A SPECIAL REPORT FROM

Adaptrade Software

Objective Methods for Identifying Chart Patterns

www.Adaptrade.com

Chart-based Support/Resistance Levels

A common distinction in trading is between systematic and discretionary trading methods. Systematic traders use hard and fast rules that are typically computerized in the form of one or more trading systems. Discretionary traders, on the other hand, don't use trading systems. While they have trading rules and methods, they haven't codified those rules into a computerized trading system. I've always found this distinction to be somewhat arbitrary. One of the tenets of good trading is to have well defined methods. Whether you computerize those methods into a trading system or choose to manually implement certain aspects of your approach, you still have something that could reasonably be called a "system." In fact, I'd go as far as to say that most discretionary methods are just trading systems yet to be programmed. A trader may be "discretionary" only because his approach seems too complicated to program.

One aspect of trading that can frustrate attempts at programming is chart patterns. Reading chart patterns, such as triangles and head and shoulder patterns, is said to be more art than science. If you have a trading approach based on chart patterns, you're probably a discretionary trader, if only because it's complicated to program a system to recognize a chart pattern.

Some time ago I was studying support and resistance levels for day trading the E-mini S&P. After scanning numerous intraday charts to locate these price levels visually, an algorithm occurred to me for identifying these support and resistance prices. In this section I'll show how this algorithm works and how to program it in TradeStation's EasyLanguage.

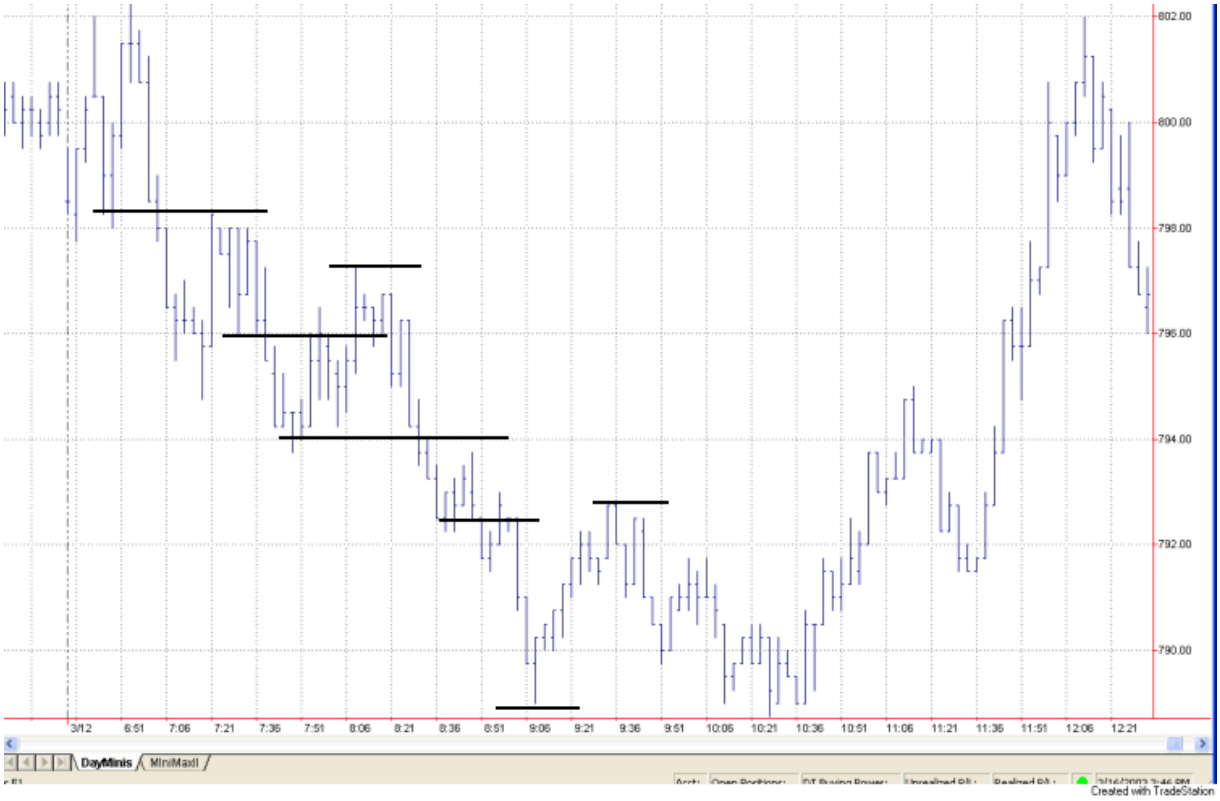

Fig. 1. A three minute chart of the E-mini S&P 500 showing several support and resistance levels.

Consider Fig. 1. I've drawn several horizontal lines by hand to suggest the kind of short-term support and resistance levels I have in mind. You might look for these price levels if you're trying to capture a few points of profit per trade. For example, you might try to sell short at the resistance level during a downtrend and cover at the support level. Similarly, in an up trend, you might try to buy at the support level and sell at the resistance level.

The logic I came up with to systematize the identification of these levels is pretty straight-forward. As each price bar is completed, I look for the most logical location for support and the most logical location for resistance relative to the close of the most recently completed bar. Since we are focusing on intraday data, which, in actual day trading, is evolving in real time, these support/resistance locations may change from bar to bar.

To identify the locations of support and resistance relative to the close, I scan all price levels within X points of the close over the last N bars. At each price level, imagine a horizontal line extending back from the current bar over the last N bars. Our goal is find the price level at which the horizontal line best fits the last N bars. I use a least-squares fitting method. At each price bar, determine whether the high or low of the bar is closest to the line and add the square of the difference between that price and the price level of the line to a running sum. This calculation is repeated for each of the N bars closest to the current bar to form the "sum of squares." Ignore price bars that are farther away from the line than Y points in order to avoid skewing the sum by these "outlier" prices. The whole process is repeated for each price level within X points of the current close.

The resistance level is then the price above the current bar's high for which the sum of squares is smallest. The support level is the price below the current bar's low for which the sum of squares is smallest.

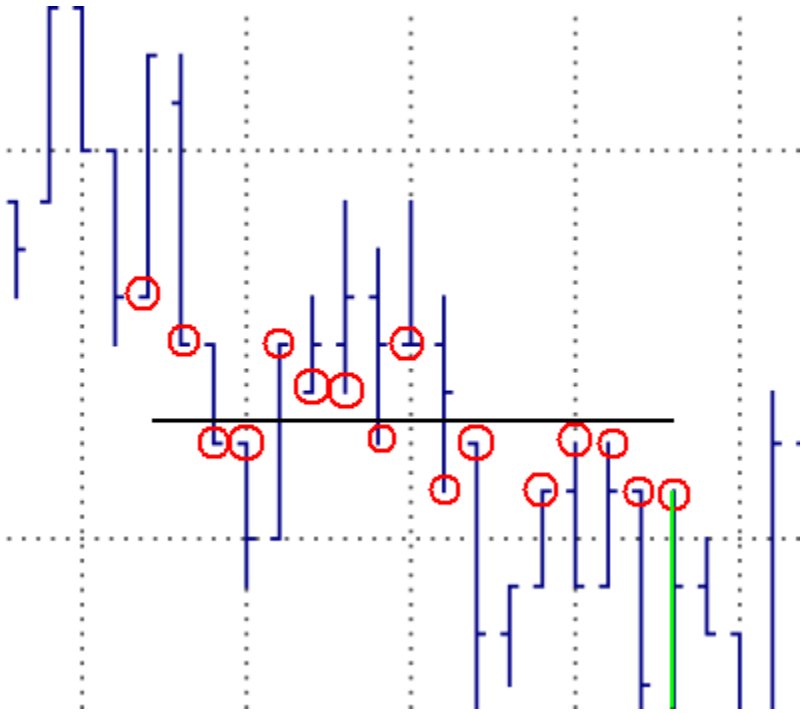

Fig. 2. Fitting a line to the price bars.

Fig. 2 illustrates the sum-of-squares process for a given price level. The horizontal line is a potential resistance level for the price bar colored green. The prices used in the sum of squares calculation are circled in red. Notice that one of the bars does not have a red circle on either its high or low. This is because the closest price on the bar is farther than Y points from the horizontal line. The sum of squares for this horizontal price level is the sum of the squares of the distances between the circled prices and the horizontal line.

I programmed the calculation of the support and resistance levels into an EasyLanguage function called SupRes2. The EasyLanguage code is shown below.

```
{
  Function: SupRes2 ("Support/Resistance")
  Locate the nearest price levels of support and resistance.

  Returns: Average of support and resistance levels; Also, the
  support and resistance levels are returned through the argument
  list.

  Michael R. Bryant
  Breakout Futures
  www.BreakoutFutures.com
  Copyright 2003-2011 Breakout Futures
}
  Input: NBars     (NumericSimple),  { # bars to lookback in search }
         PriceRnge (NumericSimple),  { # points to examine above/below close }
         PFilter   (NumericSimple),  { Include high/low within PFilter points }
         MinPoints (NumericSimple),  { min # points for a fit }
         SupPrice  (NumericRef),     { located support price }
         ResPrice  (NumericRef);     { located resistance price }

  Var:  iPrice     (0),      { price at current level }
        DtoLow     (0),      { distance from line to low }
        DtoHigh    (0),      { distance from line to high }
        Sum        (0),      { sum of weights }
        NormSum    (0),      { normalized sum }
        MinSum     (999999),{ max sum of weights }
        NPoints    (0),      { # points in sum }
        SupportP   (0),      { best support price }
        ResistP    (0),      { best resistance price }
        NPInc      (0),      { number of price increments above/below close }
        istart     (0),      { first increment to start at }
        ip         (0),      { loop counter of price increments }
        ib         (0);      { loop counter for bars }

  NPInc = PriceRnge/(MinMove/PriceScale);
  istart = IntPortion(NPInc/3);

  { Search for resistance; loop over prices above close }
  MinSum = 999999;
  ResistP = MaxList(High, ResPrice);

  For ip = istart to NPInc Begin
      iPrice = Close + ip * (MinMove/PriceScale);
      Sum = 0;
      NPoints = 0;

      { Loop over bars }
      For ib = 1 to NBars Begin

          { Add up sum of squares of distances to price line }
          DtoLow = AbsValue(L[ib] - iPrice);
```

```
        DtoHigh = AbsValue(H[ib] - iPrice);
        If DtoLow <= DtoHigh and DtoLow <= PFilter then Begin
           NPoints = NPoints + 1;
           Sum = Sum + Square(DtoLow);
        end
        else If DtoHigh < DtoLow and DtoHigh <= PFilter then Begin
           NPoints = NPoints + 1;
           Sum = Sum + Square(DtoHigh);
        end;

    end; { loop ib }

    { Record iPrice if sum is lowest so far }
    If NPoints >= MinPoints then Begin
       NormSum = SquareRoot(Sum/NPoints);
       If NormSum < MinSum then Begin
          MinSum = NormSum;
          ResistP = iPrice;
       end;
    end;
end; { loop ip }

ResistP = MaxList(High, ResistP);   { make sure resistance >= high }

{ Search for support; loop over prices below close }
MinSum = 999999;
SupportP = MinList(Low, SupPrice);
For ip = istart to NPInc Begin
    iPrice = Close - ip * (MinMove/PriceScale);
    Sum = 0;
    NPoints = 0;

    { Loop over bars }
    For ib = 1 to NBars Begin

        { Add up sum of squares of distances to price line }
        DtoLow = AbsValue(L[ib] - iPrice);
        DtoHigh = AbsValue(H[ib] - iPrice);
        If DtoLow <= DtoHigh and DtoLow <= PFilter then Begin
           NPoints = NPoints + 1;
           Sum = Sum + Square(DtoLow);
        end
        else If DtoHigh < DtoLow and DtoHigh <= PFilter then Begin
           NPoints = NPoints + 1;
           Sum = Sum + Square(DtoHigh);
        end;

    end; { loop ib }

    { Record iPrice if sum is lowest so far }
    If NPoints >= MinPoints then Begin
       NormSum = SquareRoot(Sum/NPoints);
       If NormSum < MinSum then Begin
          MinSum = NormSum;
          SupportP = iPrice;
       end;
    end;
end; { loop ip }

SupportP = MinList(Low, SupportP);   { make sure support <= Low }
SupPrice = SupportP;
ResPrice = ResistP;
```

```
  SupRes2 = (SupportP + ResistP)/2.;
```

I also created an indicator based on the SupRes2 function to plot the support and resistance levels. The code for the indicator is shown below.

```
{
  Indicator: SupRes Indicator-2
  Plot the support and resistance given by the SupRes2 function.

  Michael R. Bryant
  Breakout Futures
  www.BreakoutFutures.com
  Copyright 2003-2011 Breakout Futures
}
 Inputs: NBars     (30),      { # bars to lookback in search }
         PriceRnge (3),       { # points to examine above/below close }
         PFilter   (1.0),     { points this close to line }
         MinPoints (4);       { need at least this many points in fit }

 Var:    SupPrice  (C),       { located support price }
         ResPrice  (C);       { located resistance price }

 { Call SupRes function to find nearest support/resistance }
 Value1 = SupRes2(NBars, PriceRnge, PFilter, MinPoints, SupPrice,
                  ResPrice);

 Plot1(SupPrice, "Support");
 Plot2(ResPrice, "Resistance");
```

In Fig. 3, I show how the indicator works for the same price chart shown in Fig. 1. I used the following input parameter values: 30, 2.25, 0.75, 4. The support prices are in green and the resistance levels in red. During downtrending markets, the support levels tend to stay close to the price bar lows because none of the nearby price bars have prices lower than the most recent bar's low. Similarly, in uptrending markets, the resistance levels stay near the highs. The SupRes2 function could be employed in a trading system in a number of different ways, such as (1) to identify the trend based on whether support and resistance levels are rising or falling, and (2) to trigger entries and/or exits at the identified support and resistance levels.

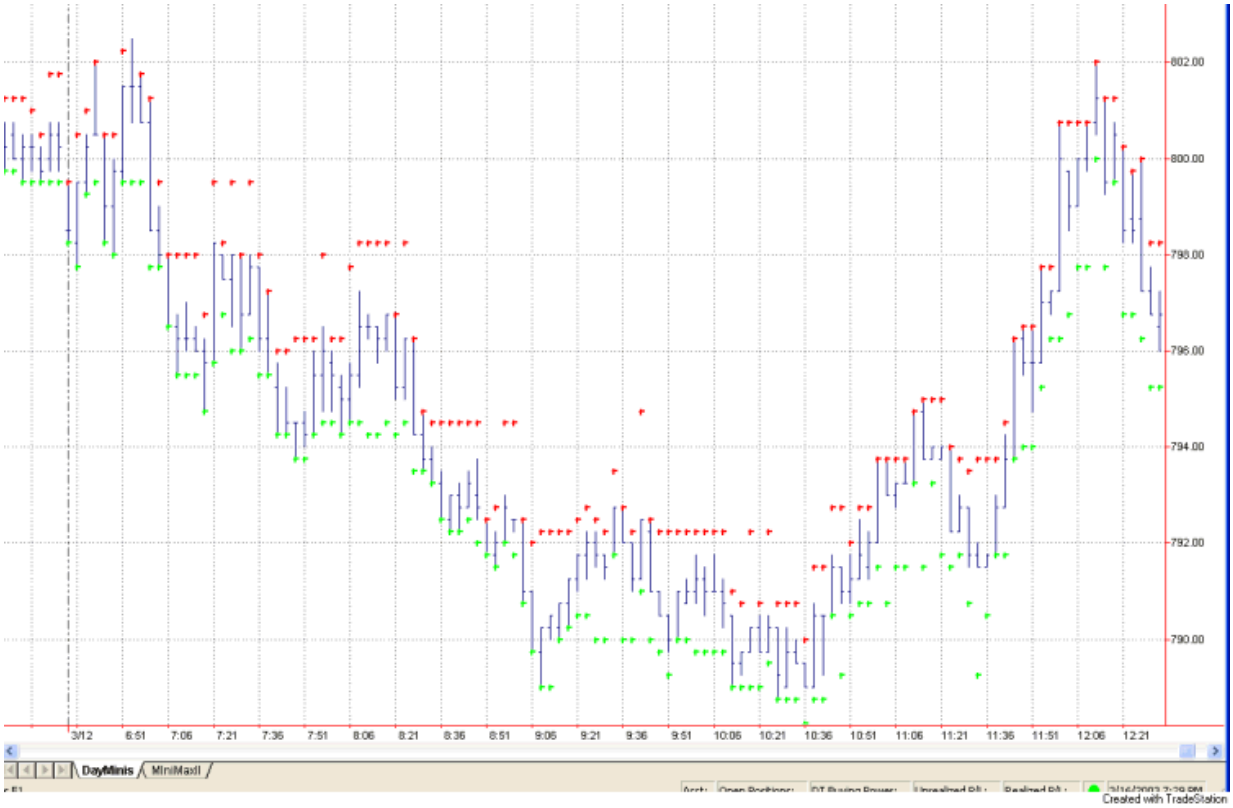

Fig. 3. SupRes Indicator plotted on a 3 minute chart of E-mini S&P data.

The SupRes2 function illustrates how a visual task like identifying a support or resistance level on a price chart can be made more objective by programming it. Incorporating this function into a trading system could turn a discretionary trading approach into a systematic one.

Consolidation Patterns

In the previous section I presented a method for identifying chart-based support and resistance levels. In this section I'll do the same thing for consolidation patterns. By consolidation pattern, I mean a trading range where the price bars tend to form a rectangular pattern on the chart. For example, in Figs. 4 and 5, I've identified several consolidation patterns by drawing a white rectangle around them.

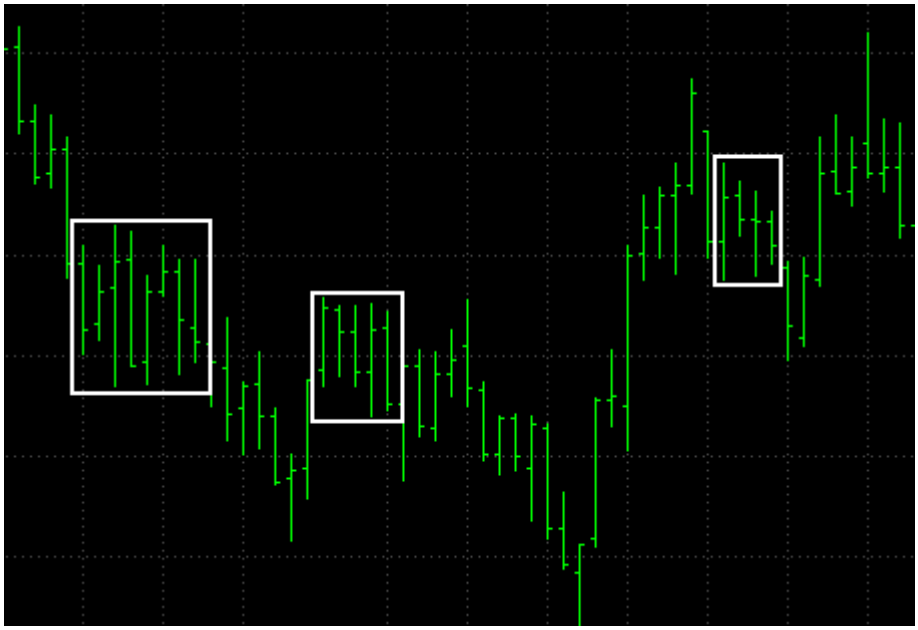

Fig. 4. Three consolidation patterns have been identified on this chart of the E-mini S&P.

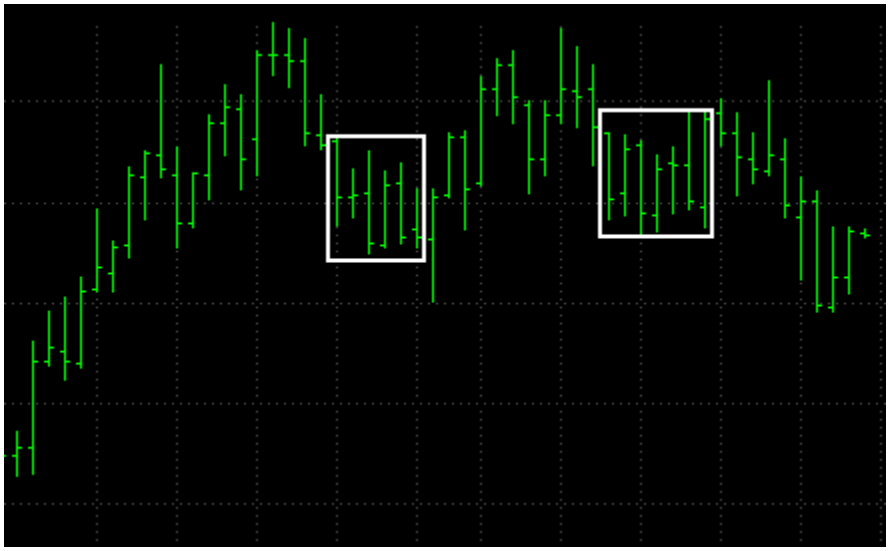

Fig. 5. Two additional consolidation patterns have been identified on this chart.

Consolidation patterns can also form other shapes, such as flags and triangles. Usually, a consolidation pattern occurs after a large move. In his book, "7 Chart Patterns That Consistently Make Money," (MarketPlace Books, 2000) Ed Downs cites consolidation patterns as the best tool to predict market direction. There are typically two methods for trading these patterns. You can enter on a breakout from the pattern by buying when the market moves above the upper boundary or selling on a downside breakout below the lower boundary. The second method is to trade within the pattern by selling at the upper boundary and buying at the lower boundary.

For the purposes of this article, I'm going to restrict the definition of consolidation patterns to horizontal trading range patterns, such as those shown in Figs. 4 and 5. To quantify the definition,

I'm going to say that a consolidation pattern is a sequence of bars on the chart where the "density" of the bars is greater than some threshold. To understand what I mean by "density," imagine drawing a rectangle around the price bars in the consolidation pattern, as I've done in Figs. 4 and 5. The rectangle surrounds the highest high and the lowest low of the bars in the pattern. The more completely the bars fill up the rectangle, the higher the density.

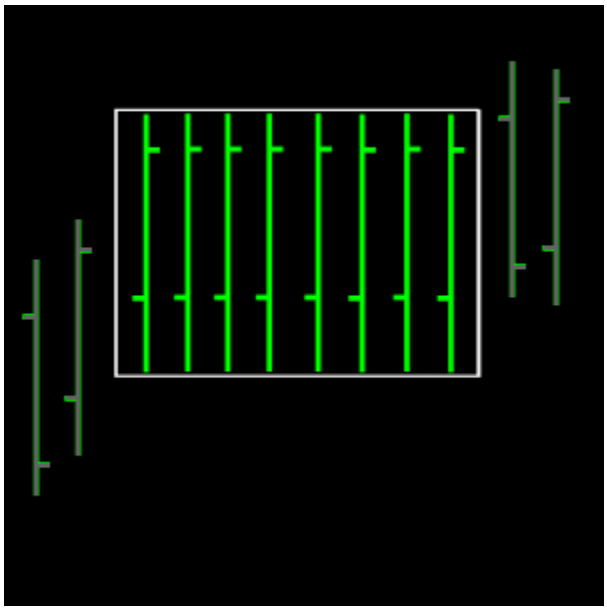

Fig. 6. A rectangle bounds a consolidation pattern with the highest possible density.

The highest possible density for a rectangular consolidation pattern is shown in Fig. 6. In this case, every bar spans the full height of the bounding rectangle. In physics, density is defined as mass per unit volume; the more mass contained within a given space, the higher the density. In our definition, the more bar area within the bounding rectangle, the higher the density. The "bar area" of the pattern can be defined as the sum of the heights of the bars within the bounding rectangle. By bar height, I mean the high minus the low or, to take gaps into account, the true range.

The area of the bounding rectangle is the number of bars in the pattern multiplied by the highest high minus the lowest low of the bars in the pattern; i.e.,

Area (bounding rectangle) = N * (Highest High - Lowest Low)

where N is the number of bars in the pattern. By this definition, the area of the bounding rectangle is the same as the bar area for the densest pattern, shown in Fig. 6.

The density of any pattern of price bars can now be defined as the bar area divided by the area of the bounding rectangle; i.e.,

Density = Sum of true ranges / [N * (Highest High - Lowest Low)].

By this definition, the pattern shown in Fig. 6 will have a density of 1. This is the highest possible density. Any other pattern will have a density value less than 1. The more completely the bars fill up the bounding rectangle, the closer the density will be to 1. Also note that since we've

normalized the bar area by the area of the bounding rectangle, the density value is independent of the number of bars, so we can compare consolidation patterns with different numbers of bars.

Now that we can calculate the density of a consolidation pattern, how do we identify those patterns? The key concept is that we're going to restrict our search for consolidation patterns to the current bar. We want to know if the current bar is part of a consolidation pattern. We could, in principle, extend the search to past price patterns, but since we can't trade in the past, the more important task is to determine the status of the current bar.

The approach is as follows: We start by looking at a minimum number of bars, NBMin. We calculate the density of the past NBMin bars. We then add one more bar to the search, and calculate the density of the past NBMin + 1 bars. We keep adding bars to the search, calculating the density of the pattern, until we reach some maximum number of bars, NBMax. For example, we might consider all patterns from four to 20 bars in length. Each pattern begins at the current bar and looks backwards. We want to find the pattern with the greatest density since this is most likely to be a consolidation pattern. Once we find this pattern, we record its density and the number of bars in the pattern. For example, we might find that the last six bars have the highest density.

The pattern with the highest density is our candidate consolidation pattern. To decide whether the candidate is an actual consolidation pattern, we have to establish a threshold value for the density. For example, we might decide that any pattern with a density greater than 0.6 is a consolidation pattern. I'll suggest how to come up with the threshold value shortly.

I programmed this method in EasyLanguage using two functions and one indicator. The function CPDensity calculates the density of the consolidation pattern. The function CPLocate calls CPDensity and determines whether the current bar is part of a consolidation pattern. CPLocate returns "true" if the current bar is part of a consolidation pattern and "false" otherwise. CPLocate also returns the density of the pattern, the number of bars in the pattern, and the upper and lower bounds of the pattern through its argument list. Finally, the indicator CPIndicate calls the CPLocate function and plots the upper and lower bounds of the consolidation pattern on the screen if the current bar is part of a consolidation pattern. CPIndicate also writes the results returned by CPLocate to the TradeStation MessageLog.

```
{
 Function: CPDensity ("Consolidation Pattern Density")
 Calculates the "density" of a consolidation/trading range pattern
 over the past NBars bars. The density is defined as the area of bars in
 the pattern divided by the maximum possible area. Area is defined as
 the sum of the height of the bars, where height is given by the
 true range. The max possible area is the area of the rectangle
 that bounds the bars; i.e., the highest high minus the lowest low
 times the number of bars.

 Michael R. Bryant
 Breakout Futures
 www.BreakoutFutures.com
 Copyright 2003-2011 Breakout Futures
}
 Input: NBars      (NumericSimple);   { number of bars in pattern }

 Var:   CPWidth    (0),  { highest high minus lowest low over last NBars bars }
        NB         (0),  { equal to NBars, provided NBars > 0 }
        SumTR      (0),  { sum of true ranges over NB bars }
        ii         (0);  { loop counter }
```

```
{ Make sure the input value of NBars is greater than zero }
NB = MaxList(NBars, 1);

{ Calculate width of rectangle bounding consolidatin pattern }
CPWidth = Highest(H, NB) - Lowest(L, NB);

{ Calculate sum of true ranges }
SumTR = 0;
For ii = 0 to NB - 1 Begin
    SumTR = SumTR + TrueRange[ii];
End;

{ Return sum of true ranges in pattern divided by max area of pattern }
CPDensity = SumTR/(CPWidth * NB);

{
Function: CPLocate
Determine if current bar is part of a consolidation/trading range
pattern. Return true if it is; false otherwise.
The pattern is identified as follows:
1. Patterns of length NBMin to NBMax are scanned using the
   function CPDensity, which calculates the "density" of
   the consolidation pattern. All patterns start at the
   current bar.
2. The length of the pattern with the highest density is
   recorded.
3. If the density of the highest density pattern matches or
   exceeds a threshold level, the pattern is considered to be
   a consolidation pattern, and TRUE is returned.
The function also returns the density, length (in bars),
and the upper and lower boundaries of the pattern with the highest
density.

Michael R. Bryant
Breakout Futures
www.BreakoutFutures.com
Copyright 2003-2011 Breakout Futures
}
Input: DensTH     (NumericSimple),   { Density threshold }
       NBMin      (NumericSimple),   { Min # of bars in pattern }
       NBMax      (NumericSimple),   { Max # of bars in pattern }
       PatDens    (NumericRef),      { Density of pattern }
       NBars      (NumericRef),      { Number of bars in pattern }
       HBound     (NumericRef),      { Upper/high boundary of pattern }
       LBound     (NumericRef);      { Lower boundary of pattern }

Var:   MaxDens    (0),               { max density found }
       NMaxDens   (0),               { length of pattern of max density }
       ibars      (0);               { loop counter }

{ Search for pattern with highest density }
MaxDens = 0.;
For ibars = NBMin to NBMax Begin
    PatDens = CPDensity(ibars);
    If PatDens > MaxDens then Begin
       MaxDens = PatDens;
       NMaxDens = ibars;
    End;
End;

{ Record results for densest pattern for return }
PatDens = MaxDens;
NBars = NMaxDens;
```

```
 HBound = Highest(H, NBars);
 LBound = Lowest(L, NBars);

 { Check if densest pattern exceeds threshold }
 If MaxDens >= DensTH then
    CPLocate = True
 Else
    CPLocate = False;

{
 Indicator: CPIndicate
 Plot the upper and lower bands of consolidation patterns identified
 by function CPLocate.

 Michael R. Bryant
 Breakout Futures
 www.BreakoutFutures.com
 Copyright 2003-2011 Breakout Futures
}
 Input: DensTH     (0.55);   { Density threshold }

 Var:   NBMin      (4),      { Min # of bars in pattern }
        NBMax      (30),     { Max # of bars in pattern }
        PatDens    (0),      { Density of pattern }
        NBars      (5),      { Number of bars in pattern }
        HBound     (0),      { Upper/high boundary of pattern }
        LBound     (0),      { Lower boundary of pattern }
        LocTF      (false);  { output of CPLocate function }

 LocTF = CPLocate(DensTH, NBMin, NBMax, PatDens, NBars, HBound, LBound);

 MessageLog(" Date: ", Date:6:0, " Time: ", time:6:0, " Density: ",
              PatDens:6:3, " # Bars: ", NBars:3:0);

 If LocTF then Begin
    Plot1(HBound, "Upper");
    Plot2(LBound, "Lower");
 End;
```

So how well does this method work? Take a second look at Figs. 4 and 5, above. Notice where I identified the consolidation patterns on the charts. Below, in Figs. 7 and 8, are the same charts, showing where the CPIndicate indicator has found consolidation patterns. CPIndicate places red and blue crosses at the upper and lower boundaries of the consolidation pattern it finds -- if any -- on the current bar. I used a density threshold value of 0.6 in this case, with NBMin = 4 and NBMax = 30. For the most part, the consolidation patterns identified by CPIndicate are the same as those I identified by eye.

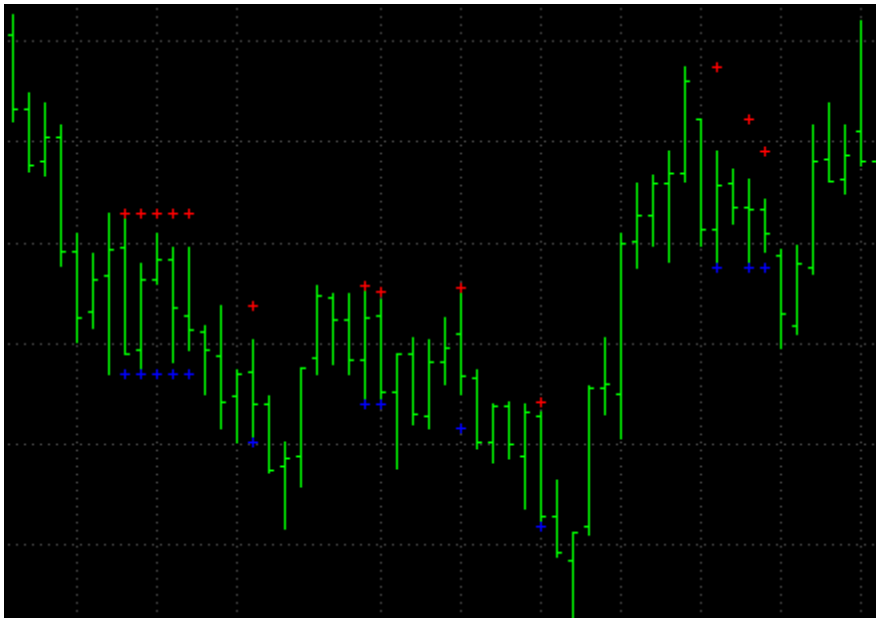

Fig. 7. Consolidation patterns identified by CPIndicate. Compare to Fig. 4.

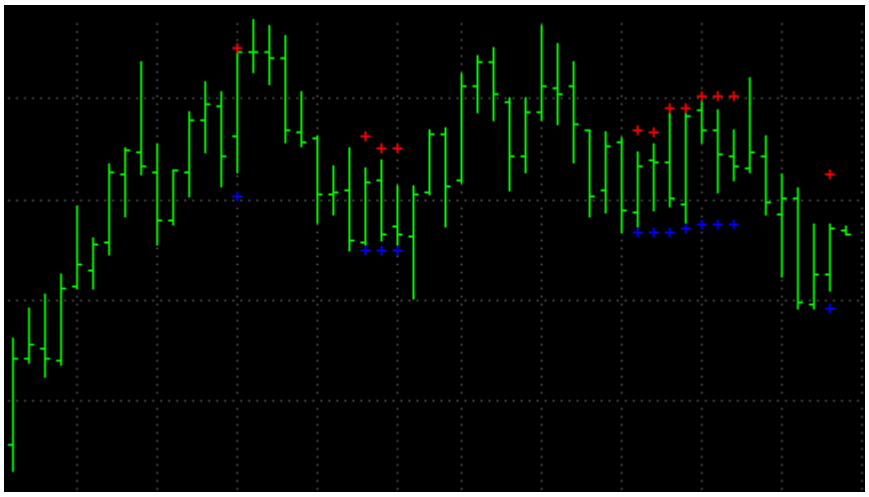

Fig. 8. Consolidation patterns identified by CPIndicate. Compare to Fig. 5.

Finding a good value for the density threshold is a kind of calibration process. You want to determine the threshold value that helps the CPLocate function find patterns that you consider to be valid consolidation patterns. One way to do this is to take several charts, mark all the consolidation patterns you can find by hand, then apply the CPIndicate indicator with a guess for the threshold value, which is the only input to the indicator. (By the way, the NBMin and NBMax variables in CPIndicate could just as easily be made inputs rather than variables.). Then look at the chart to see how closely the patterns identified by CPIndicate match the ones you came up with. Adjust the threshold value by trial and error until CPIndicate identifies the consolidation patterns you've chosen as closely as possible.

With a little work, the CPLocate function could be incorporated into a trading system. Ed Downs suggests that the market usually continues the trend established prior to the formation of

the consolidation pattern. If the market rises then consolidates, for example, it's likely that it will continue to go up after breaking out of the consolidation pattern. This logic could be combined with the CPLocate function to buy breakouts from a consolidation pattern following an up trending move and selling downside breakouts from a consolidation pattern following a down trending move. Alternatively, you could try selling at the top of the consolidation pattern and buying at the bottom of the pattern for a shorter-term trade. If nothing else, hopefully, the approach I've taken here illustrates that even seemingly vaguely-defined trading concepts, like chart patterns, can be made more objective, which is the first step in incorporating them into an objective trading plan.

About the Author:

Michael Bryant is the owner of Adaptrade Software and Breakout Futures. He holds degrees in engineering from the University of Connecticut (BS) and the University of Wisconsin-Madison (MS, PhD). He has been trading and studying the futures markets since 1994 and has developed Adaptrade Builder, an application for auto-generating trading strategies. Please visit **www.Adaptrade.com** to contact Dr. Bryant or to comment on this report.