Task 2 - Derivatives

Casper Kristiansson

2022-01-25

Introduction

The second exercise of the course Programming II consisted of creating a program which can derivative several types of functions. The author will be discussing how the problem was solved, how the different types of operations (addition, multiplication...) was develop, how the result is simplified and how the result is calculated.

Method

The author firstly started off by watching the three different lectures on recursion, trees, and derivatives. The recursion lecture covered the basics of *pattern matching* and some simple elixir methods such as appending to a list and union between two lists. The tree lecture included the basics of tuples and a few of its distinct functions (insert, duplicate, delete). The last lecture, derivatives, were a walkthrough of the exercise and how it could be solved. It covered a couple off basic functions and how they could be derivative, simplified and calculated. These three lectures gave sufficient knowledge to solve the task.

The several types of mathematics expressions that the program should be able to manage is addition, multiplication, exponential, natural log, division, square root, sin, and cos. The program should also follow the different type of derivation rules ($\frac{d}{dx}(x) = 1$, $\frac{d}{dx}(c) = 0$, $\frac{d}{dx}(f) + g = \frac{d}{dx}(f) + \frac{d}{dx}(g)$, $\frac{d}{dx}(f) * g = \frac{d}{dx}(f) * g + f * \frac{d}{dx}(g)$.) The author decided to solve the task using Abstract Syntax Trees (AST) to represent the different operations, variables, and numbers.

Result

1. Figure 1 consists of the division **test function**. Each test function creates an expression with the desired mathematics expression. Afterwards the derivative is calculated using the **deriv** method and then its value for

a specific x is calculated using **calc**. After all calculations have been made the author writes to the console the start expression, the derivative of that expression, the simplified expression, and the value for a specific x.

```
def testdiv() do
   e = {:div, {:num, 3}, {:var, :x}}
   d = deriv(e, :x)
   c = calc(d, :x, 2)
   IO.write("Expression: #{pprint(e)}\n")
   IO.write("Derivative: #{pprint(d)}\n")
   IO.write("Simplified: #{pprint(simplify(d))}\n")
   IO.write("Calculated: #{pprint(simplify(c))}\n")
 end
```

Figure 1: Test function for the method division

Derivation Functions

The author will be discussing the several types of functions for most of the operators. Some functions will not be mentioned if they are like others that already have been stated. Because the most basic operations were shown in the lecture, the author will not be discussing them.

The **division** derivation function starts of by calculating the multiplication between the negative top number and the derivation the bottom number/expression. The next step is to calculate the exponential of the denominator number with two. Combining those two calculations results in the derivation of the start expression.

```
def deriv({:div, {:num, n}, e}, v) do
   {:div, {:mul, {:num, -n}, deriv(e, v)}, {:exp, e, {:num, 2}}}
end
```

Figure 2: Division function

The **natural log** function starts of by calculating the nominator which is the derivative of the expression and the denominator is the expression.

```
def deriv({:ln, e}, v) do
   {:div, deriv(e, v), e}
end
```

Figure 3: Natural Log function

The **square root** function starts of by calculating the derivative of the expression for the nominator. The next step is to calculate multiplication between two the original expression for the denominator. The square root method is not able to compute advanced operations with square root, it can manage $\sqrt{x}$ and $\sqrt{5x}$...

```
def deriv({:sqrt, e}, v) do
   {:div, deriv(e, v), {:mul, {:num, 2}, {:sqrt, e}}}
end
```

Figure 4: Square root function

The **sin** function is calculated by multiplying the derivation of the variables inside the sin function and multiply them with original function.

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

Figure 5: Sin function

Simplifying expressions

Depending on the expression, most of the time it could be easily simplified. The expression could be simplified by creating different definition for what the function should return in a specific case. For example, if the goal is to multiply 1 with x, it could be simplified to just the variable x.

Figure 6 consists of four different cases. The first situation is the standard one where two different numbers divide each other, which can simply be calculated. The next situation is when the nominator is divided by one, which means that it equals the nominator. The last two situations cannot be simplified further and therefore the result is just returned.

```
def simplify_div({:num, n1}, {:num, n2}) do {:num, n1 / n2} end
def simplify_div({:num, 1}, e2) do {:div, {:num, 1}, e2} end
def simplify_div(e1, {:num, 1}) do e1 end
def simplify_div(e1, e2) do {:div, e1, e2} end
```

Figure 6: Simplify function for division

Calculating With Specific X Values

The next step of the task is to calculate the value of a derived expression. The **calc** function starts calculating a value from a specific X value by mapping out and filling in the different X values. Afterwards the result is entered to the simplify method which will return a single digit which is the deviated value for that X point. In figure 7 shows the different calc functions used to calculate division.

```
def calc({:num, n}, _, _) do {:num, n} end
def calc({:var, v}, v, n) do {:num, n} end
def calc({:var, v}, _, _) do {:num, v} end
def calc({:div, e1, e2}, v, n) do {:div, calc(e1, v, n), calc(e2, v, n)} end
```

Figure 7: Calculating a specific X value with the help of calc

Printing the equation

To show the expression in a more readable way the author develops the function **pprint**. The function is a recursive function which will continuously print the different variables depending on the operation. Figure 8 consists of the function for printing division.

```
def pprint({:div, e1, e2}) do "(#{pprint(e1)} / #{pprint(e2)})" end
```

Figure 8: Calculating a specific X value with the help of calc

Combining

After all of the functions have been develop and running the test from figure 1 produces the result which can be seen in figure 9.

```
Expression: sqrt((5 * x)) #x = 3
Derivative: (((0 * x) + (5 * 1)) / (2 * sqrt((5 * x))))
Simplified: (5 / (2 * sqrt((5 * x))))
Calculated: 0.6454972243679028
```

Figure 9: Sample Printout