TRUNCATED INDICATORS
By John F. Ehlers

The performance of some, but not all, indicators can be greatly enhanced by truncation. There are two fundamentally different types of digital filters, FIR and IIR. FIR stands for Finite Impulse Response, which means a fixed window of data is used to calculate a point of the filter output. That window slides across the data, and the output points are connected to provide the indicator. A Simple Moving Average (SMA) is an example of such a filter. RSI and Stochastic are examples of indicators that use this principle. FIR filters and indicators can only be degraded by truncation. IIR stands for Infinite Impulse Response, where the computation of the filter or indicator depends on a previous calculation of that filter. The Exponential Moving Average (EMA), and consequently the MACD and other indicators, are IIR types. Of course, the computation of an IIR filter does not extend to infinity. The filter computation can only start at the beginning of the data being used.

Therein lies the problem. The answer your get from an IIR filter will be different depending on your data length. If the data stream is sufficiently long the answer may be the same for all practical purposes, but it will be different nonetheless. So, initialization is one problem that is resolved by truncating the indicators. The memory of IIR indicators can also impact performance because a major data disturbance can cause the transient response of the indicator to ring like a bell, using that data disturbance in its output long after the event has occurred. This is particularly important because market data is nonstationary.

IIR filters are handy for a trader because one does not have to accept the group delay of a FIR filter, which is typically about half the length of the filter itself. I will first describe several ways to perform truncation and then I will discuss its performance impact.

An EMA depends on previous calculated values. In EasyLanguage notation, and EMA is:

Output = α*Input + (1-α)*Output[1];

Where Output[1] means the filter output one bar ago

And where α is the EMA constant (less than 1)

Of course, the output one bar ago requires its previous output two bars ago, and so on. Thus the required history goes back to infinity. The EMA equation can be rewritten as:

(Output – (1-α)*Output[1]) = α*Input

If I change notation at let Z^{-1} signify one bar of delay, the equation is rewritten as:

Output*(1 – (1-α)*Z^{-1}) = a*Input

The ratio of Output to Input is the transfer response of the filter, and so the transfer response of the filter, H, can be written as:

$H = \alpha / (1 – (1-\alpha)*Z^{-1})$

Just to simplify notation, let (1-α) = c. The equation then becomes

$H = \alpha / (1 – c*Z^{-1})$

The denominator of the transfer response carries the requirement for a calculation into the infinite past. However if we create an infinite series by dividing the dominator into the numerator by long division, the equation becomes:

$$H = \alpha*(1 + c*Z^{-1} + c^2*Z^{-2} + c^3*Z^{-3} + c^4*Z^{-4} + c^5*Z^{-5} + \ldots\ldots\ldots)$$

Truncation of this equation is easy, because we can just stop using the higher power terms at any power we choose. Since $(1-\alpha)$ is a number less that unity, we can estimate the desired length of the truncation by computing when the coefficients no longer have an impact on the transfer response. The EasyLanguage code fragment to compute a truncated EMA of a fixed length as a summation of terms is:

```
Output = 0;
For count = 0 to Length Begin
        Output = Output + Power(c, count)*Input[count];
End;
Output = a*Output;
```

One of the beauties of computers is that we can just brute force grind through calculations without resorting to mathematical sleight of hand tricks. This gives us more flexibility in easily truncating higher order filters. However, we need to have a little greater understanding of computer coding. In the example above, the terms Input and Output are variables. "Under the covers" of EasyLanguage, all variables are indexed to the current bar. Therefore counting backwards from the current bar is a snap. However, number crunching backwards requires multiple indexing, and therefore variables are not useful. Instead, we need to use arrays and do our own indexing. With Output as an array, the code fragment to brute force crunch a truncated EMA is :

```
Output[Length + 1] = Input[Length];  //initialization
For count = Length DownTo 1 Begin
        Output[count] = a*Input[count] + (1-a)*Output[count + 1];
End;
EMA = Output[1];  //converts array to a variable
```

A Bandpass filter is one I am fond of using because it enables me to winnow out the tradeable cycles in the market data. The Bandpass filter is an IIR type, and I use the brute force method of truncating this more complex filter. In Code Listing 1 I give the EasyLanguage code to compute a standard Bandpass filter in terms of its center period and percentage bandwidth. The standard computation as an IIR filter uses the computed values of the Bandpass filter both one bar ago and two bars ago. In the truncated version I use the array Trunc rather than an indexed variable. I crunch all the historical values as well as the current value for the array. I then convert the current value of the filter to a variable so I can plot it just like the standard computation.

In Figure 1 I show the Standard and Truncated Bandpass filters applied to daily bars of SPY for roughly the calendar year 2019. In both cases the center period of the filter is 20 bars and have a 10 percent of the center period bandwidth. The standard Bandpass filter is in red and the truncated Bandpass filter is in Blue. The truncated Bandpass filter only uses 10 bars of data in its computation at every bar across the chart.

The big price dip in December of 2018 certainly bangs the standard Bandpass filter and causes it to ring out like a bell for at least 5 months. On the other hand, the truncated Bandpass filter has a dampened response to that major event. It further accurately describes the price action by staying above zero during the uptrend into early May. During this time the cyclic price action is also accurately portrayed. For example, the price swing peak in the third week of March is accurately reflected in the peak of the truncated Bandpass filter, whereas the standard Bandpass filter is dead wrong at a cyclic valley at this time. Similarly, the truncated filter is above zero during the fourth quarter uptrend. The cyclic swings during August and September are also accurately reflected in the truncated indicator. In fact, you can track the cyclic swings across the entire chart and see at a glance how they correlate with the price movements.

Figure 1. A Truncated Bandpass Filter is a Better Indicator of Price Action

In summary, truncating IIR filters solves two problems associated with IIR filters. First, initialization errors are eliminated. Second, dampened transient responses of the truncated filters provide a more reliable indication of the current price action.

Code Listing 1. Standard and Truncated Bandpass Filters

```
{
    BandPass Filter and Truncated Bandpass Filter
    (C) 2005-2020  John F. Ehlers
}

Inputs:
        Period(20),
        Bandwidth(.1),
        Length(10);  //must be less than 98 due to array size

Vars:
        L1(0), G1(0), S1(0), count(0),
        BP(0), BPT(0);

Arrays:
        Trunc[100](0);

//Standard Bandpass
L1 = Cosine(360 / Period);
G1 = Cosine(Bandwidth*360 / Period);
S1 = 1 / G1 - SquareRoot( 1 / (G1*G1) - 1);
BP = .5*(1 - S1)*(Close - Close[2]) + L1*(1 + S1)*BP[1] - S1*BP[2];
If CurrentBar <= 3 Then BP = 0;

//Truncated Bandpass
Trunc[Length + 2] = 0;
Trunc[Length + 1] = 0;
For count = Length DownTo 1 Begin
        Trunc[count] = .5*(1 - S1)*(Close[count - 1] - Close[count + 1]) + L1*(1 +
S1)*Trunc[count + 1] - S1*Trunc[count + 2];
End;
BPT = Trunc[1];  //convert to a variable

Plot1(BP);
Plot4(0);
Plot2(BPT);
```