Introduction

These notes are based on the class as it was run for the second time in January 2023, taught by Professors Alan Edelman and Steven G. Johnson at MIT. The previous version of this course, run in January 2022, can be found on OCW here.

Both Professors Edelman and Johnson use he/him pronouns and are in the Department of Mathematics at MIT; Prof. Edelman is also a Professor in the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) running the Julia lab, while Prof. Johnson is also a Professor in the Department of Physics.

Here is a description of the course.:

> We all know that typical calculus course sequences begin with univariate and vector calculus, respectively. Modern applications such as machine learning and large-scale optimization require the next big step, "matrix calculus" and calculus on arbitrary vector spaces.
>
> This class covers a coherent approach to matrix calculus showing techniques that allow you to think of a matrix holistically (not just as an array of scalars), generalize and compute derivatives of important matrix factorizations and many other complicated-looking operations, and understand how differentiation formulas must be re-imagined in large-scale computing. We will discuss "reverse" ("adjoint", "backpropagation") differentiation and how modern automatic differentiation is more computer science than calculus (it is neither symbolic formulas nor finite differences).

The class involved numerous example numerical computations using the Julia language, which you can install on your own computer following these instructions. The material for this class is also located on GitHub at https://github.com/mitmath/matrixcalc.

1 Overview and Motivation

Firstly, where does matrix calculus fit into the MIT course catalog? Well, there are 18.01 (Single-Variable Calculus) and 18.02 (Vector Calculus) that students are required to take at MIT. But it seems as though this sequence of material is being cut off arbitrarily:

$$\text{Scalar} \;\rightarrow\; \text{Vector} \;\rightarrow \text{Matrices} \rightarrow\; \text{Higher-Order Arrays?}$$

After all, this is how the sequence is portrayed in many computer programming languages, including Julia! Why should calculus stop with vectors?

In the last decade, linear algebra has taken on larger and larger importance in numerous areas, such as machine learning, statistics, engineering, etc. In this sense, linear algebra has gradually taken over a much larger part of today's tools for lots of areas of study—now everybody needs linear algebra. So it makes sense that we would *want* to do calculus on these higher-order arrays, and it won't be a simple/obvious generalization (for instance, $\frac{d}{dA}A^2 \neq 2A$ for non-scalar matrices A).

More generally, the subjects of *differentiation* and *sensitivity analysis* are much deeper than one might suspect from the simple rules learned in first- or second-semester calculus. Differentiating functions whose inputs and/or outputs are in more complicated vector spaces (e.g. matrices, functions, or more) is one part of this subject. Another topic is the *efficient* evaluation of derivatives of functions involving very complicated calculations, from neural networks to huge engineering simulations—this leads to the topic of "adjoint" or "reverse-mode" differentiation, also known as "backpropagation." *Automatic differentiation (AD)* of computer programs by compilers is another surprising topic, in which the computer does something very different from the typical human process of first writing out an explicit symbolic formula and then passing the chain rule through it. These are only a few examples: the key point is that differentiation is more complicated than you may realize, and that these complexities are increasingly relevant for a wide variety of applications.

Let's quickly talk about some of these applications.

1.1 Applications

Applications: Machine learning

Machine learning has numerous buzzwords associated with it, including but not limited to: parameter optimization, stochastic gradient descent, automatic differentiation, and backpropagation. In this whole collage you can see a fraction of how matrix calculus applies to machine learning. It is recommended that you look into some of these topics yourself if you are interested.

Applications: Physical modeling

Large physical simulations, such as engineering-design problems, are increasingly characterized by *huge* numbers of parameters, and the *derivatives* of simulation outputs with respect to these parameters is crucial in order to evaluate sensitivity to uncertainties as well as to apply large-scale optimization.

For example, the shape of an airplane wing might be characterized by thousands of parameters, and if you can compute the derivative of the drag force (from a large fluid-flow simulation) with respect to these parameters then you could optimize the wing shape to minimize the drag for a given lift or other constraints.

An extreme version of such parameterization is known as "topology optimization," in which the material at "every point" in space is potentially a degree of freedom, and optimizing over these parameters can discover not only a

optimal shape but an optimal *topology* (how materials are connected in space, e.g. how many holes are present). For example, topology optimization has been applied in mechanical engineering to design the cross sections of airplane wings, artificial hips, and more into a complicated lattice of metal struts (e.g. minimizing weight for a given strength).

Besides engineering design, complicated differentiation problems arise in *fitting* unknown parameters of a model to experimental data, and also in *evaluating uncertainties* in the outputs of models with imprecise parameters/inputs. This is closely related to regression problems in statistics, as discussed below, except that here the model might be a giant set of differential equations with some unknown parameters.

Applications: Data science and multivariable statistics

In multivariate statistics, models are often framed in terms of matrix inputs and outputs (or even more complicated objects such as tensors). For example, a "simple" linear multivariate matrix model might be $Y(X) = XB+U$, where B is an unknown matrix of coefficients (to be determined by some form of fit/regression) and U is unknown matrix of random noise (that prevents the model from exactly fitting the data). Regression then involves minimizing some function of the error $U(B) = Y - XB$ between the model XB and data Y; for example, a matrix norm $\|U\|_F^2 = \operatorname{tr} U^T U$, a determinant $\det U^T U$, or more complicated functions. Estimating the best-fit coefficients B, analyzing uncertainties, and many other statistical analyses require differentiating such functions with respect to B or other parameters. A recent review article on this topic is Liu et al. (2022): "Matrix differential calculus with applications in the multivariate linear model and its diagnostics" (https://doi.org/10.1016/j.sctalk.2023.100274).

Applications: Automatic differentiation

Typical differential calculus classes are based on symbolic calculus, with students essentially learning to do what Mathematica or Wolfram Alpha can do. Even if you are using a computer to take derivatives symbolically, to use this effectively you need to understand what is going on beneath the hood. But while, similarly, some numerics may show up for a small portion of this class (such as to approximate a derivative using the difference quotient), *today's* automatic differentiation is neither of those two things. It is more in the field of the computer science topic of compiler technology than mathematics. However, the underlying mathematics of automatic differentiation is interesting, and we will learn about this in this class!

Even *approximate* computer differentiation is more complicated than you might expect. For single-variable functions $f(x)$, derivatives are defined as the limit of a difference $[f(x+\delta x) - f(x)]/\delta x$ as $\delta x \to 0$. A crude "finite-difference" approximation is simply to approximate $f'(x)$ by this formula for a small δx, but this turns out to raise many interesting issues involving balancing truncation and roundoff errors, higher-order approximations, and numerical extrapolation.

1.2 First Derivatives

The derivative of a function of one variable is itself a function of one variable– it simply is (roughly) defined as the linearization of a function. I.e., it is of the form $(f(x) - f(x_0)) \approx f'(x_0)(x - x_0)$. In this sense, "everything is easy" with scalar functions of scalars (by which we mean, functions that take in one number and spit out one number).

There are occasionally other notations used for this linearization:

- $\delta y \approx f'(x)\delta x$,
- $dy = f'(x)dx$,
- $(y - y_0) \approx f'(x_0)(x - x_0)$,

- and $df = f'(x)dx$.

This last one will be the preferred of the above for this class. One can think of dx and dy as "really small numbers." In mathematics, they are called infinitesimals, defined rigorously via taking limits. Note that here we do not want to divide by dx. While this is completely fine to do with scalars, once we get to vectors and matrices you can't always divide!

The numerics of such derivatives are simple enough to play around with. For instance, consider the function $f(x) = x^2$ and the point $(x_0, f(x_0)) = (3, 9)$. Then, we have the following numerical values near $(3, 9)$:

$$\begin{aligned} f(3.\mathbf{0001}) &= 9.\mathbf{0006}0001 \\ f(3.\mathbf{00001}) &= 9.\mathbf{00006}00001 \\ f(3.\mathbf{000001}) &= 9.\mathbf{000006}000001 \\ f(3.\mathbf{0000001}) &= 9.\mathbf{0000006}0000001. \end{aligned}$$

Here, the bolded digits on the left are Δx and the bolded digits on the right are Δy. Notice that $\Delta y = 6\Delta x$. Hence, we have that

$$f(3 + \Delta x) = 9 + \Delta y = 9 + 6\Delta x \implies f(3 + \Delta x) - f(3) = 6\Delta x \approx f'(3)\Delta x.$$

Therefore, we have that the linearization of x^2 at $x = 3$ is the function $f(x) - f(3) \approx 6(x - 3)$.

We now leave the world of scalar calculus and enter the world of vector/matrix calculus! Professor Edelman invites us to think about matrices *holistically*—not just as a table of numbers.

The notion of linearizing your function will conceptually carry over as we define the derivative of functions which take in/spit out more than one number. Of course, this means that the derivative will have a different "shape" than a single number. Here is a table on the *shape* of the first derivative. The inputs of the function are given on the left hand side of the table, and the outputs of the function are given across the top.

input ↓ and output →	scalar	vector	matrix
scalar	scalar	vector (for instance, velocity)	matrix
vector	gradient = (column) vector	matrix (called the Jacobian matrix)	higher order array
matrix	matrix	higher order array	higher order array

You will ultimately learn how to do any of these in great detail eventually in this class! The purpose of this table is to plant the notion of differentials as linearization. Let's look at an example.

Example 1

Let $f(x) = x^T x$, where x is a 2×1 matrix and the output is thus a 1×1 matrix. Confirm that $2x_0^T dx$ is indeed the differential of f at $x_0 = \begin{pmatrix} 3 & 4 \end{pmatrix}^T$.

Firstly, let's compute $f(x_0)$:

$$f(x_0) = x_0^T x_0 = 3^2 + 4^2 = 25.$$

Then, suppose $dx = [.001, .002]$. Then, we would have that

$$f(x + dx) = (3.001)^2 + (4.002)^2 = 25.\mathbf{022}005.$$

Then, notice that $2x_0^T\,dx = 2\begin{pmatrix}3 & 4\end{pmatrix}^T dx = .022$. Hence, we have that

$$f(x_0 + dx) - f(x_0) \approx 2x_0^T dx = .022.$$

As we will see right now, the $2x_0^T dx$ didn't come from nowhere!

1.3 Intro: Matrix and Vector Product Rule

For matrices, we in fact still have a product rule! We will discuss this in much more detail in later chapters, but let's begin here with a small taste.

Theorem 2 (Differential Product Rule)
Let A, B be two matrices. Then, we have the differential product rule for AB:

$$d(AB) = (dA)B + A(dB).$$

By the differential of the matrix A, we think of it as a small (unconstrained) change in the matrix A. Later, constraints may be places on the allowed perturbations.

Notice however, that (by our table) the derivative of a matrix is a matrix! So generally speaking, the products will not commute.

If x is a vector, then by the differential product rule we have

$$d(x^T x) = (dx^T)x + x^T(dx).$$

However, notice that this is a dot product, and dot products commute (since $\sum a_i \cdot b_i = \sum b_i \cdot a_i$), we have that

$$d(x^T x) = (2x)^T dx.$$

Remark 3. *The way the product rule works for vectors as matrices is that transposes "go for the ride." See the next example below.*

Example 4
By the product rule. we have

1. $d(u^T v) = (du)^T v + u^T(dv) = v^T du + u^T dv$ since dot products commute.
2. $d(uv^T) = (du)v^T + u(dv)^T$.

Remark 5. *The way to prove these sorts of statements can be seen in Section 2.*

18.S096 Matrix Calculus for Machine Learning and Beyond
Independent Activities Period (IAP) 2023