZscore=0;

zScore=(yport-movingAvg(yport, lookback))./movingStd(yport, ...
  lookback);

longsEntry=zScore < -entryZscore; % a long position means we
  % should buy EWC
longsExit=zScore >= -exitZscore;

shortsEntry=zScore > entryZscore;
shortsExit=zScore <= exitZscore;

numUnitsLong=NaN(length(yport), 1);
```

(*Continued*)

```
numUnitsShort=NaN(length(yport), 1);

numUnitsLong(1)=0;
numUnitsLong(longsEntry)=1;
numUnitsLong(longsExit)=0;
numUnitsLong =fillMissingData(numUnitsLong);

numUnitsShort(1)=0;
numUnitsShort(shortsEntry)=-1;
numUnitsShort(shortsExit)=0;
numUnitsShort =fillMissingData(numUnitsShort);

numUnits= numUnitsLong + numUnitsShort;
```

The Bollinger band strategy has an APR = 17.8 percent, and Sharpe ratio of 0.96, quite an improvement from the linear mean reversal strategy! The cumulative returns curve is shown on Figure 3.3.

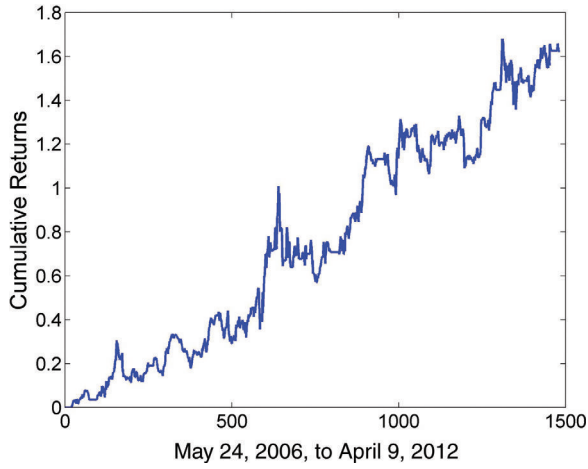

FIGURE 3.3 Cumulative Returns of Bollinger Band Strategy on GLD-USO

Does Scaling-in Work?

The notion of scaling into a position with a mean-reverting strategy is familiar to many traders. (Another name for it is *averaging-in*.) As the price (of an asset, a spread, or a portfolio) deviates further and further from its mean,

the potential profit to be reaped from an eventual reversal is also increasing; thus, it makes sense to increase the capital invested. This is exactly what our linear mean-reversal strategy does. Note also that this type of scaling-in strategies also scale out gradually: We do not have to wait until the price reverts to its mean before taking profits. The advantage of being able to exit whenever the price reverts by a small increment is that even if the price series is not really stationary and therefore never really reverts to its mean, we can still be profitable by constantly realizing small profits. An added benefit is that if you are trading large sizes, scaling-in and -out will reduce the market impact of the entry and exit trades. If we want to implement scaling-in using Bollinger bands, we can just have multiple entries and exits: for example, $entryZscore = 1, 2, 3, \ldots, N$ and $exitZscore = 0, 1, 2, \ldots, N - 1$. Of course, N is another parameter to be optimized using a training data set.

All of these seemed very commonsensical until the research by Schoenberg and Corwin proved that entering or exiting at two or more Bollinger bands is never optimal; that is, you can always find a single entry/exit level that will generate a higher average return in a backtest (Schoenberg and Corwin, 2010). They call this optimal single entry method "all-in."

To illustrate their point, let's say a future contract has recently dropped to a price L_1, and you expect it to revert to a higher final price $F > L_1$ (we have to assume mean reversion to compare averaging-in versus all-in), though there is a probability p that the price will go lower to $L_2 < L_1$ before rebounding to F. These possibilities are illustrated in Figure 3.4. We have just enough buying power to invest in a total of two contracts, whether at prices L_1, L_2, or F. Let's compare the three different methods of entry:

I. All-in at L_1: We invest all our capital when the price reaches L_1, not caring whether it will go lower to L_2.

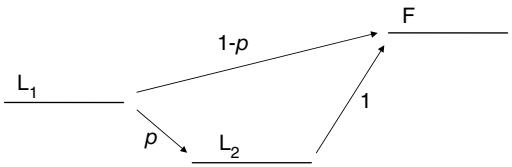

FIGURE 3.4 Two Possible Paths of Mean Reversion. Path 1 (with probability p) has price drops further from L_1 to L_2 before reverting to F. Path 2 (with probability $1 - p$) has price immediately reverts to F. (Note that mean reversion is guaranteed one way or the other in this example.)

II. All-in at L_2: We wait until the price reaches L_2 before investing all our capital. (Therefore, we invest nothing and earn zero returns if the price never reaches L_2.)
III. Average-in: We invest in one contract when the price reaches L_1, and in another contract if the price reaches L_2.

In all cases, we exit all contracts only when the price reaches F (so no average-out, even if there is average-in). What are the expected profits of each alternative? The expected profits in points are:

I. $2(F - L_1)$
II. $2p(F - L_2)$
III. $p[(F - L_1) + (F - L_2)] + (1 - p)(F - L_1) = (F - L_1) + p(F - L_2)$

Obviously, if $p = 0$, method I is the most profitable. If $p = 1$, method II is the most profitable. In fact, there is a transition probability $\hat{p} = (F - L_1) / (F - L_2)$ such that if $p < \hat{p}$, method I is more profitable than II, and vice versa if $p > \hat{p}$. It is also easy to show that if $p < \hat{p}$, method I is also more profitable than III, and if $p > \hat{p}$, method II is more profitable than III. So there is no situation where the average-in strategy is the most profitable one!

So does that mean the whole idea of scaling-in/averaging-in has been debunked? Not necessarily. Notice the implicit assumption made in my illustration: the probability of deviating to L_2 before reverting to F is constant throughout time. In real life, we may or may not find this probability to be constant. In fact, volatility is usually not constant, which means that p will not be constant either. In this circumstance, scaling-in is likely to result in a better realized Sharpe ratio if not profits. Another way to put it is that even though you will find that scaling-in is never optimal in-sample, you may well find that it outperforms the all-in method out-of-sample.

Kalman Filter as Dynamic Linear Regression

For a pair of truly cointegrating price series, determination of the hedge ratio is quite easy: just take as much historical data as you can find, and use ordinary least square (OLS) for a regression fit or use the Johansen test to find the eigenvectors. But as we have emphasized before, stationarity and cointegration are ideals that few real price series can achieve. So how best to

estimate the current hedge ratio for a pair of real price series when it can vary with time? In all the mean-reverting strategies we have discussed so far, we just took a moving look-back period and computed the regression coefficient or Johansen eigenvector over data in that period only. This has the disadvantage that if the look-back period is short, the deletion of the earliest bar and the inclusion of the latest bar as time moves forward can have an abrupt and artificial impact on the hedge ratio. We face the same problem if we use moving averages or moving standard deviations to calculate the current mean and standard deviation of a price series. In all cases, we may be able to improve the estimate by using a weighting scheme that gives more weight to the latest data, and less weight to the earlier data, without an arbitrary cutoff point. The familiar exponential moving average (EMA) is one such weighting scheme, but it is not clear why an exponential decrease in weights is optimal either. Here, we will describe a scheme of updating the hedge ratio using the Kalman filter that avoids the problem of picking a weighting scheme arbitrarily (Montana, Triantafyllopoulos, and Tsagaris, 2009).

Kalman filter is an optimal linear algorithm that updates the expected value of a hidden variable based on the latest value of an observable variable. (For a good exposition of this topic, see Kleeman, 2007.) It is linear because it assumes that the observable variable is a linear function of the hidden variable with noise. It also assumes the hidden variable at time t is a linear function of itself at time $t-1$ with noise, and that the noises present in these functions have Gaussian distributions (and hence can be specified with an evolving covariance matrix, assuming their means to be zero.) Because of all these linear relations, the expected value of the hidden variable at time t is also a linear function of its expected value prior to the observation at t, as well as a linear function of the value of the observed variable at t. The Kalman filter is optimal in the sense that it is the best estimator available if we assume that the noises are Gaussian, and it minimizes the mean square error of the estimated variables.

For every application of Kalman filtering, we need to first figure out what these variables and matrices are:

- Observable variable (vector)
- Hidden variable (vector)
- State transition model (matrix)
- Observation model (matrix)

This is actually the only creative part of the application because once these quantities are specified, the rest is just a robotic application of an existing algorithm. As traders, we don't need to know how to derive the relationships between these quantities—we only need to know where to find a good software package that gives us the right answer.

In our application where the focus is to find the hedge ratio and the average mean and volatility of the spread, the *observable variable* is one of the price series y, and the *hidden variable* is the hedge ratio β. The linear function that relates y and β is, of course,

$$y(t) = x(t)\,\beta(t) + \epsilon(t), \qquad \text{("Measurement equation")} \tag{3.5}$$

where x is the price series of the other asset, and ϵ is a Gaussian noise with variance V_ϵ. As we typically allow the spread between x and y to have a nonzero mean, we will use a 2×1 vector β to denote both the intercept μ and the slope of the linear relation between x and y, and we will augment $x(t)$ with a column vector of ones to create an $N \times 2$ array to allow for the constant offset between x and y. x actually serves as the *observation model* in the Kalman filter lingo.

It may seem strange that we regard only $y(t)$ as an observable but not $x(t)$, but this is just a mathematical trick, as every variable in the Kalman filter equations is observable except for the hidden variable and the noises, and so we have the freedom to designate which variable is *the* "observable" (y) and which one is the "observation model" (x). Next, we make a crucial assumption that the regression coefficient (our hidden variable) at time t is the same as that at time $t-1$ plus noise

$$\beta(t) = \beta(t-1) + \omega(t-1), \qquad \text{("State transition")} \tag{3.6}$$

where ω is also a Gaussian noise but with covariance V_ω. In other words, the *state transition model* here is just the identity matrix.

Given the specification of the four important quantities in italics, Kalman filtering can now iteratively generate the expected value of the hidden variable β given an observation at t. One noteworthy benefit of using the Kalman filter to find β is that not only do we obtain a dynamic hedge ratio between the two assets, we also simultaneously obtain what we used to call "the moving average" of the spread. This is because, as we mentioned, β includes both the slope and the intercept between y and x. The best current estimate of the intercept is used in place of the moving average of the spread. But, as your telemarketer often reminds you, that's not all! As a by-product, it also generates an estimate of the

standard deviation of the forecast error of the observable variable, which we can use in place of the moving standard deviation of a Bollinger band.

Despite the linearity of Kalman filtering, the matrix relations relating various quantities may seem quite complex, so I relegate them to Box 3.1 here for the patient reader to peruse.

Actually, besides the iterative equations, we also need to specify the (co) variances V_ϵ and V_ω of the measurement and state transition equations. These specifications will be included in Box 3.1 as well.

BOX 3.1

The Iterative Equations of the Kalman Filter

We denote the expected value of β at t given observation at $t-1$ by $\hat{\beta}(t|t-1)$, the expected value of β given observation at t by $\hat{\beta}(t|t)$, and the expected value of $y(t)$ given the observation at $t-1$ by $\hat{y}(t|t-1)$. Given the quantities $\hat{\beta}(t-1|t-1)$ and $R(t-1|t-1)$ at time $t-1$, we can make the one-step predictions

$$\hat{\beta}(t|t-1) = \hat{\beta}(t-1|t-1) \qquad \text{("State prediction")} \tag{3.7}$$

$$R(t|t-1) = R(t-1|t-1) + V_w \qquad \text{("State covariance prediction")} \tag{3.8}$$

$$\hat{y}(t) = x(t)\hat{\beta}(t|t-1) \qquad \text{("Measurement prediction")} \tag{3.9}$$

$$Q(t) = x(t)'R(t|t-1)x(t) + V_e \qquad \text{("Measurement variance prediction")} \tag{3.10}$$

where $R(t|t-1)$ is $\text{cov}(\beta(t) - \hat{\beta}(t|t-1))$, measuring the covariance of the error of the hidden variable estimates. (It is a covariance instead of a variance because β has two independent components.) Similarly, $R(t|t)$ is $\text{cov}(\beta(t) - \hat{\beta}(t|t))$. Remembering that the hidden variable consists of both the mean of the spread as well as the hedge ratio, R is a 2×2 matrix. $e(t) = y(t) - x(t)\hat{\beta}(t|t-1)$ is the forecast error for $y(t)$ given observation at $t-1$, and $Q(t)$ is $\text{var}(e(t))$, measuring the variance of the forecast error.

After observing the measurement at time t, the famous Kalman filter state estimate update and covariance update equations are

$$\hat{\beta}(t|t) = \hat{\beta}(t|t-1) + K(t) * e(t) \qquad \text{("State update")} \tag{3.11}$$

$$R(t|t) = R(t|t-1) - K(t) * x(t) * R(t|t-1) \qquad \text{("State covariance update")} \tag{3.12}$$

where $K(t)$ is called the Kalman gain and is given by

$$K(t) = R(t|t-1) * x(t)/Q(t) \tag{3.13}$$

To start off these recursions, we assume $\hat{\beta}(1|0) = 0$, $R(0|0) = 0$. But what about V_w and V_e? There is a method to estimate these variances from data called autocovariance least squares developed by Rajamani and Rawlings (2007, 2009). There is even a free Matlab/Octave package for implementing

(*Continued*)

BOX 3.1 (*Continued*)

this method at http://jbrwww.che.wisc.edu/software/als. But for simplicity, we will follow Montana and assume $v_\omega = \frac{\delta}{1-\delta} I$, where δ is a parameter between 0 and 1, and I is a 2 × 2 identity matrix. If $\delta = 0$, this means $\beta(t) = \beta(t-1)$, which reduces the Kalman filter to ordinary least square regression with a fixed offset and slope. If $\delta = 1$, this means the estimated β will fluctuate wildly based on the latest observation. The optimal δ, just like the optimal lookback in a moving linear regression, can be obtained using training data. With the benefit of hindsight, we pick $\delta = 0.0001$. With the same hindsight, we also pick $V_e = 0.001$.

In Example 3.3, we describe the actual implementation of using the Kalman filter to estimate a dynamic β for the EWA-EWC pair we discussed in Example 2.7.

Example 3.3: Kalman Filter Mean Reversion Strategy

We will now implement the Kalman filter equations 3.5 through 3.13 and apply them to the EWA-EWC pair. The code can be downloaded as *KF_beta_EWA_EWC.m*. We assume the price series of EWA is stored in a Tx1 array *x*, and that of EWC is stored in a Tx1 array *y*.

```
% Augment x with ones to accommodate possible offset in the
    % regression
% between y vs x.
x=[x ones(size(x))];

delta=0.0001; % delta=0 allows no change (like traditional
  % linear regression).

yhat=NaN(size(y)); % measurement prediction
e=NaN(size(y)); % measurement prediction error
Q=NaN(size(y)); % measurement prediction error variance

% For clarity, we denote R(t|t) by P(t).
% initialize P and beta.
P=zeros(2);
beta=NaN(2, size(x, 1));
Vw=delta/(1-delta)*diag(ones(2, 1));
Ve=0.001;

% Initialize beta(:, 1) to zero
beta(:, 1)=0;
```

```
for t=1:length(y)
     if (t > 1)
          beta(:, t)=beta(:, t-1); % state prediction.
            % Equation 3.7
          R=P+Vw; % state covariance prediction. Equation 3.8
     end

     yhat(t)=x(t, :)*beta(:, t); % measurement prediction.
       % Equation 3.9

     Q(t)=x(t, :)*R*x(t, :)'+Ve; % measurement variance
       % prediction. Equation 3.10


     % Observe y(t)
     e(t)=y(t)-yhat(t); % measurement prediction error

     K=R*x(t, :)'/Q(t); % Kalman gain

     beta(:, t)=beta(:, t)+K*e(t); % State update.
       % Equation 3.11
     P=R-K*x(t, :)*R; % State covariance update. Euqation 3.12

End
```


We can see from Figure 3.5 that with $\delta = 0.0001$, the Kalman-updated slope $\beta(1, t)$ of a linear fit between EWC (y) and EWA (x) oscillates around 1.

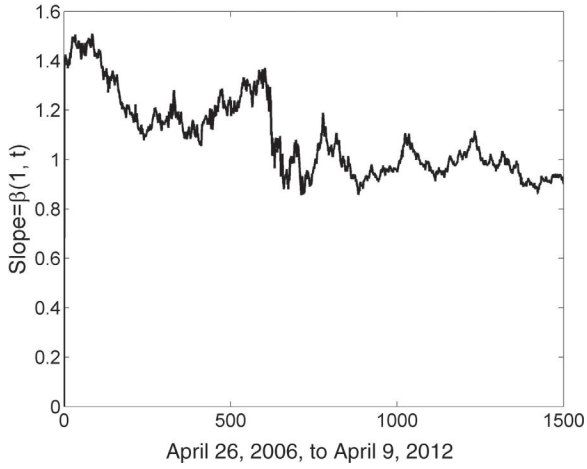

FIGURE 3.5 Kalman Filter Estimate of the Slope between EWC (y) and EWA (x)

(*Continued*)

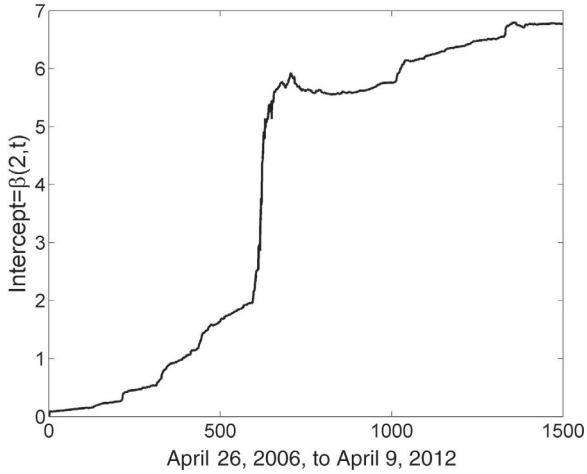

FIGURE 3.6 Kalman Filter Estimate of the Intercept between EWC (y) and EWA (x)

We can also see from Figure 3.6 that the Kalman-updated intercept $\beta(2, t)$ increases monotonically with time.

We can utilize these and other quantities computed from the Kalman filter to create a mean-reverting strategy. The measurement prediction error $e(t)$ (previously called the forecast error for $y(t)$ given observation at $t-1$) is none other than the deviation of the spread EWC-EWA from its predicted mean value, and we will buy this spread when the deviation is very negative, and vice versa if it is very positive. How negative or positive? That depends on the predicted standard deviation of $e(t)$, which is none other than $\sqrt{Q(t)}$. We can plot $e(t)$ and $\sqrt{Q(t)}$ on the same chart (Figure 3.7) to see that $\sqrt{Q(t)}$ changes quite slowly given our small δ.

The Matlab code for determining the entry and exit signals follows.

```
y2=[x(:, 1) y];

longsEntry=e < -sqrt(Q); % a long position means we should
  % buy EWC
longsExit=e > -sqrt(Q);

shortsEntry=e > sqrt(Q);
shortsExit=e < sqrt(Q);
```

Once the entry and exit signals are determined, the rest of the code is the same as *bollinger.m*—just substitute *beta(1, :)* in place of *hedgeRatio*. It has a reasonable APR of 26.2 percent and a Sharpe ratio of 2.4. Its cumulative returns are plotted on Figure 3.8.

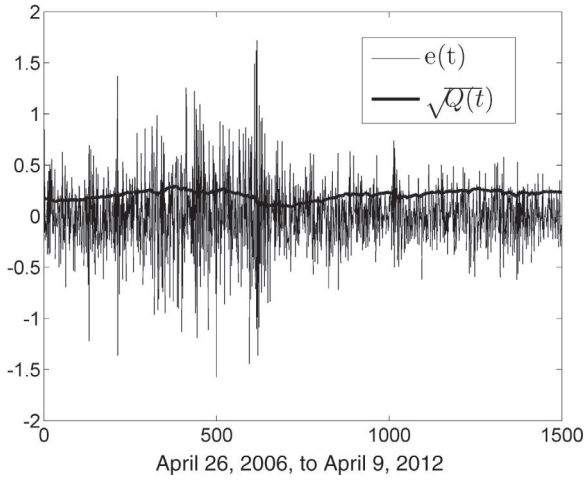

FIGURE 3.7 Measurement Prediction Error $e(t)$ and Standard Deviation of $e(t)$

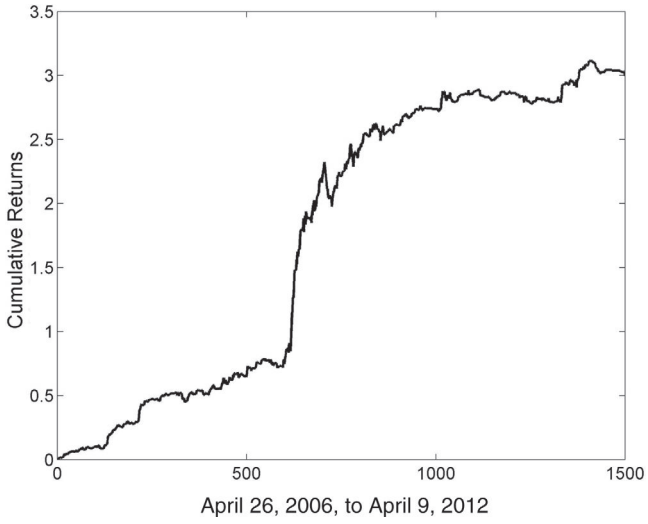

FIGURE 3.8 Cumulative Returns of Kalman Filter Strategy on EWA-EWC

(*Continued*)

Example 3.3 (*Continued*)

Instead of coding the Kalman filter yourself as we demonstrated, you can also use many free open-source MATLAB codes available. One such package can be found at www.cs.ubc.ca/~murphyk/Software/Kalman/kalman.html. Kalman filters are also available from MATLAB's Control System Toolbox.

Kalman Filter as Market-Making Model

There is another noteworthy application of Kalman filter to a mean-reverting strategy. In this application we are concerned with only one mean-reverting price series; we are not concerned with finding the hedge ratio between two cointegrating price series. However, as before, we still want to find the mean price and the standard deviation of the price series for our mean reversion trading. So the mean price $m(t)$ is the hidden variable here, and the price $y(t)$ is the observable variable. The measurement equation in this case is trivial:

$$y(t) = m(t) + \epsilon(t), \qquad \text{("Measurement equation")} \quad (3.14)$$

with the same state transition equation

$$m(t) = m(t-1) + \omega(t-1). \qquad \text{("State transition")} \quad (3.15)$$

So the state update equation 3.11 is just

$$m(t \mid t) = m(t \mid t-1) + K(t)(\, y(t) - m(t \mid t-1)). \quad \text{("State update")} \quad (3.16)$$

(This may be the time to review Box 3.1 if you skipped it on first reading.) The variance of the forecast error is

$$Q(t) = Var(m(t)) + V_e. \qquad (3.17)$$

The Kalman gain is

$$K(t) = R(t \mid t-1)/(R(t \mid t-1) + V_e), \qquad (3.18)$$

and the state variance update is

$$R(t \mid t) = (1 - K(t))R(t \mid t-1). \qquad (3.19)$$

Why are these equations worth highlighting? Because this is a favorite model for market makers to update their estimate of the mean price of an asset, as Euan Sinclair pointed out (Sinclair, 2010). To make these equations more practical, practitioners make further assumptions about the measurement error V_e, which, as you may recall, measures the uncertainty of the observed transaction price. But how can there be uncertainty in the observed transaction price? It turns out that we can interpret the uncertainty in such a way that if the trade size is large (compared to some benchmark), then the uncertainty is small, and vice versa. So V_e in this case becomes a function of t as well. If we denote the trade size as T and the benchmark trade size as T_{max}, then V_e can have the form

$$V_e = R(t \mid t-1)\left(\frac{T}{T_{max}} - 1\right) \tag{3.20}$$

So you can see that if $T = T_{max}$, there is no uncertainty in the observed price, and the Kalman gain is 1, and hence the new estimate of the mean price $m(t)$ is exactly equal to the observed price! But what should T_{max} be? It can be some fraction of the total trading volume of the previous day, for example, where the exact fraction is to be optimized with some training data.

Note the similarity of this approach to the so-called volume-weighted average price (VWAP) approach to determine the mean price, or fair value of an asset. In the Kalman filter approach, not only are we giving more weights to trades with larger trade sizes, we are also giving more weights to more recent trade prices. So one might compare this to volume *and* time-weighted average price.

The Danger of Data Errors

Data errors have a particularly insidious effect on both backtesting and executing mean-reverting strategies.

If there are errors, or "outliers," in the historical data used for backtesting, then these errors usually inflate the backtest performance of mean-reverting strategies. For example, if the actual trade prices of a stock at 11:00, 11:01, and 11:02 were \$100, \$100, and \$100, but the historical data erroneously recorded them as \$100, \$110, \$100, then your mean-reverting strategy's backtest is likely to have shorted the stock at 11:01 (\$110), and then covered the position at 11:02 (\$100) and made a tidy but fictitious profit of \$10. You can see that data quality is particularly important for intraday data, because they present much

more numerous opportunities for such errors. That's why reputable data vendors took great care in incorporating the exchange-provided cancel-and-correct codes to correct any trades that may have been canceled due to transaction prices that are too far from "normal." (What constitutes a "normal" price is solely determined, sometimes on a case-by-case basis, by the relevant exchange.) Thomas Falkenberry (2002) has written more on data cleansing issues.

However, this type of data error will suppress the backtest performance of momentum strategies, so it is not as dangerous. In the preceding example, a momentum strategy will likely buy the stock at 11:01 ($110) in backtest, and may be stopped out at a loss at 11:02 ($100).

The same kind of errors will, of course, trigger wrong trades in live trading as well, often resulting in real-life losses. In the preceding example, if the prices were bid prices, and we have the erroneous bid at $110 at 11:02, then our execution program may have sent a short market sell order at that time, which unfortunately will be filled at $100 instead since there was actually no bid at $110.

This problem with erroneous live bid/ask quotes is particularly dangerous when trading pairs or other arbitrage strategies, because in these strategies we often depend on the *differences* of the price quotes from various instruments to trigger trading signals. The difference of a pair of quotes is usually of much smaller magnitude than the quotes themselves, so any error in the quotes results in a much bigger percentage error in the spread. For example, if we are trading a pair of stocks X and Y, and X has a true bid price of $100 and Y has a true ask price of $105, so the spread Y−X is $5, which may be too small to trigger an market order to buy X and sell Y. But if data error causes Y to display an ask price of $106, then the erroneous spread becomes $6, an increase of 20 percent over the real spread of $5, and this may be enough to trigger an erroneous order to buy X and sell Y.

I have seen this problem in live trading when I used a broker's data feed to drive an equities pair-trading strategy. That data feed quite regularly triggered losing trades that I could not explain, until I switched the data feed to a third-party provider (nothing fancier than Yahoo! real-time quotes) and the bad trades stopped. Later on, I had access to Bloomberg's live data feed, and it didn't trigger any of these bad trades either.

Bad ticks in live data will also cause momentum strategies to send wrong orders. So they are equally loss-inducing to the execution of those strategies.

KEY POINTS

- Do you want to construct a mean-reverting portfolio with a fixed number of shares during the duration of a trade? Use price series to determine the hedge ratios.
- Do you want to construct a mean-reverting portfolio with fixed market values for each constituent during the duration of a trade? Use log price series to determine the hedge ratios.
- Ratio, instead of spreads, is often a good indicator for trading currency pairs.
- Afraid that the hedge ratio, mean, and standard deviation of a spread may vary in the future? Use a moving look-back period or the Kalman filter.
- A practical implementation of a linear trading strategy is the Bollinger bands with scaling-in.
- Scaling-in may not be optimal in backtests but is often useful for live trading where volatilities and probabilities do change.
- Do you want to dynamically update the expected price of an instrument based on its latest trade (price and size)? Use the Kalman filter.
- Data errors can inflate the backtest results of mean-reverting strategies but not momentum strategies.
- Strategies based on spreads are particularly sensitive to small data errors, whether in backtest or live trading.