Taking the Derivative: Assignment 1

Aditya Gupta

January 2023

1 Introduction

In this assignment, we implement a derivative calculator program using the Elixir programming language. We implement the following derivative rules:

- $\frac{\mathrm{d}x}{\mathrm{d}x} = 1$
- $\frac{\mathrm{d}c}{\mathrm{d}x} = 0$
- $\frac{\mathrm{d}((f+g)(x))}{\mathrm{d}x} = \frac{\mathrm{d}(f)}{\mathrm{d}x} + \frac{\mathrm{d}(g)}{\mathrm{d}x}$
- $\frac{\mathrm{d}((f*g)(x))}{\mathrm{d}x} = g\frac{\mathrm{d}(f)}{\mathrm{d}x} + f\frac{\mathrm{d}(g)}{\mathrm{d}x}$
- $\frac{\mathrm{d}(x^n)}{\mathrm{d}x} = n(x^{n-1})$
- $\frac{\mathrm{d}(x^n)}{\mathrm{d}x} = n(x^{n-1})$
- $\frac{\mathrm{d}(ln(x))}{\mathrm{d}x} = 1/x$
- $\frac{\mathrm{d}(1/x)}{\mathrm{d}x} = -1/x^2$
- $\frac{\mathrm{d}(\sqrt{x})}{\mathrm{d}x} = 1/(2\sqrt{x})$
- $\frac{\mathrm{d}\sin(x)}{\mathrm{d}x} = \cos(x)$

These rules are some of the fundamental functionality one expects to have to be able to define the derivative operation for the above functions.

2 Representing mathematical statements using expressions

As humans, it is quite easy for us to interpret mathematical statements written in the form of strings, but when we want to operate on said strings using an elixir program, parsing and interpreting can be difficult. For the purpose of this assignment, we find that tuples and atoms work very well.

```
@type literal() :: {:num, number()} | {:var, atom()}

@type expr() :: {:add, expr(), expr()} | {:mul, expr(), expr()} |
{:div, expr(), expr()} | {:exp, expr(), literal()} | {:sin, expr()}
| {:ln, expr()} | {:cos, expr()} | literal()
```

In the above arguments, we tell the type of information we are working with. If it starts with the atom `:num`, we know that the argument that follows it is a number. Similarly, we can have a tuple beginning with an operation being represented by an atom as well as can be seen above where we use atoms like `:add` to represent the add operations between two expressions. We follow the same process for the other operations and functions as well.

3 Taking the Derivative

```
def deriv({:sin, e1}, v), do: {:mul, {:cos, e1}, deriv(e1)}
def deriv({:sin, _}, _), do: {:num, 0}
```

The above code was used to define the derivative function for the sin(x) function. In mathematics we know that the derivative of a composite function is given in the following way: $\frac{\mathrm{d}(f(g(x)))}{\mathrm{d}x} = f'(g(x))\frac{\mathrm{d}(g(x))}{\mathrm{d}x}$

The above statement defines the derivative of a function using the chain rule. If we take the derivative of the $sin()$ function we know that we get $cos()$ of whatever expression the function was operating on multiplied by the derivative of the expression with respect to the variable we are taking the derivative with respect to.

We do basically the same thing in the code above where we define the derivative of the $sin()$ function based on the above equation which means for this function the equation is defined as the multiplication of $cos()$ of the expression with the derivative of the expression.

$$\frac{\mathrm{d}(sin(g(x)))}{\mathrm{d}x} = cos(g(x))\frac{\mathrm{d}(g(x))}{\mathrm{d}x}$$

So for example, if we take g(x) = 2x or we take the derivative of the function of sin(2x), we get the following:

$$\frac{\mathrm{d}(sin(2x))}{\mathrm{d}x} = cos(2x)\frac{\mathrm{d}(2x)}{\mathrm{d}x} = 2cos(2x)$$

4 Simplification

When we represent the expression used to make it easier to represent the math statements for the program to be able to operate on them, we also in turn make them less readable. Thus when we get the result we need to simplify it further to make it more readable and remove any redundant terms that get created during the process of taking the derivative of the expression. The redundant terms could be literals or sub-expressions that are multiplied with 0 or that are being added to net 0 quantities. To illustrate the unreadable aspect of an expression used to represent a simple mathematical statement, we take the following example:

$5x^2 + 3$ [MATHEMATICAL STATEMENT]

```
{:add, {:mul, {:exp, {:var, :v}, {:num, 2}} , {:num, 5}}, {:num, 3}}
```

[ELIXIR EXPRESSION]

The result after taking the derivative: $((2 * (v^1 * 1)) * 5) + (0 * v^2)$
This is a fairly simple statement which keeps it still somewhat readable. However, it still contains a redundant term where the v^2 is being multiplied by 0 which after being added should just leave us with the rest of the non-zero terms but as we can see it hasn't been eliminated.

To be able to answer what the simplest form has been defined based on we can look at the question given in the assignment comparing the 2 different notations for representing a mathematical statement:

1. x*(y+2)
2. xy + 2y

In my opinion, one can use either notation but for the purpose of this assignment, I think it would be more suitable to use the first approach.

```
def simplify_mul({:num, 1}, e2) do e2 end
def simplify_mul(e1, {:num, 1}) do e1 end
def simplify_mul({:num, 0}, _) do {:num, 0} end
def simplify_mul(_, {:num, 0}) do {:num, 0} end
```

The code above is used to further simplify expressions containing the `:mul` atom. Whenever an expression is multiplied by zero it can be directly simplified to 0 and similarly any expression multiplied by 1 would result in the same expression. The other cases for simplification as well as the different possible cases for other operations can be found in my GitHub repository linked in the Code section.