Chapter 5 + ϵ

Derivatives for Scientific Computing

Outline

- Summary of different types of differentiation
  - Symbolic differentiation
  - Numerical differentiation, a.k.a. finite-differences
  - Automatic differentiation (AD)
- Introduction to forward-mode AD with dual numbers
- Exercises in MATLAB[1]

Introduction

The derivative is a ubiquitous tool in computation, optimization, and data science. This lesson is designed to introduce you to the concept of automatic differentiation, which is an algorithmic technique for calculating the derivatives of functions. The lesson will begin with an overview of symbolic differentiation and finite differences before introducing automatic differentiation. The lesson will then cover the use of dual numbers for automatic differentiation and will explore other forward-mode and reverse-mode methods.

Exercise 0.1: First exercise

Draw a word cloud of concepts related to "differentiation." Hint: Try to list some definitions, notations, methods, uses, etc. . .

[1] `https://github.com/williamjsdavis/SIO112-Derivatives`

ϵ.1 Types of Differentiation

Differentiation is a fundamental operation in calculus that is used to find the rate at which a function changes. The derivative of a function at a particular point is defined as the slope of the tangent line to the curve at that point. In mathematical notation, the derivative of a function $f(x)$ at a point x is denoted as $f'(x)$.

ϵ.1.1 Symbolic Differentiation

Symbolic differentiation is a technique used to find the derivative of a function using algebraic manipulation. This method involves applying a set of predefined rules to the algebraic expression of a function to obtain its derivative. This can be done by a human following rules you learnt in calculus classes (often called "manual differentiation"), or by a computer algebra system (CAS).

Exercise 1.1: Symbolic methods

What are some differentiation methods or rules you've heard of, or used? Have you used any computer algebra system? Add these to your word cloud.

While symbolic differentiation is exact, it can't handle any function, and can become computationally expensive for complicated functions.

Exercise 1.2: Symbolic examples

Symbolically differentiate (by hand, or use a CAS[a]) the following functions:

$$f_1(x) = 4x^3 + 2x - 100$$

$$f_2(x) = \sin(x)$$

$$f_3(x) = \text{abs}(x)$$

$$f_4(x) = s_4, \text{where } s_1 = x, \text{and } s_{n+1} = 4s_n(1 - s_n)$$

[a] E.g., use `https://www.wolframalpha.com/`

What did you find? What kind of "object" does symbolic differentiation return? Take note of this.

ϵ.1.2 Numerical Differentiation

Numerical Differentiation—commonly known as "finite differences"—is a numerical method used to *approximate* the derivative of a function by calculating the difference between

two function values. This method involves computing the function at two points that are close to each other, and then dividing the difference in the function values by the difference in the points.

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{ϵ.1}$$

While finite differences is easy to implement, it can be less accurate than symbolic differentiation, and the choice of step size can significantly affect the results.

Numerical Differentiation in `MATLAB`

Let's do some examples in `MATLAB`. If you want hints, look at the file on github[2]: `section1.m`.

Exercise 1.3: Finite difference example

Use finite differences in `MATLAB` to evaluate the derivatives of

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

in the range $x \in [0, 1]$.

Make a plot of the true derivative and overlay the finite difference output. Try changing Δx and see what happens.

Finite differences are often used in solving partial differential equations. Why do you think this is? What kind of "object" does numerical differentiation return? Take note of this.

Exercise 1.4: Finite difference errors

Repeat exercise 1.3 but only evaluate the derivative of f at a single point, say $x^* = 0.55$. Calculate the error

$$E(\Delta x) = \left| f'(x^*) - \tilde{f}'(x^*, \Delta x) \right|$$

where $\tilde{f}'(x^*, \Delta x)$ is your finite difference at x^* using step size Δx. Repeat this calculation for a range of Δx values, and make a plot of $E(\Delta x)$.

What did you find? Is this what you expected? Why do you think this is happening?

[2] `https://github.com/williamjsdavis/SIO112-Derivatives`

ϵ.1.3 Symbolic vs. Numerical Differentiation

Symbolic differentiation is exact but it can't handle every function. Manually computing derivatives by hand can be time consuming and prone to error, and CAS methods can become computationally expensive for complex functions. Finite differences is easy to implement but is an approximation; the choice of step size can significantly affect the results. Perhaps there is another method for differentiation?

ϵ.2 Introduction to Automatic Differentiation

Automatic differentiation is a set of numerical methods used to compute derivatives through accumulation of values during code execution to generate numerical derivative evaluations rather than derivative expressions. It is also known as "algorithmic differentiation" or "autodiff." Automatic differentiation is more accurate than finite differences and usually faster than symbolic differentiation. Automatic differentiation can be performed using forward-mode or reverse-mode. In forward-mode, the derivative is computed by propagating the function evaluation and derivative information from the input to the output, while in reverse-mode, the derivative is computed by propagating the derivative information from the output to the input. Today, we will make a first foray into forward-mode automatic differentiation by using a technique called "dual numbers."

ϵ.2.1 An Analogous Introduction: Complex Numbers

Recall complex numbers, which mathematical extension of the real numbers. A complex number z is written in the form

$$z = a + bi,$$

where a and b are real numbers, and i is the imaginary unit, which satisfies the equation $i^2 = -1$. In common terminology, a is the "real" part, and b is the "imaginary" part. These are indicated with the functions $\mathrm{Re}(\circ)$ and $\mathrm{Im}(\circ)$,

$$\begin{aligned}\mathrm{Re}(a + bi) &= a,\\ \mathrm{Im}(a + bi) &= b.\end{aligned}$$

The important part here is that complex numbers are still numbers, they just have different rules (and therefore applications) than real numbers.

Exercise 2.1: In your own time: Properties of complex numbers

Verify that complex numbers form a mathematical structure called a "field." I.e., show that the following properties hold true:

- Associativity of addition and multiplication,

- Commutativity of addition and multiplication,
- There exist distinct additive and multiplicative identities,
- There exists an additive inverse for every element,
- There exists a multiplicative inverse for every element (except for the additive identity).

ϵ.2.2 Introducing Dual Numbers

Dual numbers are a different mathematical extension of the real numbers.[3] Like complex numbers, they have different rules than real numbers. These different rules will allow us to use them to calculate derivatives.

Dual numbers consist of a real part and a dual part. A dual number z can be represented as

$$z = a + b\epsilon,$$

where a and b are real numbers, and ϵ is the "nilpotent" unit, which satisfies the rules $\epsilon^2 = 0$ and $\epsilon \neq 0$. In common terminology, a is the "real" or "standard" part, and b is the "infinitesimal" part. These are indicated with the functions $\mathrm{St}(\circ)$ and $\mathrm{In}(\circ)$,

$$\mathrm{St}(a + b\epsilon) = a,$$
$$\mathrm{In}(a + b\epsilon) = b.$$

Exercise 2.2: In your own time: Properties of dual numbers

Investigate what rules in Exercise 2.2 hold true for dual numbers, and which rules are not held.

Dual numbers have some weird properties, which turn out to be useful for differentiation. Let's see what happens when we put dual numbers into the function

$$f(z) = z^2.$$

Try evaluating the function at

$$z = 1 + \epsilon,$$
$$z = 4 + \epsilon,$$
$$z = 10 + \epsilon,$$

[3] They have a weird history, which is intertwined with nonstandard analysis, the creation of calculus, and "hyperreal" numbers, `https://en.wikipedia.org/wiki/Hyperreal_number`.

and look at the standard and infinitesimal parts.

Let's try something more complicated,

$$f(z) = z^2 + 2z + 1,$$

evaluated at $z = 2+\epsilon$. The real part of the result corresponds to the value of the function at $x = 2$, while the infinitesimal part corresponds to the derivative of the function at $x = 2$, which is 6. The reason for this behaviour can be identified by looking at a Taylor series of an arbitrary function f,

$$f(a + b\epsilon) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)b^n\epsilon^n}{n!} = f(a) + bf'(a)\epsilon.$$

By computing functions over the dual numbers and examining the coefficient of ϵ in the result, we can "automatically" compute the derivative of the function. **Note that we only get the derivative at the evaluation point!** Working with symbols on pen and paper is all fun, but we can implement this in code too!

ϵ.2.3 Automatic differentiation in MATLAB

Unfortunately, MATLAB isn't the most flexible language for implementing dual numbers.[4] Instead, we can represent dual numbers using 2×2 matrices,[5]

$$a + b\epsilon \equiv \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}.$$

```
function dual = Dual(a,b)
%Constructs a dual number, z = a + be
dual = [a,b;0,a];
end
```

Text

```
function real_part = st(z)
%Gets the standard, or real, part of a dual number, z = a + be -> a
real_part = z(1,1);
end
```

text

```
function infinitesimal_part = in(z)
%Gets the infinitesimal part of a dual number, z = a + be -> b
infinitesimal_part = z(1,2);
end
```

Let's do some examples. If you want hints, look at the file on github[6]: `section1.m`.

[4] Typically you need to be able to define custom classes/structs, and write rules for operator overloading. MATLAB can do some of this, but it's messy...

[5] In your own time, think about why this is possible.

[6] https://github.com/williamjsdavis/SIO112-Derivatives

Exercise 2.3: First dual number experiment

Use your `MATLAB` implementation of dual numbers to calculate the derivative of $f(x) = x^2$ at $x = 5$. Remember, you'll have to evaluate the function with the dual number $z = 5 + \epsilon$, and then extract the infinitesimal part. Repeat this calculation at other evaluation points.

Exercise 2.4: Another dual number experiment

Repeat Exercise 2.3 for function $f_1(x) = 4x^3 + 2x - 100$ in Exercise 1.2. Use your `MATLAB` implementation of dual numbers to calculate the derivative at $x = 1$.

Exercise 2.5: Another dual number experiment

Repeat Exercise 2.3 for function

$$f_4(x) = s_4, \text{where } s_1 = x, \text{and } s_{n+1} = 4s_n(1 - s_n)$$

in Exercise 1.2. Use your `MATLAB` implementation of dual numbers to calculate the derivative at $x = 1.01$.

If we have time, If you want hints, we can look at `demonstration1.m` on the github.[7]

ϵ.3 Automatic Differentiation: The Big Picture

Put simply, automatic differentiation turns function evaluation into derivative evaluation (see Figure ϵ.3 for a summary). Dual numbers provide one way of implementing forward-mode automatic differentiation. Forward-mode is preferable for functions of the form $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n \ll m$. For functions with $n \gg m$ (for example, in neural networks), a method called "reverse-mode" is preferred. It is beyond the scope of this lesson, but interested readers can read the paper Baydin et al. (2018) for more details.

References

Baydin, Atilim Gunes et al. (2018). "Automatic differentiation in machine learning: a survey". In: *Journal of Marchine Learning Research* 18, pp. 1–43.

[7] `https://github.com/williamjsdavis/SIO112-Derivatives`

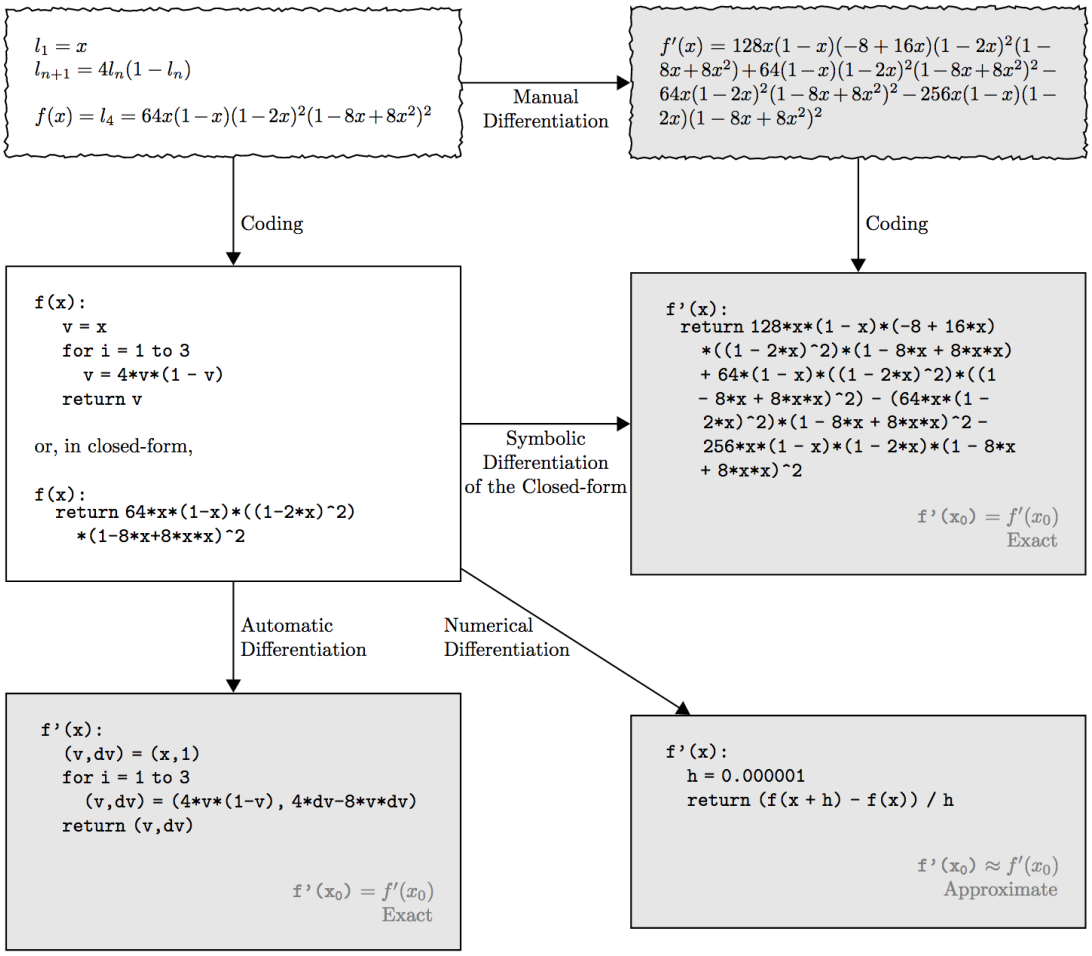

Figure ϵ.1: From (Baydin et al., 2018).