Automatic Annotation Tutorial

Make annotation of graphics automatic, so the intended meaning is illuminated *at-a-glance*.

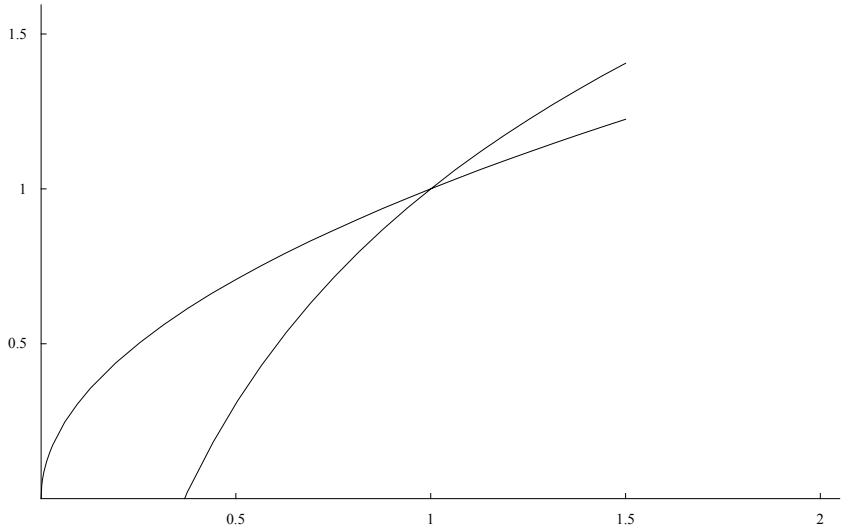
Roger Williams, 07/25/07, CongruentiaLuminaire@yahoo.com

The Problem

Normally a graph is presented in the newspaper or in an academic paper with the axes and tick marks labelled, but with the key points that the graph is to illuminate described in text. This defeats some of the advantage of the graph. In line with the thinking of Tufte, it would be better if the key points that the graph seeks to show were annotated with the graphic assisting in *at-a-glance* explanations.

Mathematica has extensive facilities to annotate graphic plots. In many texts, graphs are displayed in a bare fashion similar to this:

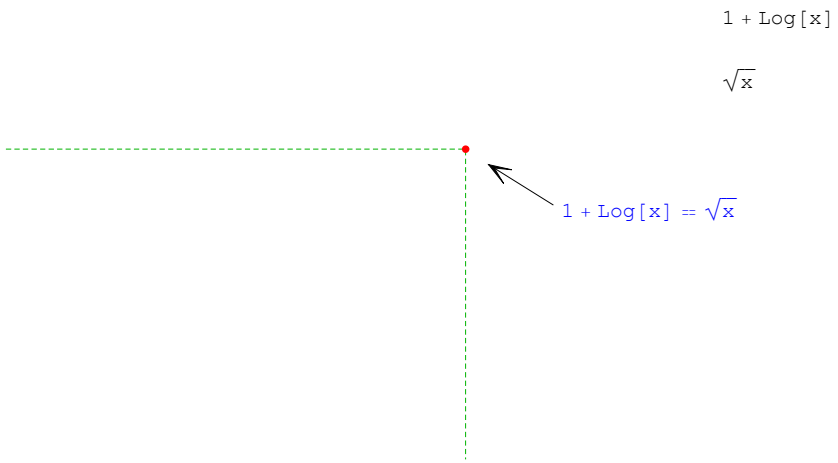
```
p1 = Plot[{Log[x] + 1, Sqrt[x]}, {x, 0, 1.5}, FormatType → TraditionalForm,
   TextStyle → {FontFamily -> "Times", FontSize → 7}, PlotRange → {{0, 2.05}, {0, Automatic}},
   ImageSize → 400, Ticks → {{0.5, 1, 1.5, 2}, {0.5, 1, 1.5}}];
```



In this example, some possible improvements include 1) adding reference lines where the functions are equal, 2) adding notations with the formula for the functions, among others. With *Mathematica,* one can also design these improvements as a set of annotations expressed as graphics primitives, thusly:

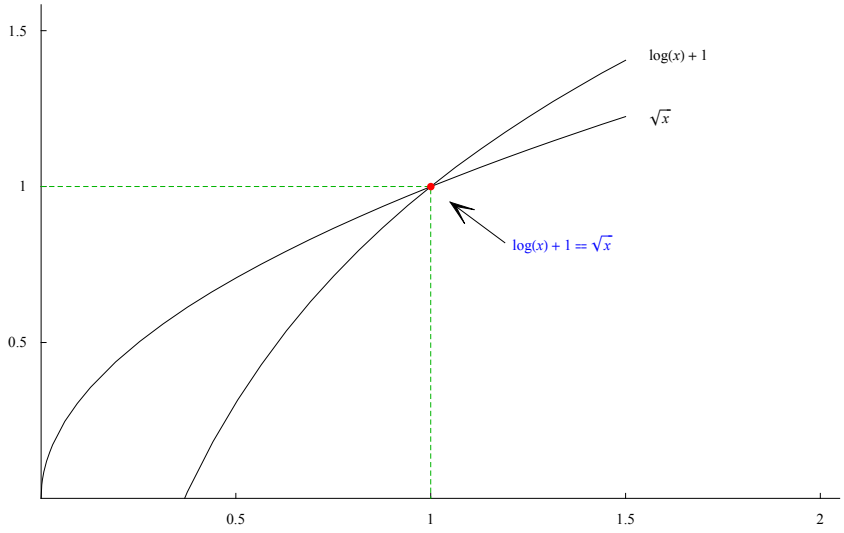
```
<< Graphics`Arrow`
```

```
p2 = Show[Graphics[{Text[Log[x] + 1, {1.55, 1.42}, {-1, 0}],
     Text[Sqrt[x], {1.55, 1.22}, {-1, 0}], {AbsoluteThickness[0.3],
      Arrow[{1.19, 0.82}, {1.05, 0.95}, HeadLength → 0.03, HeadWidth → 0.7, HeadCenter → 0.3]},
     RGBColor[0, 0, 1], Text[Log[x] + 1 == Sqrt[x], {1.2, 0.85}, {-1, 1}],
     RGBColor[0, 0.7, 0], AbsoluteDashing[{2}], Line[{{0, 1}, {1, 1}, {1, 0}}],
     RGBColor[1, 0, 0], AbsolutePointSize[3.5], Point[{1, 1}]}],
   ImageSize → 400, PlotRange → All];
```



Now these two graphics can be composed in order to produce an annotated graphic, incorporating the improvements described above.

```
Show[p1, p2];
```



This is fine and looks much better that the original bare graphic. One remaining issue is that the annotation primitives are *hard-coded*. The **Plot[]** routine accepts a different function easily, but the annotation code would have to have (at least) the coordinates changed. Obviously, if the functions are changed, the annotation code will no longer be correct to express the intended idea. For instance, the $\sqrt{x}$ annotation is at the point {1.55,1.22}. As an exercise, it is easy to re-evaluate the commands above to demonstrate when this annotation code would be incorrect.

This tutorial will show how to make the annotation code automatically update as the function is updated (albeit for a different type of example).

The Approach

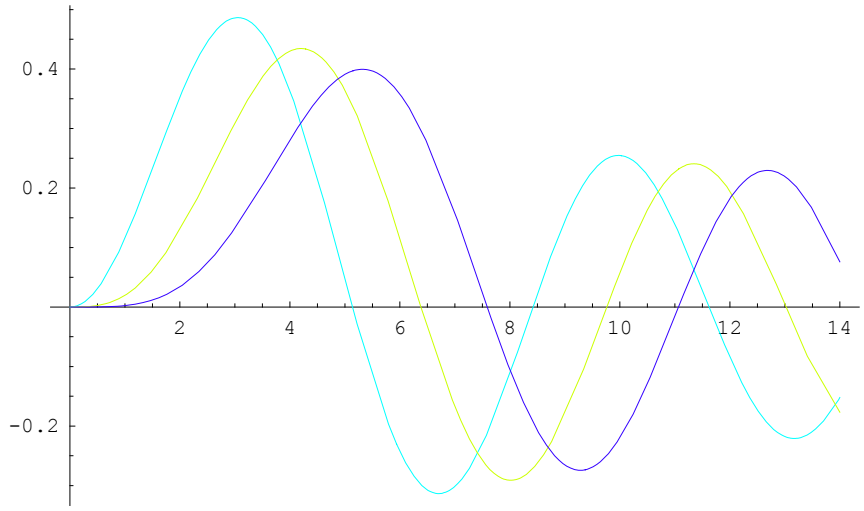
What we must do is to *parameterize* the annotation primitives to have them use the function as well as what we want the graphic to say in order for the annotation to be correct even when the function is changed. We will use a slightly different example to illustrate code of this type.

There are lots of charts and graphs in the literature of stocks and bonds. One widely described and measured quantity is termed a *moving average*. A moving average is the average price over a certain period of time of a stock (for example) with the average value sliding each day in keeping with the daily price of the stock. Typically, 15 and 30-day moving averages are computed and graphed.

You can see complicated moving average graphs in this Wikipedia entry. There is an investment method that can informally be described as *buy on the upslope when the price is between the 15 and 30-day moving averages*. Apparently this investment method is now discredited by the recent dot-com bust (as described in the entry). However we can use this example to show how our annotation code can work.

Using Bessel functions to simulate the averages, the graph below is a simulation of this concept:

```
g0 = Plot[{BesselJ[2, z], BesselJ[3, z], BesselJ[4, z]},
   {z, 0, 14}, ImageSize → 400, PlotStyle → {Hue[0.5], Hue[0.2], Hue[0.7]}];
```

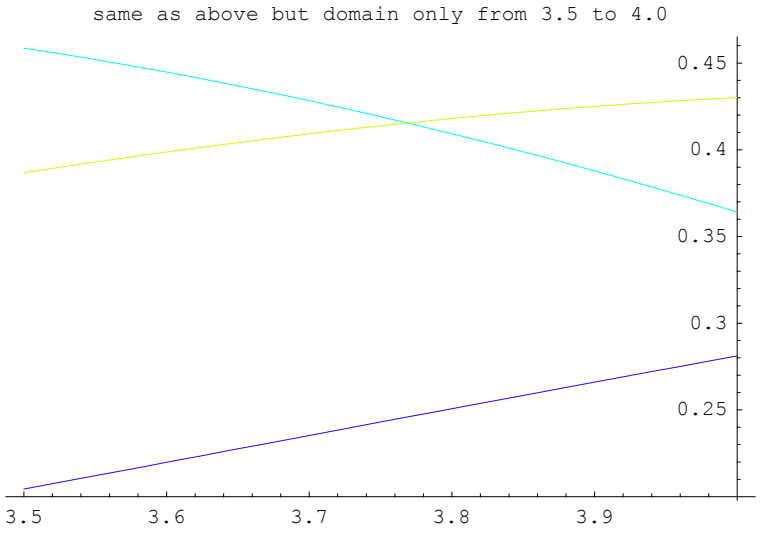



```
?BesselJ
```

BesselJ[n, z] gives the Bessel function of the first kind J(n, z). More...

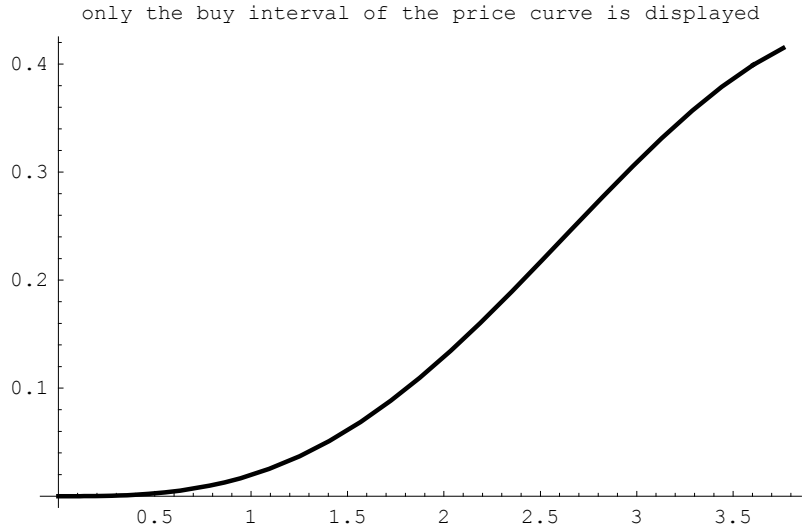
In this graphic the price is the lightest colored line, the 15 day average is the light blue line and the 30 day average is simulated by the dark-blue line. From eyeballing this graphic, I can see that the first buy section ends around 3.6. I can make this more precise, by zooming this graphic to get a better number.

```
Plot[{BesselJ[2, z], BesselJ[3, z], BesselJ[4, z]},
  {z, 3.5, 4.}, ImageSize → 400, PlotStyle → {Hue[0.5], Hue[0.2], Hue[0.7]},
  PlotLabel → "same as above but domain only from 3.5 to 4.0"];
```



From this graph **I can see** that the crossing point is about 3.76. So I can make an annotation graphic thusly:

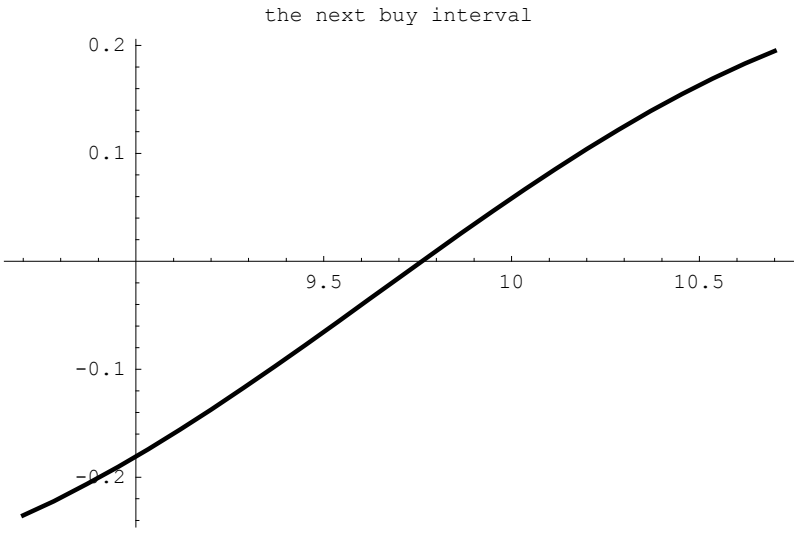
```
g1 = Plot[{BesselJ[3, z]}, {z, 0, 3.76},
   ImageSize → 400, PlotStyle → {AbsoluteThickness[2.1], Hue[0.2]},
   PlotLabel → "only the buy interval of the price curve is displayed"];
```



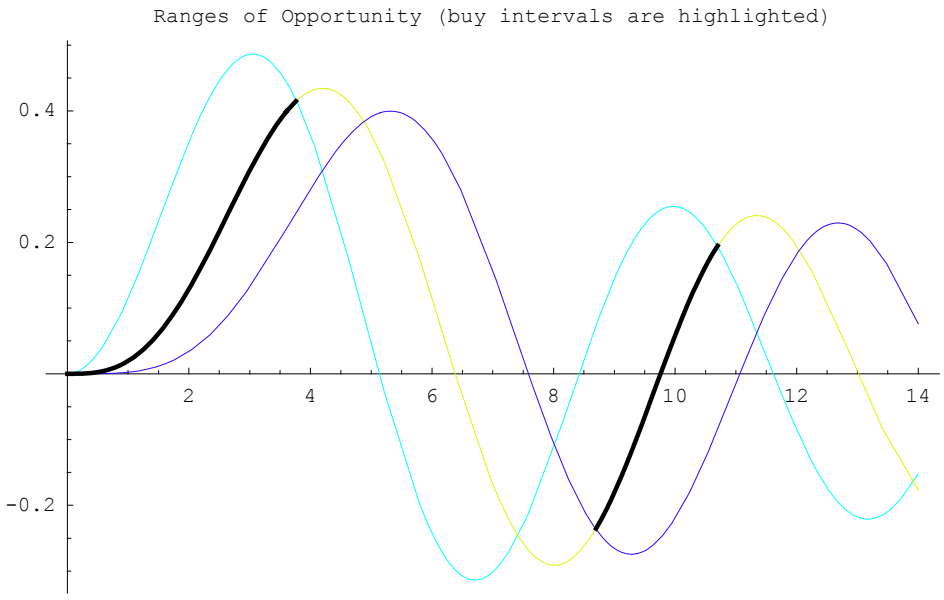
This is the section of the graph showing the first buy opportunity according to this investment method. The important point to note is the *I can see* part of the statement above. The 3.76 is a number that would have to be changed when using the same investment methods but some different average and price data.

So our approach will be to use *Mathematica*'s extensive group of **Solve**rs to determine the intervals that meet the criteria for this method and create annotations that will be shown with the plot and can be seen and understood at a glance. To end this section, here is how the annotated graph looks with hard-coded intervals.

```
g2 = Plot[{BesselJ[3, z]}, {z, 8.7, 10.7}, ImageSize → 400,
   PlotStyle → {AbsoluteThickness[2.1], Hue[0.2]}, PlotLabel → "the next buy interval"];
```




```
Show[g0, g1, g2,
  PlotLabel → "Ranges of Opportunity (buy intervals are highlighted)", ImageSize → 450];
```

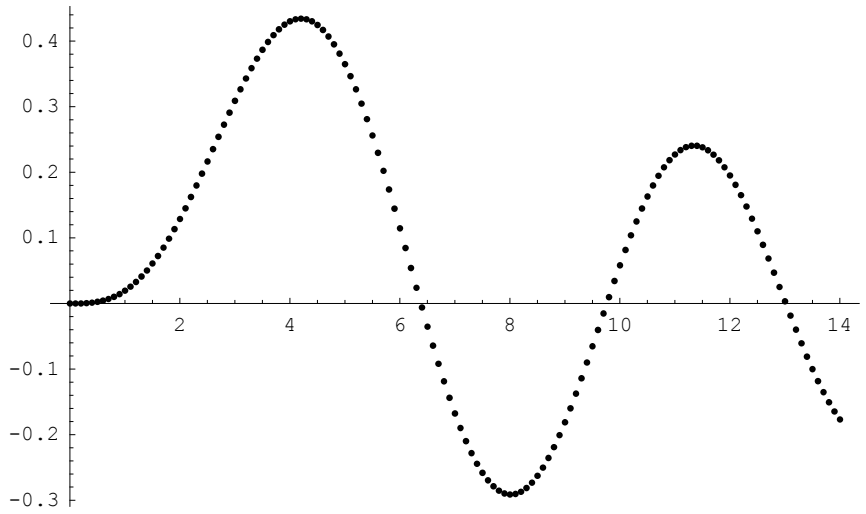


The Data

In the example above, the data was plotted as the range of a Bessel Function. Normally, the data is observed and measured and there is no function of the data that is known apriori. For this example, we will take the points from the BesselJ function and do a Chebyshev Approximation to those data points. This approximation method fits the original curves fairly well except at the endpoints.

The goal is to solve an inequality to determine the intervals where the current price is less than the 15-day moving average and greater than the 30-day moving average. One approach to annotate the graph is to eye-ball the curve like we did above. To annotate the graph a better approach is to use the *Mathematica* functions. To start, here is how the price data looks at the plotted points:

```
ListPlot[Table[{x, BesselJ[3, x]}, {x, 0, 14, 0.1}], ImageSize → 400];
```



We intend to do 6th degree Chebyshev intepolation using the **RationalInterpolation[]** function.

```
<< NumericalMath`Approximations`
```

```
?RationalInterpolation
```

```
RationalInterpolation[func, {x, m, k}, {x1, x2, ..., xn}, (opts)], (n = m+k+1), gives the
   rational interpolant to func (a function of the variable x), where m  and k are
   the degrees of the numerator and denominator, respectively, and {x1, x2, ..., xn}
   is a list of m+k+1 abscissas of the interpolation points. An alternative form is
 RationalInterpolation[func, {x, m, k}, {x, x0, x1}, (opts)], which
   specifies the list of abscissas implicitly: the abscissas come from
   the interval (x0,x1).  The function func must be Listable. More...
```

```
cheb6Fif = RationalInterpolation[BesselJ[2, x], {x, 6, 0}, {x, 0, 14}]
```

$$0.063711 - 0.451774\,x + 0.673088\,x^2 - 0.24969\,x^3 + 0.0376122\,x^4 - 0.00250038\,x^5 + 0.0000610006\,x^6$$

```
cheb6Prc = RationalInterpolation[BesselJ[3, x], {x, 6, 0}, {x, 0, 14}]
```

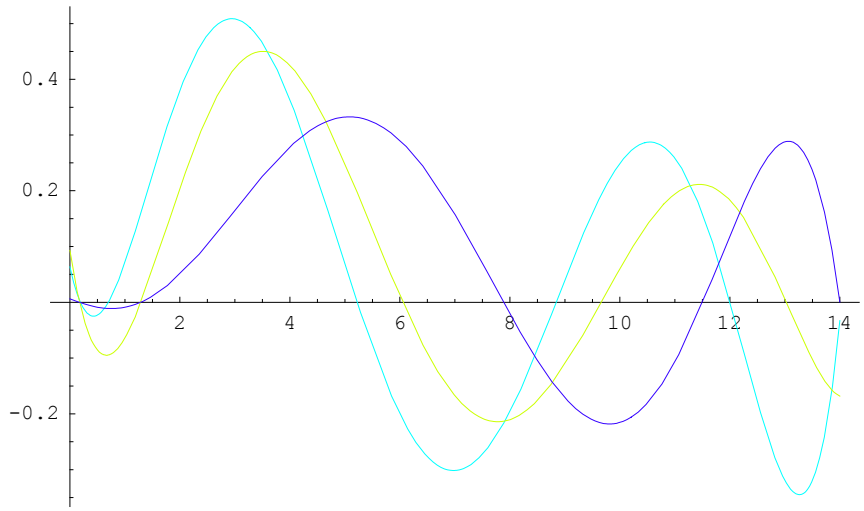
$$0.0931478 - 0.64037\,x + 0.663655\,x^2 - 0.206262\,x^3 + 0.0273512\,x^4 - 0.00163612\,x^5 + 0.0000363676\,x^6$$

```
cheb6Thr = RationalInterpolation[BesselJ[4, x], {x, 6, 0}, {x, 0, 14}]
```

$$0.00627797 - 0.0373929\,x + 0.00552682\,x^2 + 0.022739\,x^3 - 0.00646091\,x^4 + 0.000590376\,x^5 - 0.0000175678\,x^6$$

These 3 equations are 6th degree Chebyshev approximations to the Bessel function. By plotting these, we can see that these polynomial equations have a similar shape to the Bessel functions. The points from these functions are the ones we will use to annotate out graph.

```
p0 = Plot[{cheb6Fif, cheb6Prc, cheb6Thr}, {x, 0, 14},
   PlotStyle → {Hue[0.5], Hue[0.2], Hue[0.7]}, ImageSize → 400];
```



The Solution

Now we need to use *Mathematica*'s **InequalitySolve[]** to determine the intervals. Originally, I tried to do this with the Bessel functions directly.

```
<< Algebra`InequalitySolve`
```

```
InequalitySolve[BesselJ[2, x] ≥ BesselJ[3, x] ≥ BesselJ[4, x] , x]
```

```
InequalitySolve::nineq :
 BesselJ[2, x] ≥ BesselJ[3, x] ≥ BesselJ[4, x] is not a formula constructed
   with univariate polynomial equations and inequalities in x.

InequalitySolve[BesselJ[2, x] ≥ BesselJ[3, x] ≥ BesselJ[4, x], x]
```

but **InequalitySolve[]** requires polynomial equations. That's why we used the approximations above. Using these alternate arguments yields:

```
intervals = InequalitySolve[cheb6Fif ≥ cheb6Prc ≥ cheb6Thr , x]
```

```
x ≤ -1.72308 || 0.155709 ≤ x ≤ 0.175785 || 1.29679 ≤ x ≤ 3.60036 || 8.7748 ≤ x ≤ 11.2884 || x ≥ 14.2
```

Now this is what we want! The problem is how to convert this into range specifications for the **Plot[]** command.

```
? Plot
```

```
Plot[f, {x, xmin, xmax}] generates a plot of f as a function of x from xmin to
   xmax. Plot[{f1, f2, ... }, {x, xmin, xmax}] plots several functions fi. More...
```

So we need to make the intervals (such as 1.29679 <= x <= 3.60036) into a list like {x,1.29679,3.60036). This is the point where *Mathematica*'s consistency of representation is so remarkable. Let's look under the hood of what **InequalitySolve[]** returned.

```
Length[intervals]
```

```
5
```

```
intervals[[2]]
```

```
0.155709 ≤ x ≤ 0.175785
```

```
intervals[[2, 1]]
```

```
0.155709
```

```
intervals[[2, 2]]
```

```
LessEqual
```

So the values returned by **InequalitySolve[]** is really just another list that we can re-formulate into something that the **Plot[]** function wants. Now we can transform one of the inequalities in the list of intervals and make a range specification directly as follows:

```
range0 = {intervals[[2, 3]], intervals[[2, 1]], intervals[[2, 5]]}
```

```
{x, 0.155709, 0.175785}
```

Now we can extract the other ranges in the intervals list.

```
range1 = {intervals[[3, 3]], intervals[[3, 1]], intervals[[3, 5]]}
```

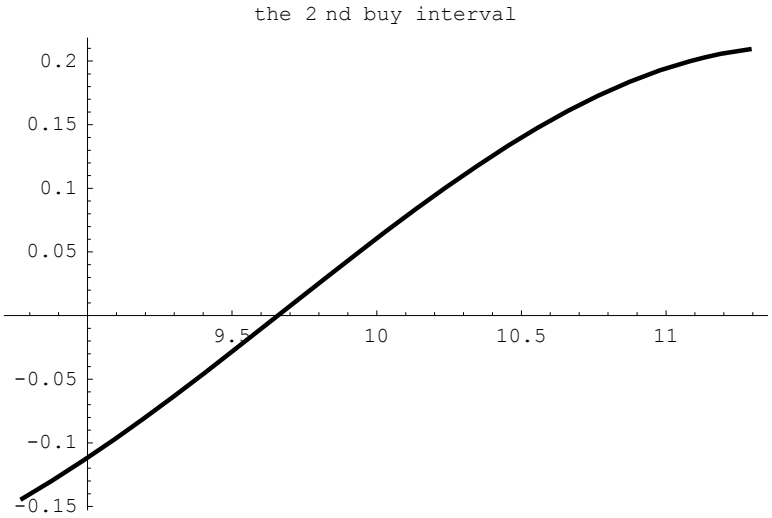
```
{x, 1.29679, 3.60036}
```

```
range2 = {intervals[[4, 3]], intervals[[4, 1]], intervals[[4, 5]]}
```

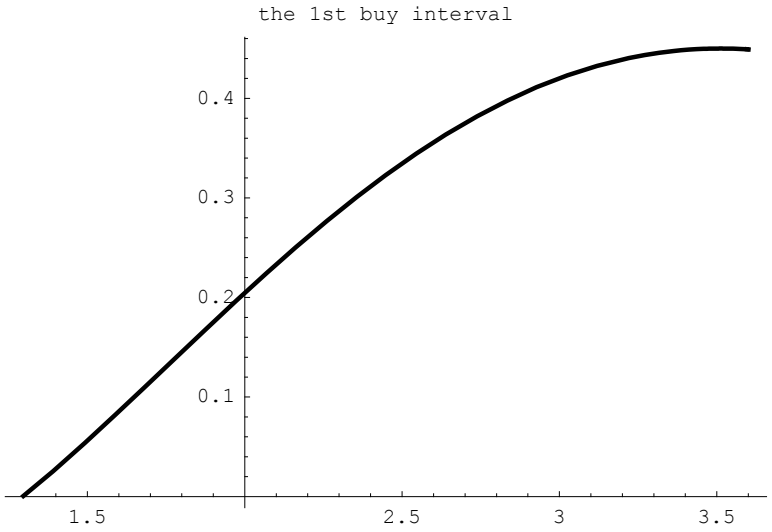
```
{x, 8.7748, 11.2884}
```

Here is an example of the price data plotted in the third buy interval:

```
p2 = Plot[cheb6Prc, Evaluate[range2], ImageSize → 400,
   PlotStyle → {AbsoluteThickness[2.1]}, PlotLabel → "the 2nd buy interval"];
```

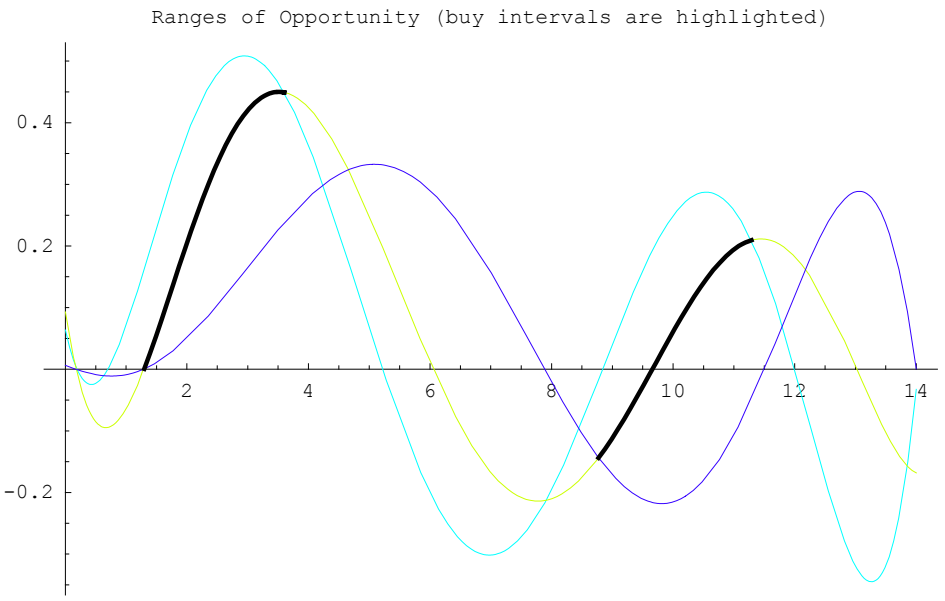



```
p1 = Plot[cheb6Prc, Evaluate[range1], ImageSize → 400,
   PlotStyle → {AbsoluteThickness[2.1]}, PlotLabel → "the 1st buy interval"];
```



Notice how in both of the Plot[] statements above the range is a variable and not hard-coded. This is eactly what we were trying to achieve.

```
Show[p0, p1, p2,
  PlotLabel → "Ranges of Opportunity (buy intervals are highlighted)", ImageSize → 450];
```



Conclusion

We have shown how to do automatic annotation of our price curves in a way that makes the buy regions visible *at-a-glance*. Furthermore, *Mathematica*'s consistency and modularity of object represenation was demonstrated to automate the process of annotation. In the education offerings and in the The *Mathematica* Book, it is always stressed that in the *Mathematica* CAS, *every object is an expression and every expression has a consistent and modular representation*. This is why we were able to extract parts of the return value of one function and create an argument to another function.

In a future tutorial, we will learn how to combine these steps into a callable function.

References

Maeder, Roman E., Computer Science with *Mathematica*, 2000
Ruskeepaa, Heikki, Mathematica Navigator, 2004
Tufte, Edward R., The Visual Display of Quantitative Information, 1983
Wolfram, Stephen, The *Mathematica* Book, 1999