Haskell: Derivada de um ponto

Alunos: Anabelle Elizabeth Araujo de Souza e Jules Severo Barcos

Prof. Me. Alexandre Garcia

Santos, 16 de Março de 2023

CONTROLE DE VERSÃO			
Autor	**Versão**	**Data**	**Descrição**
Anabelle Souza	1.0	14/03/2023	Criação do documento
Jules Severo	2.0	15/03/2023	Criação do documento

Sumário

1 Definição de Func

Estruturas em Haskell:

A definição da estrutura "Func": essa é a definição da nossa estrutura de dados para expressões matemáticas. Cada construtor dessa estrutura corresponde a uma operação que pode ser feita com funções matemáticas. Por exemplo, X é o construtor para uma função que retorna x, Sin é o construtor para uma função seno, Power é o construtor para uma função potência, e assim por diante.Veja nossa aplicação abaixo:

```
data Func = X -- f(x) = x
          | Sin Func -- f(x) = sen(g(x))
          | Cos Func -- f(x) = cos(g(x))
          | Exp Func -- f(x) = e^g(x)
          | Ln Func  -- f(x) = ln(g(x))
          | Power Func Double -- f(x) = g(x)^n
          | Const Double -- f(x) = k
          | Sum Func Func -- h(x)= f(x) + g(x)
          | Mult Func Func -- h(x) = f(x) * g(x)
          | Div  Func Func -- h(x) = (f(x) / g(x))
          deriving Show
```

Figura 1: Definição de "Func".

2 Instância Read

A instância Read que atua com a estrutura Func: essa é uma instância necessária para que possamos ler expressões matemáticas diretamente do console, como fizemos nos testes. Ela basicamente permite que possamos ler um valor de tipo String e convertê-lo para um valor de tipo Func, usando a função read.

```
instance Read Func where
  readsPrec _ input = [(Const (read input :: Double), "")]
```

Figura 2: Instância Read.

3 Definição de DDX e Eval

DDX é a função que calcula a derivada de uma expressão matemática em um ponto específico. Ela usa as regras de derivação para calcular a derivada de cada operação possível, e usa recursão para calcular a derivada de expressões mais complexas. Por exemplo, para calcular a derivada da soma de duas funções f e g, ela simplesmente calcula a derivada de f e de g e as soma.

A definição da função eval: essa é a função que avalia uma expressão matemática em um ponto específico. Ela usa recursão para avaliar expressões mais complexas, como a soma de duas funções f e g.

```
eval :: Func -> Double -> Double
eval X x = x
eval (Sin f) x = sin (eval f x)
eval (Cos f) x = cos (eval f x)
eval (Exp f) x = exp (eval f x)
eval (Ln f) x = log (eval f x)
eval (Power f n) x = eval f x ** n
eval (Const k) _ = k
eval (Sum f g) x = eval f x + eval g x
eval (Mult f g) x = eval f x * eval g x
eval (Div f g) x = eval f x / eval g x

main :: IO ()
main = putStrLn "Olá, usuário! Vamos calcular?"
```

Figura 3: DDX e Eval.

Lembrando que algumas definições são:

- A definição da regra para X: essa é a regra mais simples de todas. A derivada de x é 1, e é isso que essa definição faz.
- As regras para Sin, Cos, Exp e Ln: essas são as definições das regras de derivação para funções trigonométricas, exponenciais e logarítmicas. Elas usam as fórmulas de derivação correspondentes para calcular a derivada da função em um ponto específico.
- A definição da regra para Power: essa é a definição da regra de derivação para funções potência. Ela usa a regra da cadeia para calcular a derivada da função em um ponto específico.
- Para Sum, Mult e Div: essas são as definições das regras de derivação para as operações de soma, multiplicação e divisão de funções. Elas usam as fórmulas correspondentes para calcular a derivada da função em um ponto específico.
- A definição da regra para Const: essa é a regra mais simples de todas. A derivada de uma constante é 0, e é isso que essa definição faz.

4 Exemplos Práticos - Testes no Console

Para realizar os testes, é necessário digitar a expressão e depois dos parênteses o ponto X que deseja que seja calculado. Veja os exemplos abaixo:

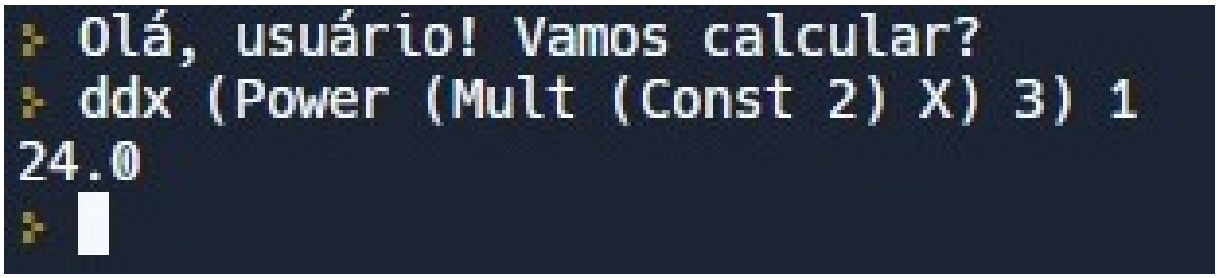

Figura 4: x = 1

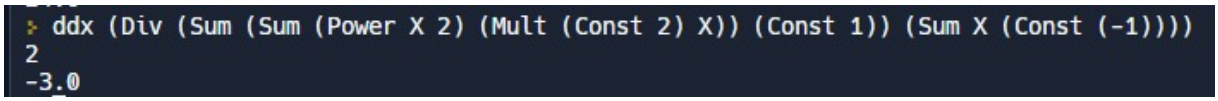

Figura 5: x = 2

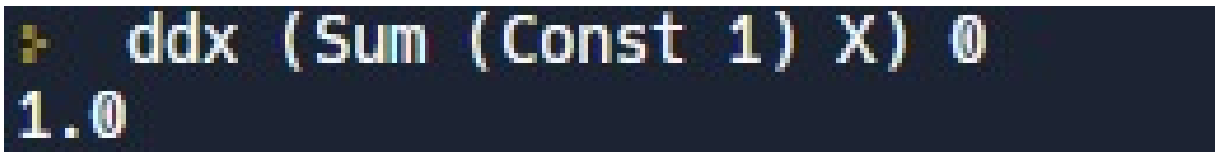

Figura 6: x = 0

Figura 7: x = 5

Figura 8: x = 0

5 Github

Link do repositório no Github: AnabelleSouza/Haskell