Perl modules help solving and plotting formulas

Easy Algebra

Millions of students shudder at the thought of algebra classes – they are just so unreal. Instead of poring over your formula sheet, why not let Perl come up with the answers? This issue shows a new module to solve symbolic equations and another one to draw neat graphs to match.

BY MICHAEL SCHILLI

Throughout our lives we are continually confronted with problems that use a variable x. No matter what you are doing, whether attempting to calculate your fuel consumption, or looking at hares and foxes, the principles of algebra can be applied. Solving equations by hand, however, can be error-prone. To automate the process, CPAN offers a module that helps juggling with formulas.

Let's look at a simple formula as our first example. In the UK and the US, automobile manufacturers quote consumption figures in miles per gallon. This seems strange to people living in continental Europe, who expect these figures in liters of gasoline per 100 kilometers traveled. Thus, in continental Europe, a higher figure shows poor consumption.

Listing 1: *mpgal*

```
01 #!/usr/bin/perl
02 ##################################
03 # mpgal - miles/gallon =>
04 #         liters/100km
05 # Mike Schilli, 2004
06 # (m@perlmeister.com)
07 ##################################
08 use warnings;
09 use strict;
10
11 use Math::Algebra::Symbols;
12
13 my ($gallons, $miles) =
14   symbols(qw(gallons miles));
15
16 my $liters = $gallons *
17             37854118/10000000;
18 my $kilometers = $miles *
19               1609344/1000000;
20 my $usage = $liters /
21             $kilometers * 100;
22
23 print "Formula: $usage\n";
24
25 for $miles (qw(20 30 40)) {
26
27     $gallons = 1;
28
29     printf "$miles m/gal: " .
30            "%4.1f l/100km\n",
31            eval $usage;
32 }
```

Helping the Europeans Understand

Our task is to convert typical miles per gallon consumption figures into continental liters/100 kilometer figures. The mpgal script in Listing 1 uses the *Math::Algebra::Symbols* module from CPAN and defines two symbols, *$gallons* and *$miles*. The formula is built up step by step, starting in line 16. The expression defines a gallon as 3.7854118 liters.

There are two things to watch out for: firstly, *Math::Algebra::Symbols* in the current version 1.16 does not handle long floating point values gracefully, forcing programmers to use fractions instead. Secondly, *$liters* specifies the number of liters consumed. Thus, we

need to multiply the number of gallons by 3.7854118. Two gallons thus correspond to 7.5708236 *$liters*. The same principle applies to kilometers and miles, where a mile is defined as 1.609344 kilometers.

The formula in line 20 provides the fuel consumption per 100 kilometers. As we have already defined formulas for *$liters* and *$kilometers* in the script, *Math::Algebra::Symbols* substitutes the symbols and generates an equation that only needs *$gallons* and *$miles*. Line 23 outputs the results:

```
Formula: 94635295/402336*
$gallons/$miles
```

Math::Algebra::Symbols uses fractional maths to cancel down the constants. Mpgal then assigns values to the variables *$gallons* and *$miles* to provide real values for the formula, and calls *eval $usage* in line 31 to apply the formula. The script then prints typical consumption figures in comparison:

```
20 m/gal: 11.8 l/100km
30 m/gal: 7.8 l/100km
40 m/gal: 5.9 l/100km
```

Sure, a simple Perl function could have done this easily. However, *Math::Algebra::Symbols* can even resolve variables in more complex formulas, as the example in Listing 2 shows.

Hares and Foxes

The next task is is a classical text-based problem: At a distance of 10 meters, a fox notices a hare running away at a constant speed of 5 meters per second, and takes up the pursuit. The fox accelerates at 14 meters per second/per second. How long does it take for the fox to catch up with the hare?

To solve this problem, the Race program in Listing 2 defines the symbol *$t* for the elapsed time in seconds, and describes the distance covered by the hare and the fox in relation to the elapsed time (lines 14 and 15):

```
my $hare = 10 + 5 * $t;
my $fox = 7 * $t * $t;
```

The hare has a head start of 10 meters, and the distance covered can be derived by applying the formula for constant speed (s = v * t). At a given time *t*, the hare will have covered a distance of 10 + 5 * t, whereas applying the formula for constant acceleration (s = a / 2 * t^2) tells us that the fox will have caught up exactly 7 * t^2 meters. The fox catches the hare when the two distances are equal, that is at the point where the equation in line 17 returns a value of zero.

To work this out on paper, I would need to solve a quadratic equation, and that would mean hunting for that formula sheet from way back in my school days. But thanks to *Math::Algebra::Symbols*, we can resolve *$gotcha* simply by applying the *$gotcha->solve("t")* method to *$t*. As we are solving a quadratic equation, this returns a reference to an array with two symbolic equations (in line 20):

```
Solution: 1/14*sqrt(305)+5/14
Solution: -1/14*sqrt(305)+5/14
```

As negative time values have no practical effect on the well-being of the hare, we can discard the second result in line 25. To transform the solution into a floating point number, Perl's *eval* is used in line 23.

Unfortunately for the hare, the chase is over after about 1.60 seconds. After setting the symbolic variable *$t* to this value in line 30, we also have the distance covered by the fox: *eval $fox* returns approximately 18.02 meters.

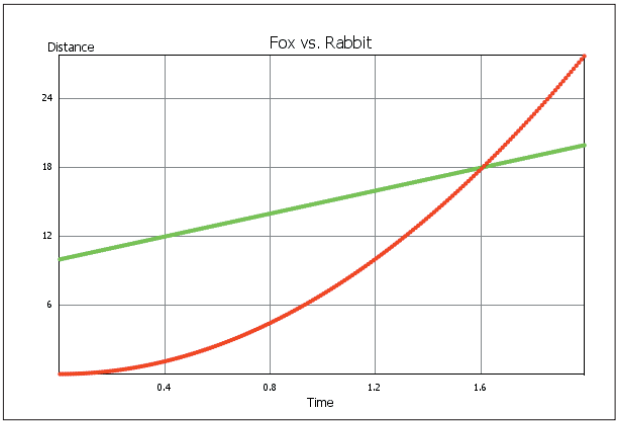

Figure 1: With its constant acceleration, the fox catches the hare, which is running away at a constant speed, after about 1.6 seconds, despite the hare's head start of 10 meters. Two Perl modules simplify the task of programming the function and provide a plot of the results.

Graphics Wizardry

Listing 3 shows a graphic version of the chase, as shown in Figure 1. The Imager::Plot module, and a few lines of Perl code, allow you to create professional plots in various image formats. Line 14 creates a new Imager::Plot object using the tahoma.ttf Truetype font in the specified path for the legends. Your installation might vary, please adjust the path accordingly.

The *for* loop starting in line 24, iterates in steps of 0.01 through the X values 0.0

Listing 2: *race*

```
01 #!/usr/bin/perl
02 ##################################
03 # race - Fox chasing a hare
04 # Mike Schilli, 2004
05 # (m@perlmeister.com)
06 ##################################
07 use warnings;
08 use strict;
09
10 use Math::Algebra::Symbols;
11
12 my ($t) = symbols(qw(t));
13
14 my $hare = 10 + 5 * $t;
15 my $fox =   7 * $t * $t;
16
17 my $gotcha = ($hare - $fox);
18
19 for my $solution
20   (@{$gotcha->solve("t")}) {
21   print "Solution: ",
22         "$solution\n";
23   my $val = eval $solution;
24   if($val < 0) {
25     print "Discarded\n";
26     next;
27   } else {
28     printf "%.2f seconds\n",
29            $val;
30     $t = $val;
31     printf "%.2f meters\n",
32            eval $fox;
33   }
34 }
```

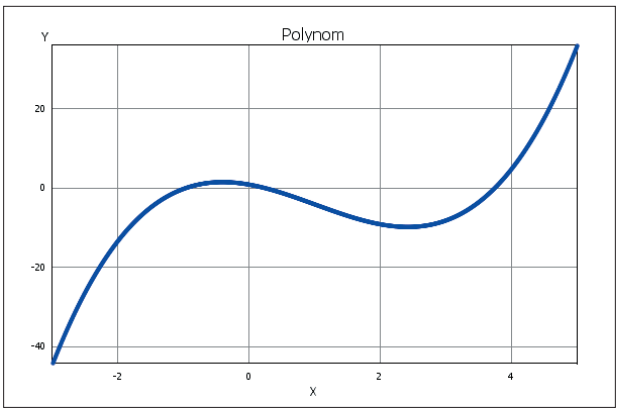

Figure 2: The plot of the polynomial $t^3 - 3t^2 - 3t + 1$. The *Math::Algebra::Symbols* Perl module helps programmers who are too lazy to do the math involved, to determine maxima and minima.

through 2.0, creating three arrays: *@t* for the X axis values, and *@hare* or *@fox* for the location values for the hare and the fox. The location values are mapped to time values by applying the motion formulas.

Line 32 inserts the green line for the hare into the coordinate system, lines 44 and following add the red line for the fox. The *Render* function in line 69 handles the drawing, and *write* in line 72 writes the whole thing to a PNG file.

The fox and hare task doesn't exhaust the range of features that *Math::Algebra::Symbols* has to offer. Figure 2 shows the graph of $y = t^3 - 3t^2 - 3^t + 1$. Its two amplitudes show local maximum and minimum values. At school you are taught that the gradient of the curve at these points is zero. To be able to determine the corresponding X values, you need to differentiate the function, make its result equal to zero, and resolve the equation created in this step. Fortunately, *Math::Algebra::Symbols* can differentiate simple functions:

```
my ($x) = symbols('x');

my $y = $x**3 - 3*$x**2 - 3➥
*$x + 1;
my $diff = $y->d('x');

my $extrema = $diff->solve('x');
print eval($_), "\n" ➥
for @$extrema;
```

The *$y->d('x')* method call differentiates the function defined in *$y* to *$x*, transforming the cubic polynomial into a quadratic. The call to the *solve()* method solves the latter and returns a reference to an array with two elements, which *eval* transforms into floating point values, as seen previously:

```
-0.414213562373095
2.41421356237309
```

That gives us the x-values of the extremes. *Math::Algebra::Symbols* can also handle any number of trigonometric functions, although it does tend to choke on more complex structures.

Caution, Men at Work!

Math::Algebra::Symbols and *Imager::Plot* are both available from CPAN, and are excellent for messing around with mathematical puzzles. *Math::Algebra::Symbols* is still very much alpha, but its author, Philip Brenan, is continuously improving it. Maybe, one day, it will be as powerful as Mathematica. The moral of today's math lesson is: Do your math, and learn something for life! ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/46/Perl*

Listing 3: graph

```
01 #!/usr/bin/perl
02 ##################################
03 # graph – Graph of fox/hare
04 #          chase
05 # Mike Schilli, 2004
06 # (m@perlmeister.com)
07 ##################################
08 use strict;
09 use warnings;
10
11 use Imager;
12 use Imager::Plot;
13
14 my $plot = Imager::Plot->new(
15   Width  => 550,
16   Height => 350,
17   GlobalFont =>
18   '/usr/share/fonts' .
19   '/truetype/tahoma.ttf');
20
21 my (@t, @hare, @fox);
22
23     # Generate function data
24 for(my $i = 0.0; $i < 2.0;
25     $i += 0.01) {
26     push @t, $i;
27     push @hare, 10 + 5 * $i;
28     push @fox, 7 * $i * $i;
29 }
30
31     # Add hare plot
32 $plot->AddDataSet(
33   X => \@t,
34   Y => \@hare,
35   style => {
36     marker => {
37       size  => 2,
38       symbol => 'circle',
39       color =>
40       Imager::Color->new(
41                   'green')}});
42
43     # Add fox plot
44 $plot->AddDataSet(
45   X => \@t,
46   Y => \@fox,
47   style => {
48     marker => {
49       size   => 2,
50       symbol => 'circle',
51       color =>
52       Imager::Color->new(
53                   'red')}});
54
55 my $img = Imager->new(
56   xsize => 600,
57   ysize => 400);
58
59 $img->box(filled => 1,
60           color => 'white');
61
62     # Add text
63 $plot->{'Ylabel'} =
64                   'Distance';
65 $plot->{'Xlabel'} = 'Time';
66 $plot->{'Title'} =
67               'Fox vs. hare';
68
69 $plot->Render(Image => $img,
70     Xoff => 40, Yoff => 370);
71
72 $img->write(
73         file => "graph.png");
```