Derivatives

Simon Yllmark

January 2024

Introduction

In this assignment I created a program in Elixir that takes mathematical expressions and calculate the derivative of them. The essential tool here is pattern recognition or pattern matching, because what the program those is that it takes a mathematical expression and divide it into smaller patterns where in the program there is defined rules for how to derive the derivative from these patterns. One essential piece of information is that these smaller patterns should be independent from each other.

Representing functions

In Elixir it is possible to write functions using syntax for example in the VS code terminal. That means that we do not have to write a program in Elixir that contains the functions that we would want to use. One negative aspect of that is that we can not view the body of the function which is kinda important if we would like to take the derivative of functions. That is the main reason to why I programmed functions in Elixir because then details like how the functions are defined or the body of the function is specified.

An expressions

When representing mathematical expressions in Elixir we need to think about what datatype that should represent them. One important thing to think about in this assignment is to group the constants and variables into two separate groups due to derivative rules such as:

- $\frac{d}{dx}x = 1$
- $\frac{d}{dx}c = 0$

That is why in my code I represent the different functions as two different tuples where one is defined to hold an integer or a float and the other is defined to hold another atom, such as:

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

Expressions

When writing the different mathematical expressions in Elixir I used tuples because they work when one would want to list a line of data, here is one example from my code:

```
e = {:add, {:mul, {:num, 2}, {:var, :x}}, {:num, 4}}
```

This example also contains atoms used to represent functions such as addition and multiplication and is supposed to represent the mathematical expression $2 * x + 4$. The different functions in my Elixir program are defined as:

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

Then we have the case where the different expressions expresses themselves. It becomes the case where the function defines itself while calling itself recursively. For example the expression:

```
@type expr() :: {:add, expr(), expr()}
```

Defines the addition operation as containing two other expressions.

The derivative of

When then taking the derivative of for example the mathematical expression $2 * x + 4$ we can then separate the expression into two smaller expression like $2 * x$ and 4, then we can derive these expression using these derivative rules:

- $\frac{d}{dx}x = 1$
- $\frac{d}{dx}c = 0$
- $\frac{d}{dx}(f + g) = \frac{d}{dx}f + \frac{d}{dx}g$

- $\frac{d}{dx}(f * g) = \frac{d}{dx}f * g + \frac{d}{dx}g * f$
- $\frac{d}{dx}f(g(x)) =' f(g(x)) * \frac{d}{dx}g(x)$

My code that implement these derivative rules will then produce the mathematical expression $(((0 * x) + (2 * 1)) + 0)$ which is equivalent to 2. The derivative of $2 * x + 4$ is 2, which is correct. Thus, I implemented a function that takes the derivative of an expression and another one that simplify them. That makes my life easier when I have to verify that the derivative function works.

Carrying on

Then carrying on I made functions that take into consideration derivatives of other function as the following:

1. x^n
2. $\ln x$
3. $\frac{1}{x} = x^{-1}$
4. $\sqrt{x} = x^{1/2}$
5. $\sin x$

By applying math to the different expressions I could then deduce that 3 out of 5 expressions could be written as x^n where n is a real number. This then lead to that I had only to consider the derivative of 3 cases instead of 5.

Simplification

Just taking the derivative of a expression is enough to solve the main task of this assignment but verifying that it works is something that could be made simpler by shortening the length of the produced mathematical expression. This makes my life easier when it comes to debugging output that the derivative function produces. For example the mathematical expression $(((0 * x) + (2 * 1)) + 0)$ would be simplified by my program to 2.

Simplest form

Then comes the discussion of what is the simplest form because it's part of the assignment. According to my program the simplest form is the form where we take away redundancies, for example the expression $2 + (x * 0)$ would be simplified to 2 by my program. In that case the program then

takes away all the $+0$ or $0 * x$ or $0 * n$. My program can also replace the variable x with a value and then do mathematical operation in order to further simplify mathematical expression.

In this assignment we where supposed to chose which of the two forms that was simplest $x(y + 2)$ or $xy + 2x$ and I would say the latter option because it allows the program to separate the mathematical expression into two independent expressions. Both would still work but there would be an ordered operation on the mathematical operation $x(y + 2)$ where we would have to calculate $y + 2$ first and then $x * (y + 2)$. The other option let's the program calculating xy, $2x$ or both at the same time in concurrency or parallel.