Taking the Derivative ID1019

Azer Hojlas

January 31, 2022

1 Introduction

The task at hand is to compute the derivative of most common functions. With the help of high school level mathematical principals, we can with relevant ease extrapolate these formulas into a program and use it to calculate the derivative of said functions. The problem is not inherently in how to perform these calculations, but how to simplify and present the results so that the common client can interpret the result without a priori programming knowledge.

This assignment primarily differed in the change of focus from list-operations to tuple-operations. With no earlier knowledge of working with tuples, this task clearly illustrates their functionality and how to use atoms and integers in conjunction. I was somewhat amazed about how to use pattern matching on atoms in order to decide which function would process a certain expression.

I found it very interesting to find out if one could reverse engineer the simplify function. I figured that if the client has to input a complex data structure in order to receive their simplified result, then the opposite should hold true as well (e.g the client only needs to input a simple human-readable function, say "2x + 5" and the program will parse this to {:add, {:mul, {:num, 2}, {:var, {:num, 5}}). That is, the program could handle simplified input and transform said input into a function that the code can understand and process. I tried to implement such a function but it didn't yield any desirable results.

2 Code

As stated earlier, the challenging part of the assignment belongs to simplifying the expressions rather than calculating their derivative. As I couldn't implement simplify functions for the more complex expressions, I only implemented them in the form of deriv(). For the sake of demonstration (as my earlier report was insufficient), I managed to implement a partial simplify to ln(g(x) in order to go through it more in depth. Below I'll post the deriv() for the more complicated functions as proof of copmletion and the in-depth ln will be submitted further down in the text.

```
def deriv({:sin, {:var, x}}, x) do {:cos, {:var, x}} end

def deriv({:sin, {:mul, {:num, n}, {:var, x}}}, x) do {:mul, {:num, n},
{:cos,{:mul, {:num, n}, {:var, x}}}} end

def deriv({:exp, e, {:num, n}}, v) do{:mul, {:mul, {:num, n}, {:exp, e,
{:num,n-1}}}, deriv(e, v)} end

def deriv({:div,_,{:var,v}},v) do {:div,{:num,-1},{:exp,{:var,v},{:num,2}}} end
def deriv({:div, _, {:exp, {:var, x}, {:num, n} } }, x) do{:div, {:num, -n},
{:exp, {:var, x}, {:add, {:num, n}, {:num, 1}}}}end

def deriv({:sqrt, {:var, x}}, x) do {:div, {:num, 1}, {:mul, {:num, 2},
{:sqrt, {:var, x }}}} end
```

2.1 In-depth ln(g(x))

I found ln(g(x)) to be the easiest to implement simplify() and pprint() for, and even in that case I somehow didn't manage to do it entirely. It works however sufficiently to the extent that the reader can understand the process of execution.

@i

@i

The deriv() for ln has two cases, one is the most basic ln(x) which returns 1/x. The other one implements the chain rule where ln(g(x)) becomes (1/g(x))*g'(x). The simplify functions go through the expected base cases for division, e.g division with 1 and so forth, then regular division and finally the part that simplifies expressions. I did not manage to simplify the expressions entirely, but it is however clear enough for the naked eye to understand. The pretty print is pretty (pun intended) self-explanatory.

2.2 Demonstration of $\ln(x^2)$

Below you can see an example execution of $\ln(x^2)$. As you can see, the final result doesn't get simplified all the way, as it should be $2/b$ instead of $2b/b^2$. Otherwise it works just fine when computing a derivative.

```
iex(209)> c
{:ln, {:exp, {:var, :b}, {:num, 2}}}
iex(210)> b = Deriv.deriv(c, :b)
{:div, {:mul, {:mul, {:num, 2}, {:exp, {:var, :b}, {:num, 1}}}, {:num, 1}},
 {:exp, {:var, :b}, {:num, 2}}}
iex(211)> a = Deriv.simplify(b)
{:div, {:mul, {:num, 2}, {:var, :b}}, {:exp, {:var, :b}, {:num, 2}}}
iex(212)> Deriv.pprint(a)
"2b/b^(2)"
```

Here is an execution of the simple derivative of ln(b) with regards to "b".

```
iex(117)> c = {:ln, {:var, :b}}
{:ln, {:var, :b}}
iex(118)> a = Deriv.deriv(c, :b)
{:div, {:num, 1}, {:var, :b}}
iex(119)> Deriv.pprint(a)
"1/b"
```

2.3 Difficulties and additions

I did not have as much success when coding on my own. I am still somewhat confused as to how some functions get simplified to their simplest form even though it appears that they cannot be simplified any further. More specifically, how expressions get further simplified when they reach certain base cases, e.g:

```
def simplify_exp(e1, e2) do {:exp, e1, e2} end
```

They appear to get simplified without any further recursion. Should they not be returned as a tuple? Hopefully the peer reviewer of this assignment can answer that.
Apart from the largely untouched boilerplate-code from Johan, I only changed visuals like parentheses to be more easier on the eye for the viewer. This was the result from an arbitrary execution.

```
expression: (x^(3) + 4)
derivative: (3x^(2)1 + 0)
simplified: 3x^(2)
calculated: 48.0
:ok
```

I have not submitted the sin() example because it's pprint(), as with the other functions was too difficult to implement. You can however from my code derive that :sin, :var, :b becomes :cos, :var, :b in the deriv().

3 Conclusion

The assignment was difficult although a great learning experience. Apart from learning how to process tuples, I've been introduced to the IO functions that I will use in future assignments. This was not really an assignment in derivatives, but rather a assignment in illustrating trees in a comprehensible way. As mentioned before, the challenging part was pprint() and simplify(). The deriv() and calc() in themselves added nothing new in general knowledge, but the assignment was very pleasing to do in general.