Derivative

Björn Formgren

Spring 2023

Introduction

The first assignment in ID1019 consisted of creating symbolic expressions in elixir and create a program that could take the derivative out of the expressions. The skeleton of the program came from the lectures. Three different types of functions was created. The *derive* functions that controlled the actual differentiation. The *simplify* functions that simplified the expressions in different ways. Lastly, the *pprint* functions that took care of presenting the expressions in a nice way to the output.

Representing the expression

The student chose to represent the expressions the same way as in the lectures using tuples with atoms specifying the operation.

```
@type literal() :: {:num, number()} | {:var, atom()}
@type expr() ::
literal()
|   {:add, expr(), expr()}
|   {:mul, expr(), expr()}
|   {:exp, expr(), literal()}
|   {:log, expr()}
|   {:sin, expr()}
|   {:cos, expr()}
```

A literal is either a number or a variable and an expression could be anything. There now existed a way to represent the mathematical expressions. The next step was to implement the differentiation rules so it was possible to test the program.

Differentiation

The common rules of differentiating sums and products where the first to be implemented. After all the basic cases was covered the student moved on to the rest of the special cases.

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

The fact that an exponential representation was already in place helped when implementing the $ln(x)$. Instead of adding the ability to disclose division, the student used the fact that $1/x$ is the same thing as x^{-1} to represent the differentiation of $ln(x)$. This is of course a limitation of the program but the student this was sufficient to cover the case of $ln(x)$. The returning tuple was an *exp* expression which there was already a pprint function for. However, it was made sure that differentiating on ln of a number with respect to anything resulted in zero. The beauty of the exponential function was that it resolved differentiating on $\sqrt{x}$, $1/x$ and of course x^n. This only left the trigonometric functions to be implemented.

```
def deriv({:sin, e}, v) do {:mul, {:cos, e}, deriv(e, v)} end
def deriv({:cos, e}, v) do
{:mul, {:mul, {:num, -1}, {:sin, e}}, deriv(e, v)} end
```

The differentiation of $sin(x)$ is $cos(x)$. The atoms *cos* and *sin* needed to be added as well as two additional deriv/2 function that either received a $sin(x)$ expression a $cos(x)$ expression, and returned the correct differentiated expression. Differentiating on $sin(x)$ was easy since this only entailed returning a $cos(x)$ expression. Differentiating on $cos(x)$ gives $-sin(x)$ which was resolved by multiplying with -1 as seen in the code snippet above. It was now possible to take the derivative on most expressions.

In the assignment it is said to find the general rule of finding the derivative. It was not very clear what this meant but the student theorized it was the chain rule that was expected to be implemented. As can be seen from the code examples above this was implemented worked like it was intended. Here is an example of an expression that was tested with the chain rule and how it was simplified using the functions. This is of course not simplified all the way but this was as far as the author got. $Expression : (2*x+ln(3*x))$
$Derivative : ((0*x+2*1)+(0*x+3*1)*(3*x)^{-1})$
$Derivativesimplied : (2+3*(3*x)^{-1})$