# Task 1 Derivatives

Adrian Jonsson Sjödin

Spring Term 2023

## Introduction

To get started with this task, we first watched all the uploaded videos about how to code derivatives in Elixir. This gave an understanding on how one should represent numbers and variables in this program, and how to use this representation. In this report we are going to assume these videos have been watched and thus not repeat the information given there.

## Method

The first four derivatives $\frac{d}{dx}x$, $\frac{d}{dx}c,\ c \in \mathbb{R}$, $\frac{d}{dx}(f+g)$ and $\frac{d}{dx}(f \cdot g)$ are implemented according to the mentioned uploaded videos.

Using the knowledge gained from implementing them, we implemented six other derivative rules.

1. $\frac{d}{dx}(u(x)^n) = n \cdot u(x)^{n-1}u'(x),\ n \in \mathbb{R}$
2. $\frac{d}{dx}(\frac{k}{u(x)^n}) = \frac{-nk \cdot u'(x)}{u(x)^{n+1}},\ n, k \in \mathbb{R}$
3. $\frac{d}{dx}(ln(u(x))) = \frac{u'(x)}{u(x)}$
4. $\frac{d}{dx}(\sqrt{u(x)}) = \frac{u'(x)}{2\sqrt{u(x)}}$
5. $\frac{d}{dx}(sin(u(x))) = u'(x) \cdot cos(u(x))$
6. $\frac{d}{dx}(cos(u(x))) = -u'(x) \cdot sin(u(x))$

The implementation of these where straight forward. We only had to follow the same pattern as in the previous implemented derivatives. For example the 1st derivative it is simply

```
def deriv({:exp, u, {:num, n}}, v) do
{:mul,
    {:mul, {:num, n}, {:exp, u, {:num, n - 1}}},
```

```
        deriv(u, v)
    }
  end
```

As we can see to write the functions we simply need to use the formatting above and write the corresponding expression found after the equal sign shown above.

We also coded some simplifier functions whose task was to simplify the expression, for example by removing expressions that where multiplied with zero or where the dividend is zero.

Lastly we needed a way to convert from our representation of an expression to something more readable. For this we coded a couple of functions called `p_print/1` and `p_print/2`.

## Result

Below are the result from derivating different one variable functions.

```
iex> Deriv.test_add()
Expression: (x + 5)
Derivative of expression: (1 + 0)
Simplified: 1
:ok
iex> Deriv.test_mul()
Expression: (x + 5*(x)^(4))
Derivative: (1 + (0*(x)^(4) + 5*4*(x)^(3)*1))
Simplified: (1 + 5*4*(x)^(3))
:ok
iex> Deriv.test_exp2()
Expression: ((2*x + 3))^(2)
Derivative of expression: 2*((2*x + 3))^(1)*((0*x + 2*1) + 0)
Simplified: 2*(2*x + 3)*2
:ok
iex> Deriv.test_div()
Expression: (4/((x*3 + 2))^(2))
Derivative of expression: (4*-2*((1*3 + x*0) + 0)/((x*3 + 2))^(3))
Simplified: (-24/((x*3 + 2))^(3))
:ok
iex> Deriv.test_ln1()
Expression: ln((x)^(2))
Derivative of expression: (2*(x)^(1)*1/(x)^(2))
Simplified: (2*x/(x)^(2))
:ok
iex> Deriv.test_ln()
```

```
Expression: 2*ln(((2*x + 3))^(2))
Derivative of expression: (0*ln(((2*x + 3))^(2)) +
2*(2*((2*x + 3))^(1)*((0*x + 2*1) + 0)/((2*x + 3))^(2)))
Simplified: 2*(2*(2*x + 3)*2/((2*x + 3))^(2))
:ok
iex(72)> Deriv.test_sqrt()
Expression: 2*sqrt((3*(x)^(2) + x))
Derivative of expression: (0*sqrt((3*(x)^(2) + x)) +
2*(((0*(x)^(2) + 3*2*(x)^(1)*1) + 1)/2*sqrt((3*(x)^(2) + x))))
Simplified: 2*((3*2*x + 1)/2*sqrt((3*(x)^(2) + x)))
:ok
iex> Deriv.test_sin()
Expression: 2*sin((3*(x)^(2) + 4*x))
Derivative of expression: (0*sin((3*(x)^(2) + 4*x)) +
2*((0*(x)^(2) + 3*2*(x)^(1)*1) +
(0*x + 4*1))*cos((3*(x)^(2) + 4*x)))
Simplified: 2*(3*2*x + 4)*cos((3*(x)^(2) + 4*x))
:ok
iex> Deriv.test_cos()
Expression: 2*cos((3*(x)^(2) + 4*x))
Derivative of expression: (0*cos((3*(x)^(2) + 4*x)) +
2*-1*((0*(x)^(2) + 3*2*(x)^(1)*1) +
(0*x + 4*1))*sin((3*(x)^(2) + 4*x)))
Simplified: 2*-1*(3*2*x + 4)*sin((3*(x)^(2) + 4*x))
:ok
```

## Discussion

The result show in the section is correct and matches those given by WolframAlpha. We can thus conclude that our program works as intended and can derive all the functions specified in the task description. However the result is not in the most user friendly form. Unfortunately this was something I couldn't fix with my limited knowledge of Elixir, and thus after spending a couple of hours on it I was forced to accept defeat. One workaround just to confirm my results is to simply copypaste the result into WolframAlpha and let it simplify it for you.

All the code can be found on my GitHub.