Mathematical Expression Handling With Perl



Jonathan Worthington

YAPC::EU::2005

Mathematical Expression Handling

Overview

This talk looks at a number of issues relating to working with math expressions in Perl.

- Analytical vs. numerical methods
- Ways of representing expressions
- The Math::Calculus::Expression module
- Modules implementing differentiation, Newton Raphson and Taylor series.
- Expression equivalence

Mathematical Expression Handling

Analytical vs. Numerical

What's the difference?

- Analytical methods work with the expressions themselves, a bit like when you are doing algebra or calculus on paper. The result could be another expression.
- Numerical methods evaluate expressions and then work with the numbers. The result will always be a number.

Mathematical Expression Handling

Analytical vs. Numerical

Why might an analytical method be useful?

- It can potentially give an exact result by avoiding floating point calculations that a numerical method would have to do.
- It can give a more general result – numerical ones are often specific to a certain problem.
- Good for checking work done by hand or even automating it.

Mathematical Expression Handling

Analytical vs. Numerical

When are analytical methods likely not useful?

- When performance matters
- When a numerical method is sufficient for the task at hand

Sometimes a mixture of the two is called for.

- A program that takes an expression from a user needs to parse and evaluate it.
- A numerical method may then be used.

Mathematical Expression Handling

Expression Representation

On paper, expressions are written in mathematical notation.

$$\cos(\frac{\exp(a^2 - x^2)}{x + 2a})$$

This is usually translated to a form that is easy to type on a standard keyboard using * for multiplication, / for division and ^ for powers.

```
cos(exp(a^2 - x^2) / (x + 2*a))
```

Mathematical Expression Handling

Expression Representation

The modules discussed in this article accept and return expressions in a textual form (the second one shown on the previous page).

- Perl is great at manipulating text, so how about performing the operations on expressions by doing a series of string manipulations?

Mathematical Expression Handling

Expression Representation

Manipulating expressions while in textual form turned out to be a Bad Idea™.

- Much of Perl's strength with handling text comes through its regex support.
- Regexes can parse more than just regular languages, but they are still very much rooted in regular languages.
- Mathematical expressions are not a regular language (arbitrarily nested brackets).

Mathematical Expression Handling

Expression Representation

Manipulating expressions as text soon led to hard to read code and a very fragile system that was difficult to build on.

```
#These are instances of x or a linear function of x raised to a power.
if ($element =~ /^(-?)([.\w]*)$variable([.\w]*)$/) {
          #kx goes to k
          $element = $1 . ("$2$3" || 1);
} elsif ($element =~ /^(-?)([.\w]*)$variable\^(\-?\d+\.?\d*)$/) {
          #ax^n goes to anx^(n-1)
          $element = "$1$2" . ($2 ? '*' . $3 : $3) . $variable . '^' . ($3 - 1);
} elsif ($element =~ /^(-?)([.\w]*)\(([+\-]?[.\w]*)$variable([.\w]*[+\-]?[.\w]*)\)\^([+\-
]?[.\w]+)$/) {
          #a(bx+c)^n goes to abn(bx-c)^(n-1)
          my $power = $5;
          $element = "$1$2*$power*$3($3$variable$4)^" . ( $power =~ /^[.\d]+$/ ? $power - 1 :
"($power-1)");
```

Mathematical Expression Handling

Expression Representation

The solution is to use a different internal representation.

- Write a routine that converts the user-visible format into the internal representation.
  - You could think of this as a parser.
- Write a routine that converts the internal representation into the user-visible one.
  - You could think of this as a pretty-printer.

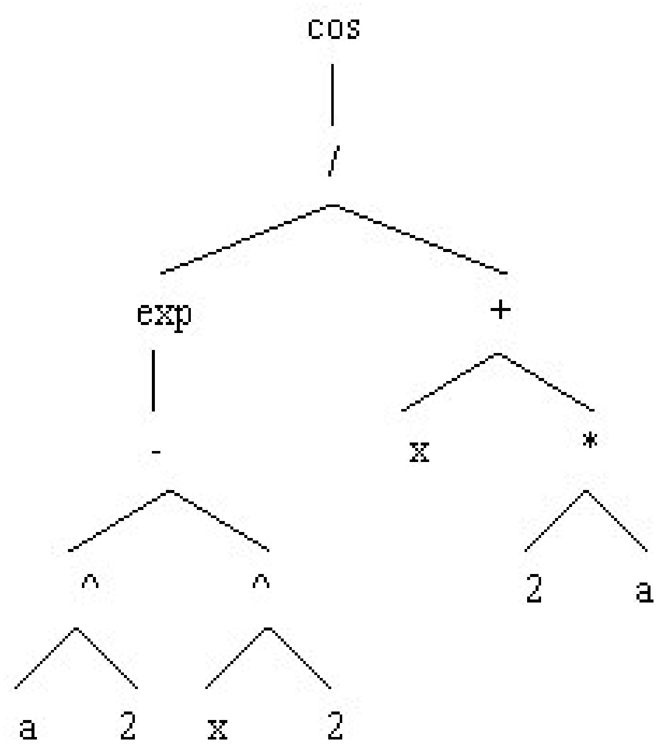
Mathematical Expression Handling

Expression Representation

I chose to represent expressions using non-strict binary trees.

The tree to the right represents:

`cos(exp(a^2 - x^2) / (x + 2*a))`



A node of the tree can either be a constant, a variable, an operator (+, -, *, /, ^) or a function.

Mathematical Expression Handling

Expression Representation

Why is the tree representation a good idea?

- No need for code manipulating the tree to worry about precedence (or bracketing) – it's encoded as tree depth.
- Many problems (evaluation, differentiation) are neatly represented using recursion, and recursion is cheap on a tree structure such as this one.
- Trivial to extract sub-expressions.

Mathematical Expression Handling

Expression Representation

There are a few things to be aware of with regard to the tree representation.

- White space **will not** be preserved.
- Extraneous brackets **will not** be preserved.
- The meaning of the expression **will** be preserved.

Mathematical Expression Handling

Math::Calculus::Expression

This OO module provides some of the most basic expression manipulation functionality:

- Taking an expression as text, parsing it and building the internal expression tree
- Turning the expression tree back into text
- Evaluating the expression (to a number)
- Doing some basic simplifications
- Testing if two internal representations match

Mathematical Expression Handling

Math::Calculus::Expression

Here's an example of using the module.

```
# Create an expression object.
use Math::Calculus::Expression;
my $exp = Math::Calculus::Expression->new;

# Set an expression and set its variable.
$exp->setExpression('2*x^2 + sin(2*t - x) + 10');
$exp->addVariable('x');

# Evaluate it with x = 4, t = 2.
my $val = $exp->evaluate(
      x => 4,
      t => 2
);
print "Evaluates to $val"; # 42
```

Mathematical Expression Handling

Math::Calculus::Expression

If you call a method not implemented by this module (or the subclass of it that you’re using) then it attempts to be helpful.

- By convention, the most significant method a module adds will have the same name as the module itself, apart from an initial lowercase letter.
- So the differentiate method is implemented in Math::Calculus::Differentiate.

Mathematical Expression Handling

Math::Calculus::Expression

This standard naming scheme makes it straightforward to Do The Right Thing.

- AUTOLOAD is implemented. It takes the name of the method being called and tries to load the appropriate module.
- If the module can be loaded, a call is made into that module, passing the current expression object into it.
- Basically fakes runtime class composition.

Mathematical Expression Handling

Math::Calculus::Expression

The binary tree is actually made up of hashes with keys operation, operand1 and operand2.

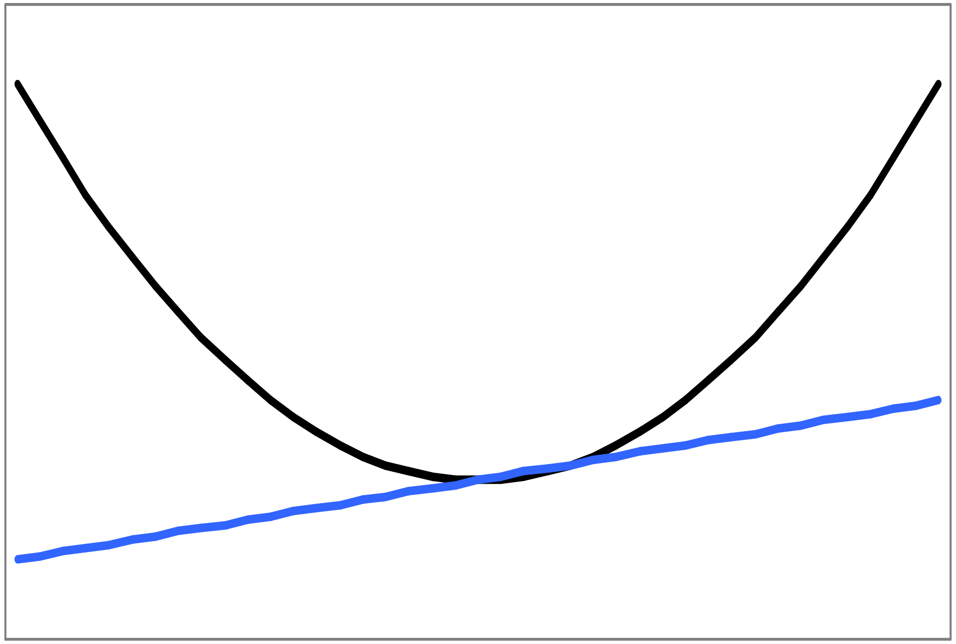
- Branches are simply hashrefs.
- At the bottom of the tree, instead of having a hashref to another node, a letter or number is stored. Thus it is possible to check if the branch is a subtree simply by using `ref`.
- Not the cheapest solution, but readable and allows the tree to be augmented with ease.

Mathematical Expression Handling

Math::Calculus::Differentiate

The derivative of an expression describes its gradient - how steep the curve is at each point.

- The black line is the function x^2.
- The blue line is the gradient of x^2, which works out to be $2x$.



Mathematical Expression Handling

Math::Calculus::Differentiate

This module implements differentiation and is a subclass of Math::Calculus::Expression.

- It adds the differentiate method, which transforms the currently represented expression into its derivative.
- At this time the module only does partial differentiation - that is, differentiation with respect to a single variable. Other variables will be treated like constants.

Mathematical Expression Handling

Math::Calculus::Differentiate

```
# Create an expression object, set up an example
# expression and set its variable.
use Math::Calculus::Differentiate;
my $exp = Math::Calculus::Differentiate->new;
$exp->setExpression('2*x^2 + sin(2*t - x) + 10');
$exp->addVariable('x');

# Differentiate with respect to x. This prints:
# 2*2*1*x^(2 - 1) + (2*t - 1)*cos(2*t - x) + 0
$exp->differentiate('x');
print $exp->getExpression . "\n";

# If we simplify it, things get cleaner. This prints:
# 4*x + (2*t - 1)*cos(2*t - x)
$exp->simplify.
print $exp->getExpression . "\n";
```

Mathematical Expression Handling

Math::Calculus::Differentiate

Differentiation is implemented recursively.

- Feels quite natural – maps well to the chain rule and its results.
- For example, the rule for differentiating an expression, ***e***, to a constant power involves differentiating ***e*** itself.

$$\frac{d}{dx}[(e)^n] = n\frac{d}{dx}[e](e)^{(n-1)}$$

- Example code can be found in the paper.

Mathematical Expression Handling

Math::Calculus::NewtonRaphson

This module implements the Newton Raphson method.

- Newton Raphson is a numerical method for finding a solution to an equation.

- Must be in the form $f(x) = 0$, where $f(x)$ is an expression (which we can represent).

Mathematical Expression Handling

Math::Calculus::NewtonRaphson

Newton Raphson is an iterative method.

- Takes an initial estimate of the result, feeds it into the iteration and gets a better estimate.
- Usually a stable iteration with quadratic convergence.
- The iteration involves the derivative of the function – which we can find analytically now!

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Mathematical Expression Handling

Math::Calculus::NewtonRaphson

Here's an example of using the module to solve x^x + sin(2*x) = 7.

```
# Create an expression object.
use Math::Calculus::NewtonRaphson;
my $exp = Math::Calculus::NewtonRaphson->new;

# Set an expression and set its variable.
$exp->setExpression('x^x + sin(2*x) - 7');
$exp->addVariable('x');

# Attempt to solve it with initial guess 3.
my $sol = $exp->newtonRaphson('x', 3);
print "Solution is $sol\n"; # 2.38828587710838
```

Mathematical Expression Handling

Expression Equivalence

Two mathematical expressions are equivalent if they are equal when evaluated for all possible values of their variable(s).

- Essentially, if two expressions are equivalent, they can always be used in place of each other.
- Testing whether the two expressions evaluate to the same thing for every value is infeasible – need something else.

Mathematical Expression Handling

Expression Equivalence

Does having the same internal representation say anything about equivalence?

- Yes! Obviously, two expressions with the same representation are equivalent
- Cheap to implement.
- However, it is possible for two expressions with different representations to be equivalent, e.g. 2*x and x + x have different internal representations but are equivalent.

Mathematical Expression Handling

Expression Equivalence

What about re-arranging the expression using a certain set of rules?

- An ordering scheme can help with identifying, for example, “x+2” and “2+x” as equivalent. Apply from the bottom of the tree.
- To identify “(x + 1)*(x - 1)” and “x^2 – 1” as equivalent, multiply out brackets and simplify.
- Despite growing complexity, still has no chance of determining sin(x)/cos(x) = tan(x).

Mathematical Expression Handling

Expression Equivalence

What we really want is to find a canonical form for representing expressions.

- A canonical form is one where all equivalent expressions have the same representation.
- A Taylor Series is such a form.
  - Represents any continuous, differentiable expression as an infinite polynomial.
  - n^{th} coefficient related to n^{th} derivative.

Mathematical Expression Handling

Expression Equivalence

We can use Taylor Series to investigate equivalence.

- If the Taylor Series of two expressions are equivalent, it can be said that the expressions themselves are equivalent.
- As the coefficients are found by evaluating the expression or its derivatives at a fixed point, two equivalent expressions will have the same Taylor series.

Mathematical Expression Handling

Expression Equivalence

Taylor series are infinite.

- Obviously cannot compute every co-efficient – would take infinite time and space.
- Instead, compute and compare the first N coefficients of the Taylor series.
- Size of N determines how accurate the equivalence testing needs to be.

Mathematical Expression Handling

Expression Equivalence

It isn't all plain sailing. Evaluating a large number of coefficients becomes expensive.

- Computing many derivatives is time consuming.
- For some functions the size of the derivative expression, under the currently available expression simplifier on CPAN at the time of writing, blows up exponentially.
- Also need to compute fast-growing factorial.

Mathematical Expression Handling

Expression Equivalence

However, it works!

- An implementation is available now on CPAN as Math::Calculus::TaylorEquivalent.
- It spots all of the equivalences mentioned in this talk so far, including the trigonometric identity.
- It was also very simple to write, especially having a TaylorSeries module already written.

Mathematical Expression Handling

Conclusions

- Only a handful of people here will actually need to do analytical manipulation of mathematical expression.
- However, some of the concepts are very portable to other fields of application – particularly the idea of a separate internal representation.
- Working web front ends to these modules on my site: http://www.jwcs.net/~jonathan/