Lab 1 - Derivative

Weichen Chai

Spring 2023

Introduction

This paper seeks to answer the questions presented in Taking the Derivative for ID1019. In this assignment, we will explore how the functional language Elixir can be utilized in a function, as well as using this language to find the derivative of various numerical procedures.

Tasks

The assignment consists of seven sections, the first three of which are explanations of different functions of Elixir.

Defining and Testing

From task 1 to task 5, we were given some instructions to implement methods of implementing derivatives. Something left incomplete in the assignment is defining the derivative forms for different conditions, which can be seen completed below:

```
def derive({:num, _}, _) do {:num, 0} end // Derivative of num = 0
def derive({:var, v}, v) do {:num, 1} end // Derivative of variable = 1
def derive({:var, _}, _) do {:num, 0} end
  // see slides for multiplication and addition equation.
def derive({:add, e1, e2}, v) do {:add, derive(e1, v), derive(e2, v)}
end
def derive({:mul, e1, e2}, v) do
{:add, {:mul, derive(e1, v), e2}, {:mul, e1, derive(e2, v)}}
end
```

Through using this implementation, we can attempt to run a test on the equation presented in part 3 of the assignment: 2x+3, if run correctly, it should return the number 2. When we run the program, it becomes returns in the terminal the following:

```
{:add, {:add, {:mul, {:num, 0}, {:var, :x}}, {:mul, {:num, 2}, {:num, 1}}}
```

This can be read as 0+2, which is 2 and this is the correct answer for that particular equation. However, it is quite unreadable. To make the answer more comprehensible to regular people, we can attempt to simplify the equation.

Expanding on more derivatives

Section 6 asks for the addition of exponents, logarithms and various other numerical procedures we should take the derivative of. They can be implemented by following the steps shown above for addition and multiplication, where we first implement the derivative of the procedure. An example can be logarithm, where:

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

Then we write the simplification for that function to make sure that the result is readable. After testing the new expressions, I have confirmed that it works as intended.

Simplification

Taking addition as an example, when adding two elements, we can simplify them with the following:

```
  def simplify({:add, e1, e2}) do simplify_add(simplify(e1), simplify(e2))
  end
  ...
  def simplify_add(e1, {:num, 0}) do e1 end
  def simplify_add({:num, 0}, e2) do e2 end
  def simplify_add({:num, n1}, {:num, n2}) do {:num, n1 + n2} end
  def simplify_add(e1, e2) do {:add, e1, e2} end
```

With the method above, we have listed all the possible scenarios of calculation with addition, and gave a designated simplification for each. In the end, we end up with a simple answer of the equation used before after simplifying both addition and multiplication, which is seen below:

```
{:num, 2}
```

Of course, this became a lot more readable, however, with lengthier answers, it will still be quite complicated, if we want to further simplify the output results, we can try to display the simplified answer as a string. This can be done by simply writing a function pprint() for multiplication and addition :

```
def pprint({:add, e1, e2}) do "(#{pprint(e1)} + #{pprint(e2)})" end
def pprint({:mul, e1, e2}) do "#{pprint(e1)} * #{pprint(e2)}" end
```

this will return string values proposed by the user. when testing with for example: 2(x*x), it returns

```
"2 * (x + x)"
```

Discussion and Conclusion

In part 7 of the assignment, we were asked to discuss about how the simplest form should look like

$$x(y + 2)$$

as this is the form that is does not have any repetitive expressions.

In conclusion, In this assignment, I have gone through basic functions of elixir and gotten used to working with it in a efficient manner, as well as implementing simple derivative solvers using those basic functions. I also got an insight as to returning string products in elixir.