CMSC22300 Final Project: Forward-Mode Automatic Differentiation

Jay Shen

1 Introduction

This project implements a forward-mode automatic differentiation (AD) library in Haskell using dual numbers. Unlike symbolic differentiation, which manipulates expressions, or numerical differentiation, which approximates derivatives via finite differences, AD computes exact derivatives by augmenting each value with its derivative and propagating both through arithmetic operations via operator overloading. The library supports higher-order derivatives through function composition and gradients of multivariable functions. We demonstrate its correctness and utility through regressions and visualizations.

2 Implementation

2.1 Dual Numbers

The core abstraction is the dual number type:

```
data Dual a = Dual { primal :: !a, tangent :: !a }
```

A dual number (x, x') carries a primal value x and a tangent x'. We hard code a complete set of arithmetic operations in the type class. For example, we define multiplication

$$(x, x') \cdot (y, y') = (xy,\ x'y + xy')$$

corresponding to the product rule. We provide `Num`, `Fractional`, and `Floating` instances that encode the chain rule for every elementary operation (addition, multiplication, division, `sin`, `cos`, `exp`, `log`, `sqrt`, etc.).

2.2 Differentiation Combinators

The key combinators are:

- `var x = Dual x 1` instantiates a differentiable variable with derivative 1.
- `diff f = \x -> tangent (f (var x))` extracts the derivative of a univariate function as a first-class function.

- `grad f xs` computes the gradient ∇f by compiling a list of partial derivatives

Because `diff` returns an ordinary function, higher-order derivatives are obtained by composition: `(diff . diff) f` is f'', `(diff . diff . diff) f` is f''', and so on. This works because the user's function is polymorphic over `Num`/`Floating`, allowing nested layers of `Dual`.

3 Regression Examples

Both regression examples use `grad` to compute gradients of a loss function, which are then used for gradient descent optimization. The training loop is model-agnostic: it takes any polymorphic loss function `[a] -> a` and updates parameters by $\theta \leftarrow \theta - \eta \nabla L(\theta)$.

3.1 Linear Regression

We generate $n = 50$ noisy samples from $y = mx + b$ on $[-4, 4]$ with Gaussian noise ($\sigma = 0.5$) and minimize mean squared error:

$$L(m, b) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - (mx_i + b)\big)^2$$

Starting from $(m_0, b_0) = (0, 0)$ with learning rate $\eta = 0.001$ over 1000 steps, gradient descent recovers parameters close to the true values (Table 1).

Parameter	True	Fitted	Final Loss
m	3.0	3.029	0.212
b	1.0	0.874	

Table 1: Linear regression results after 1000 steps ($\eta = 0.001$).

3.2 Logistic Regression

We generate 100 binary classification samples from two Gaussians ($\mu_0 = -2$, $\mu_1 = 2$, $\sigma = 1$) and fit a logistic model $P(y = 1 \mid x) = \sigma(wx + b)$ by minimizing binary cross-entropy:

$$L(w, b) = -\frac{1}{n} \sum_{i=1}^{n} \big[y_i \log p_i + (1 - y_i) \log(1 - p_i)\big]$$

where $p_i = \sigma(wx_i + b)$. With learning rate $\eta = 0.1$ over 1000 steps, the model learns a decision boundary and achieves high classification accuracy (Table 2).

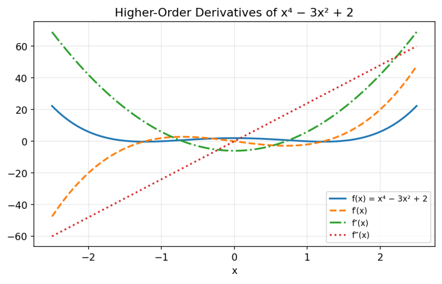
Parameter	Fitted	Decision Boundary	Final Loss	Accuracy
w	3.360	-0.032	0.032	99%
b	0.106			

Table 2: Logistic regression results after 1000 steps ($\eta = 0.1$).

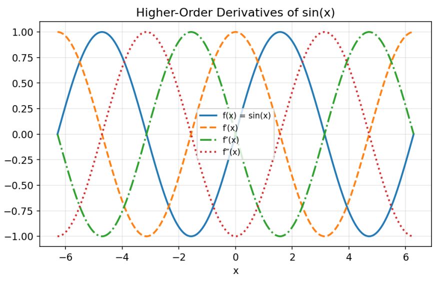
4 Visualizations

4.1 Higher-Order Derivatives

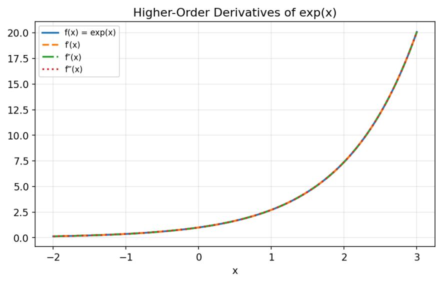
We evaluate four test functions and their first three derivatives (computed via iterated `diff`) over a range of x values. Figure 1 shows that the AD-computed derivatives match the expected analytic forms.



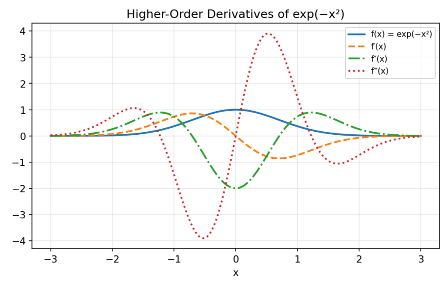
(a) $f(x) = x^4 - 3x^2 + 2$



(b) $f(x) = \sin(x)$



(c) $f(x) = e^x$



(d) $f(x) = e^{-x^2}$

Figure 1: Four test functions and their first three derivatives, computed via iterated `diff`.

4.2 Gradient Field

We compute the gradient of $f(x, y) = \sin(x)\cos(y)$ over a 50×50 grid on $[-\pi, \pi]^2$ using `grad`. Figure 2 shows the 3D surface alongside a contour plot with overlaid gradient vectors, confirming that the gradient field points in the direction of steepest ascent.

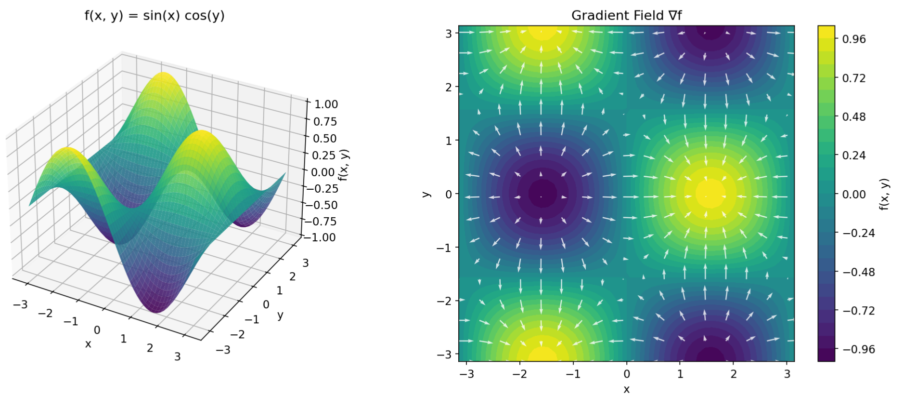


Figure 2: Surface and gradient field of $f(x, y) = \sin(x)\cos(y)$.

5 Conclusion

This project demonstrates that forward-mode AD via dual numbers provides a clean, composable approach to exact differentiation in a purely functional setting. Haskell's typeclass system makes the implementation concise: overloading arithmetic operators on `Dual` numbers is sufficient to differentiate any polymorphic function. The regression examples show that the computed gradients are accurate enough to drive gradient descent to convergence, and the visualizations confirm correctness of higher-order derivatives and multivariable gradients.