3 Jacobians of Matrix Functions

When we have a function that has *matrices* as inputs and/or outputs, we have already seen in the previous lectures that we can still define the derivative as a linear operator by a *formula* for f' mapping a small change in input to the corresponding small change in output. However, when you first learned linear algebra, probably most linear operations were represented by matrices multiplying vectors, and it may take a while to get used to thinking of linear operations more generally. In this chapter, we discuss how it is still *possible* to represent f' by a **Jacobian matrix** even for matrix inputs/outputs, and how the most common technique to do this involves **matrix "vectorization"** and a new type of matrix operation, a **Kronecker product**. This gives us another way to think about our f' linear operators that is occasionally convenient, but at the same time it is important to become comfortable with other ways of writing down linear operators too—sometimes, the explicit Jacobian-matrix approach can obscure key structure, and it is often computationally inefficient as well.

For this section of the notes, we refer to the linked Pluto Notebook for computational demonstrations of this material in Julia, illustrating multiple views of the derivative of the square A^2 of 2×2 matrices A.

3.1 Derivatives of matrix functions: Linear operators

As we have already emphasized, the derivative f' is the linear operator that maps a small change in the input to a small change in the output. This idea can take an unfamiliar form, however, when applied to functions $f(A)$ that map matrix inputs A to matrix outputs. For example, we've already considered the following functions on square $m \times m$ matrices:

- $f(A) = A^3$, which gives $df = f'(A)[dA] = dA\,A^2 + A\,dA\,A + A^2\,dA$.
- $f(A) = A^{-1}$, which gives $df = f'(A)[dA] = -A^{-1}\,dA\,A^{-1}$

Example 20

An even simpler example is the *matrix-square* function:

$$f(A) = A^2\,,$$

which by the product rule gives

$$df = f'(A)[dA] = dA\,A + A\,dA\,.$$

You can also work this out explicitly from $df = f(A+dA) - f(A) = (A+dA)^2 - A^2$, dropping the $(dA)^2$ term.

In all of these examples, $f'(A)$ is described by a simple formula for $f'(A)[dA]$ that relates an arbitrary change dA in A to the change $df = f(A+dA) - f(A)$ in f, to first order. If the differential is distracting you, realize that we can plug any matrix X we want into this formula, not just an "infinitesimal" change dA, e.g. in our matrix-square example we have

$$f'(A)[X] = XA + AX$$

for an arbitrary X (a directional derivative, from Sec. 2.2.1). This is *linear* in X: if we scale or add inputs, it scales or adds outputs, respectively:

$$f'(A)[2X] = 2XA + A\,2X = 2(XA + AX) = 2f'(A)[X]\,,$$

$$f'(A)[X+Y] = (X+Y)A + A(X+Y) = XA + YA + AX + AY = XA + AX + YA + AY$$
$$= f'(A)[X] + f'(A)[Y]\,.$$

This is a perfectly good way to define a linear operation! We are *not* expressing it here in the familiar form $f'(A)[X] = (\text{some matrix?}) \times (X \text{ vector?})$, and that's okay! A formula like $XA + AX$ is easy to write down, easy to understand, and easy to compute with.

But sometimes you still may want to think of f' as a single "Jacobian" matrix, using the most familiar language of linear algebra, and it is possible to do that! If you took a sufficiently abstract linear-algebra course, you may have learned that *any* linear operator can be represented by a matrix once you choose a basis for the input and output vector spaces. Here, however, we will be much more concrete, because there is a conventional "Cartesian" basis for matrices A called "vectorization", and in this basis linear operators like $AX + XA$ are particularly easy to represent in matrix form once we introduce a new type of matrix product that has widespread applications in "multidimensional" linear algebra.

3.2 A simple example: The two-by-two matrix-square function

To begin with, let's look in more detail at our matrix-square function

$$f(A) = A^2$$

for the simple case of 2×2 matrices, which are described by only four scalars, so that we can look at every term in the derivative explicitly. In particular,

Example 21

For a 2×2 matrix

$$A = \begin{pmatrix} p & r \\ q & s \end{pmatrix},$$

the matrix-square function is

$$f(A) = A^2 = \begin{pmatrix} p & r \\ q & s \end{pmatrix}\begin{pmatrix} p & r \\ q & s \end{pmatrix} = \begin{pmatrix} p^2+qr & pr+rs \\ pq+qs & qr+s^2 \end{pmatrix}.$$

Written out explicitly in terms of the matrix entries (p, q, r, s) in this way, it is natural to think of our function as mapping 4 scalar inputs to 4 scalar outputs. That is, we can think of f as equivalent to a "vectorized" function $\tilde{f} : \mathbb{R}^4 \to \mathbb{R}^4$ given by

$$\tilde{f}\left(\begin{pmatrix} p \\ q \\ r \\ s \end{pmatrix}\right) = \begin{pmatrix} p^2+qr \\ pq+qs \\ pr+rs \\ qr+s^2 \end{pmatrix}.$$

Converting a matrix into a column vector in this way is called **vectorization**, and is commonly denoted by the

operation "vec":

$$\operatorname{vec} A = \operatorname{vec}\begin{pmatrix} p & r \\ q & s \end{pmatrix} = \begin{matrix} A_{1,1} \\ A_{2,1} \\ A_{1,2} \\ A_{2,2} \end{matrix}\begin{pmatrix} p \\ q \\ r \\ s \end{pmatrix},$$

$$\operatorname{vec} f(A) = \operatorname{vec}\begin{pmatrix} p^2+qr & pr+rs \\ pq+qs & qr+s^2 \end{pmatrix} = \begin{pmatrix} p^2+qr \\ pq+qs \\ pr+rs \\ qr+s^2 \end{pmatrix}.$$

In terms of vec, our "vectorized" matrix-squaring function $\tilde{f}$ is defined by

$$\tilde{f}(\operatorname{vec} A) = \operatorname{vec} f(A) = \operatorname{vec}(A^2)\,.$$

More generally,

Definition 22

The **vectorization** $\operatorname{vec} A \in \mathbb{R}^{mn}$ of any $m \times n$ matrix $A \in \mathbb{R}^{m\times n}$ is a defined by simply **stacking the columns** of A, from left to right, into a column vector $\operatorname{vec} A$. That is, if we denote the n columns of A by m-component vectors $\vec{a}_1, \vec{a}_2, \ldots \in \mathbb{R}^m$, then

$$\operatorname{vec} A = \operatorname{vec}\underbrace{\begin{pmatrix} \vec{a}_1 & \vec{a}_2 & \cdots & \vec{a}_n \end{pmatrix}}_{A\in\mathbb{R}^{m\times n}} = \begin{pmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vdots \\ \vec{a}_n \end{pmatrix} \in \mathbb{R}^{mn}$$

is an mn-component column vector containing all of the entries of A.

On a computer, matrix entries are typically stored in a consecutive sequence of memory locations, which can be viewed a form of vectorization. In fact, $\operatorname{vec} A$ corresponds exactly to what is known as "column-major" storage, in which the column entries are stored consecutively; this is the default format in Fortran, Matlab, and Julia, for example, and the venerable Fortran heritage means that column major is widely used in linear-algebra libraries.

.

Problem 23

The vector $\operatorname{vec} A$ corresponds to the coefficients you get when you express the $m \times n$ matrix A in a *basis* of matrices. What is that basis?

Vectorization turns unfamilar things (like matrix functions and derivatives thereof) into familiar things (like vector functions and Jacobians or gradients thereof). In that way, it can be a very attractive tool, almost *too* attractive—why do "matrix calculus" if you can turn everything back into ordinary multivariable calculus? Vectorization has its drawbacks, however: conceptually, it can obscure the underlying mathematical structure (e.g. $\tilde{f}$ above doesn't look much like a matrix square A^2), and computationally this loss of structure can sometimes lead to severe inefficiencies (e.g. forming huge $m^2 \times m^2$ Jacobian matrices as discussed below). Overall, we believe

that the *primary* way to study matrix functions like this should be to view them as having matrix inputs (A) and matrix outputs (A^2), and that one should likewise generally view the derivatives as linear operators on matrices, not vectorized versions thereof. However, it is still useful to be familiar with the vectorization viewpoint in order to have the benefit of an alternative perspective.

3.2.1 The matrix-squaring four-by-four Jacobian matrix

To understand Jacobians of functions (from matrices to matrices), let's begin by considering a basic question:

Question 24. *What is the* size *of the Jacobian of the matrix-square function?*

Well, if we view the matrix squaring function via its vectorized equivalent $\tilde{f}$, mapping $\mathbb{R}^4 \mapsto \mathbb{R}^4$ (4-component column vectors to 4-component column vectors), the Jacobian would be a 4×4 matrix (formed from the derivatives of each output component with respect to each input component). Now let's think about a more general square matrix A: an $m \times m$ matrix. If we wanted to find the Jacobian of $f(A) = A^2$, we could do so by the same process and (symbolically) obtain an $m^2 \times m^2$ matrix (since there are m^2 inputs, the entries of A, and m^2 outputs, the entries of A^2). Explicit computation of these m^4 partial derivatives is rather tedious even for small m, but is a task that symbolic computational tools in e.g. Julia or Mathematica can handle. In fact, as seen in the Notebook, Julia spits out the Jacobian quite easily. For the $m = 2$ case that we wrote out explicitly above, you can either take the derivative of $\tilde{f}$ by hand or use Julia's symbolic tools to obtain the Jacobian:

$$\tilde{f}' = \begin{array}{c} \\ (1,1) \\ (2,1) \\ (1,2) \\ (2,2) \end{array} \begin{array}{c} \begin{array}{cccc} (1,1) & (2,1) & (1,2) & (2,2) \end{array} \\ \begin{pmatrix} 2p & r & q & 0 \\ q & p+s & 0 & q \\ r & 0 & p+s & r \\ 0 & r & q & 2s \end{pmatrix} \end{array}. \tag{3}$$

For example, the first row of $\tilde{f}'$ consists of the partial derivatives of $p^2 + qr$ (the first output) with respect to the 4 inputs $p, q, r,$ and s. Here, we have labeled the rows by the (row,column) indices $(j_{\text{out}}, k_{\text{out}})$ of the entries in the "output" matrix $d(A^2)$, and have labeled the columns by the indices $(j_{\text{in}}, k_{\text{in}})$ of the entries in the "input" matrix A. Although we have written the Jacobian $\tilde{f}'$ as a "2d" matrix, you can therefore also imagine it to be a "4d" matrix indexed by $j_{\text{out}}, k_{\text{out}}, j_{\text{in}}, k_{\text{in}}$.

However, the matrix-calculus approach of viewing the derivative $f'(A)$ as a *linear transformation on matrices* (as we derived above),

$$f'(A)[X] = XA + AX\,,$$

seems to be much more revealing than writing out an explicit component-by-component "vectorized" Jacobian $\tilde{f}'$, and gives a formula for any $m \times m$ matrix without laboriously requiring us to take m^4 partial derivatives one-by-one. If we really want to pursue the vectorization perspective, we need a way to recapture some of the structure that is obscured by tedious componentwise differentiation. A key tool to bridge the gap between the two perspectives is a type of matrix operation that you may not be familiar with: **Kronecker products** (denoted $\otimes$).

3.3 Kronecker Products

A linear operation like $f'(A)[X] = XA + AX$ can be thought of as a "higher-dimensional matrix:" ordinary "2d" matrices map "1d" column vectors to 1d column vectors, whereas to map 2d matrices to 2d matrices you might imagine a "4d" matrix (sometimes called a *tensor*). To transform 2d matrices back into 1d vectors, we already saw the concept of vectorization (vec A). A closely related tool, which transforms "higher dimensional" linear operations

on matrices back into "2d" matrices for the vectorized inputs/outputs, is the Kronecker product $A \otimes B$. Although they don't often appear in introductory linear-algebra courses, Kronecker products show up in a wide variety of mathematical applications where multidimensional data arises, such as multivariate statistics and data science or multidimensional scientific/engineering problems.

Definition 25

If A is an $m \times n$ matrix with entries a_{ij} and B is a $p \times q$ matrix, then their **Kronecker product** $A \otimes B$ is defined by

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \implies \underbrace{A}_{m\times n} \otimes \underbrace{B}_{p\times q} = \underbrace{\begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}}_{mp\times nq},$$

so that $A \otimes B$ is an $mp \times nq$ matrix formed by "pasting in" a copy of B multiplying every element of A.

For example, consider 2×2 matrices

$$A = \begin{pmatrix} p & r \\ q & s \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} a & c \\ b & d \end{pmatrix}.$$

Then $A \otimes B$ is a 4×4 matrix containing all possible products of entries A with entries of B. Note that $A \otimes B \neq B \otimes A$ (but the two are related by a re-ordering of the entries):

$$A \otimes B = \begin{pmatrix} pB & rB \\ qB & sB \end{pmatrix} = \begin{pmatrix} pa & pc & ra & rc \\ pb & pd & rb & rd \\ qa & qc & sa & sc \\ qb & qd & sb & sd \end{pmatrix} \quad \neq \quad B \otimes A = \begin{pmatrix} aA & cA \\ bA & dA \end{pmatrix} = \begin{pmatrix} ap & ar & cp & cr \\ aq & as & cq & cs \\ bp & br & dp & dr \\ bq & bs & dq & ds \end{pmatrix},$$

where we've colored one copy of B red for illustration. See the Notebook for more examples of Kronecker products of matrices (including some with pictures rather than numbers!).

Above, we saw that $f(A) = A^2$ at $A = \begin{pmatrix} p & r \\ q & s \end{pmatrix}$ could be thought of as an equivalent function $\tilde{f}(\operatorname{vec} A)$ mapping column vectors of 4 inputs to 4 outputs ($\mathbb{R}^4 \mapsto \mathbb{R}^4$), with a 4×4 Jacobian that we (or the computer) laboriously computed as 16 element-by-element partial derivatives. It turns out that this result can be obtained *much* more elegantly once we have a better understanding of Kronecker products. We will find that the 4×4 "vectorized" Jacobian is simply

$$\tilde{f}' = \mathrm{I}_2 \otimes A + A^T \otimes \mathrm{I}_2,$$

where I_2 is the 2×2 identity matrix. That is, the matrix linear operator $f'(A)[dA] = dA\,A + A\,dA$ is equivalent, after vectorization, to:

$$\operatorname{vec} \underbrace{f'(A)[dA]}_{dA\,A + A\,dA} = \underbrace{(\mathrm{I}_2 \otimes A + A^T \otimes \mathrm{I}_2)}_{\tilde{f}'} \operatorname{vec} dA = \underbrace{\begin{pmatrix} 2p & r & q & 0 \\ q & p+s & 0 & q \\ r & 0 & p+s & r \\ 0 & r & q & 2s \end{pmatrix}}_{\tilde{f}'} \underbrace{\begin{pmatrix} dp \\ dq \\ dr \\ ds \end{pmatrix}}_{\operatorname{vec} dA}.$$

In order to understand *why* this is the case, however, we must first build up some understanding of the algebra of Kronecker products. To start with, a good exercise is to convince yourself of a few simpler properties of Kronecker

products:

Problem 26

From the definition of the Kronecker product, derive the following identities:

1. $(A \otimes B)^T = A^T \otimes B^T$.
2. $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.
3. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. (Follows from property 2.)
4. $A \otimes B$ is orthogonal (its transpose is its inverse) if A and B are orthogonal. (From properties 1 & 3.)
5. $\det(A \otimes B) = \det(A)^m \det(B)^n$, where $A \in \mathbb{R}^{n,n}$ and $B \in \mathbb{R}^{m,m}$.
6. $\operatorname{tr}(A \otimes B) = (\operatorname{tr} A)(\operatorname{tr} B)$.
7. Given eigenvectors/values $Au = \lambda u$ and $Bv = \mu v$ of A and B, then $\lambda\mu$ is an eigenvalue of $A \otimes B$ with eigenvector $u \otimes v$. (Also, since $u \otimes v = \operatorname{vec} X$ where $X = vu^T$, you can relate this via Prop. 27 below to the identity $BXA^T = Bv(Au)^T = \lambda\mu X$.)

3.3.1 Key Kronecker-product identity

In order to convert linear operations like $AX + XA$ into Kronecker products via vectorization, the key identity is:

Proposition 27

Given (compatibly sized) matrices A, B, C, we have

$$(A \otimes B) \operatorname{vec}(C) = \operatorname{vec}(BCA^T).$$

We can thus view $A \otimes B$ as a vectorized equivalent of the linear operation $C \mapsto BCA^T$. We are tempted to introduce a parallel notation $(A \otimes B)[C] = BCA^T$ for the "non-vectorized" version of this operation, although this notation is not standard.

One possible mnemonic for this identity is that the B is just to the left of the C while the A "circles around" to the right and gets transposed.

Where does this identity come from? We can break it into simpler pieces by first considering the cases where either A or B is an identity matrix I (of the appropriate size). To start with, suppose that $A = \mathrm{I}$, so that $BCA^T = BC$. What is $\operatorname{vec}(BC)$? If we let $\vec{c}_1, \vec{c}_2, \ldots$ denote the columns of C, then recall that BC simply multiples B on the left with each of the columns of C:

$$BC = B \begin{pmatrix} \vec{c}_1 & \vec{c}_2 & \cdots \end{pmatrix} = \begin{pmatrix} B\vec{c}_1 & B\vec{c}_2 & \cdots \end{pmatrix} \Longrightarrow \operatorname{vec}(BC) = \begin{pmatrix} B\vec{c}_1 \\ B\vec{c}_2 \\ \vdots \end{pmatrix}.$$

Now, how can we get this $\operatorname{vec}(BC)$ vector as something multiplying $\operatorname{vec} C$? It should be immediately apparent that

$$\operatorname{vec}(BC) = \begin{pmatrix} B\vec{c}_1 \\ B\vec{c}_2 \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} B & & \\ & B & \\ & & \ddots \end{pmatrix}}_{\mathrm{I} \otimes B} \underbrace{\begin{pmatrix} \vec{c}_1 \\ \vec{c}_2 \\ \vdots \end{pmatrix}}_{\operatorname{vec} C},$$

but this matrix is exactly the Kronecker product $I \otimes B$! Hence, we have derived that

$$(\mathrm{I} \otimes B)\,\mathrm{vec}\,C = \mathrm{vec}(BC).$$

What about the A^T term? This is a little trickier, but again let's simplify to the case where $B = \mathrm{I}$, in which case $BCA^T = CA^T$. To vectorize this, we need to look at the columns of CA^T. What is the first column of CA^T? It is a linear combination of the columns of C whose coefficients are given by the first column of A^T (= first row of A):

$$\text{column 1 of } CA^T = \sum_j a_{1j}\vec{c}_j\ .$$

Similarly for column 2, etc, and we then "stack" these columns to get $\mathrm{vec}(CA^T)$. But this is exactly the formula for multipling a matrix A by a vector, if the "elements" of the vector were the columns $\vec{c}_j$. Written out explicitly, this becomes:

$$\mathrm{vec}(CA^T) = \begin{pmatrix} \sum_j a_{1j}\vec{c}_j \\ \sum_j a_{2j}\vec{c}_j \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} a_{11}\,\mathrm{I} & a_{12}\,\mathrm{I} & \cdots \\ a_{21}\,\mathrm{I} & a_{22}\,\mathrm{I} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}}_{A\otimes\mathrm{I}} \underbrace{\begin{pmatrix} \vec{c}_1 \\ \vec{c}_2 \\ \vdots \end{pmatrix}}_{\mathrm{vec}\,C},$$

and hence we have derived

$$(A \otimes \mathrm{I})\,\mathrm{vec}\,C = \mathrm{vec}(CA^T).$$

The full identity $(A \otimes B)\,\mathrm{vec}(C) = \mathrm{vec}(BCA^T)$ can then be obtained by straightforwardly combining these two derivations: replace CA^T with BCA^T in the second derivation, which replaces $\vec{c}_j$ with $B\vec{c}_j$ and hence I with B.

3.3.2 The Jacobian in Kronecker-product notation

So now we want to use Prop. 27 to calculate the Jacobian of $f(A) = A^2$ in terms of the Kronecker product. Let dA be our C in Prop. 27. We can now immediately see that

$$\mathrm{vec}(A\,dA + dA\,A) = \underbrace{(\mathrm{I}\otimes A + A^T \otimes \mathrm{I})}_{\text{Jacobian } \tilde{f}'(\mathrm{vec}\,A)}\,\mathrm{vec}(dA)\,,$$

where I is the identity matrix of the same size as A. We can also write this in our "non-vectorized" linear-operator notation:

$$A\,dA + dA\,A = (\mathrm{I}\otimes A + A^T \otimes \mathrm{I})[dA]\,.$$

In the 2×2 example, these Kronecker products can be computed explicitly:

$$\underbrace{\begin{pmatrix} 1 & \\ & 1 \end{pmatrix}}_{\mathrm{I}} \otimes \underbrace{\begin{pmatrix} p & r \\ q & s \end{pmatrix}}_{A} + \underbrace{\begin{pmatrix} p & q \\ r & s \end{pmatrix}}_{A^T} \otimes \underbrace{\begin{pmatrix} 1 & \\ & 1 \end{pmatrix}}_{\mathrm{I}} = \underbrace{\begin{pmatrix} p & r & & \\ q & s & & \\ & & p & r \\ & & q & s \end{pmatrix}}_{\mathrm{I}\otimes A} + \underbrace{\begin{pmatrix} p & & q & \\ & p & & q \\ r & & s & \\ & r & & s \end{pmatrix}}_{A^T\otimes\mathrm{I}}$$

$$= \begin{pmatrix} 2p & r & q & 0 \\ q & p+s & 0 & q \\ r & 0 & p+s & r \\ 0 & r & q & 2s \end{pmatrix} = \tilde{f}'\,,$$

which exactly matches our laboriously computed Jacobian $\tilde{f}'$ from earlier!

Example 28

For the matrix-cube function A^3, where A is an $m \times m$ square matrix, compute the $m^2 \times m^2$ Jacobian of the vectorized function $\operatorname{vec}(A^3)$.

Let's use the same trick for the matrix-cube function. Sure, we could laboriously compute the Jacobian via element-by-element partial derivatives (which is done nicely by symbolic computing in the notebook), but it's much easier and more elegant to use Kronecker products. Recall that our "non-vectorized" matrix-calculus derivative is the linear operator:

$$(A^3)'[dA] = dA\,A^2 + A\,dA\,A + A^2\,dA,$$

which now vectorizes by three applications of the Kronecker identity:

$$\operatorname{vec}(dA\,A^2 + A\,dA\,A + A^2\,dA) = \underbrace{\left((A^2)^T \otimes \mathrm{I} + A^T \otimes A + \mathrm{I} \otimes A^2\right)}_{\text{vectorized Jacobian}} \operatorname{vec}(dX)\,.$$

You could go on to find the Jacobians of A^4, A^5, and so on, or any linear combination of matrix powers. Indeed, you could imagine applying a similar process to the Taylor series of any (analytic) matrix function $f(A)$, but it starts to become awkward. Later on (and in homework), we will discuss more elegant ways to differentiate other matrix functions, not as vectorized Jacobians but as linear operators on matrices.

3.3.3 The computational cost of Kronecker products

One must be cautious about using Kronecker products as a *computational* tool, rather than as more of a *conceptual* tool, because they can easily cause the computational cost of matrix problems to explode far beyond what is necessary.

Suppose that A, B, and C are all $m \times m$ matrices. The cost of multiplying two $m \times m$ matrices (by the usual methods) scales proportional to $\sim m^3$, what the computer scientists call $\Theta(m^3)$ "complexity." Hence, the cost of the linear operation $C \mapsto BCA^T$ scales as $\sim m^3$ (two $m \times m$ multiplications). However, if we instead compute the *same answer* via $\operatorname{vec}(BCA^T) = (A \otimes B) \operatorname{vec} C$, then we must:

1. Form the $m^2 \times m^2$ matrix $A \otimes B$. This requires m^4 multiplications (all entries of A by all entries of B), and $\sim m^4$ memory storage. (Compare to $\sim m^2$ memory to store A or B. If m is 1000, this is a *million* times more storage, terabytes instead of megabytes!)
2. Multiply $A \otimes B$ by the vector $\operatorname{vec} C$ of m^2 entries. Multiplying an $n \times n$ matrix by a vector requires $\sim n^2$ operations, and here $n = m^2$, so this is again $\sim m^4$ arithmetic operations.

So, instead of $\sim m^3$ operations and $\sim m^2$ storage to compute BCA^T, using $(A \otimes B) \operatorname{vec} C$ requires $\sim m^4$ operations and $\sim m^4$ storage, vastly worse! Essentially, this is because $A \otimes B$ has a lot of structure that we are not exploiting (it is a *very special* $m^2 \times m^2$ matrix).

There are many examples of this nature. Another famous one involves solving the linear *matrix* equations

$$AX + XB = C$$

for an unknown matrix X, given A, B, C, where all of these are $m \times m$ matrices. This is called a "Sylvester equation." These are *linear* equations in our unknown X, and we can convert them to an ordinary system of m^2 linear equations by Kronecker products:

$$\operatorname{vec}(AX + XB) = (\mathrm{I} \otimes A + B^T \otimes \mathrm{I}) \operatorname{vec} X = \operatorname{vec} C,$$

which you can then solve for the m^2 unknowns $\operatorname{vec} X$ using Gaussian elimination. But the cost of solving an $m^2 \times m^2$ system of equations by Gaussian elimination is $\sim (m^2)^3 = m^6$. It turns out, however, that there are clever algorithms to solve $AX + XB = C$ in only $\sim m^3$ operations (with $\sim m^2$ memory)—for $m = 1000$, this saves a factor of $\sim m^3 = 10^9$ (a *billion*) in computational effort.

(Kronecker products can be a more practical computational tool for *sparse* matrices: matrices that are mostly zero, e.g. having only a few nonzero entries per row. That's because the Kronecker product of two sparse matrices is also sparse, avoiding the huge storage requirements for Kronecker products of non-sparse "dense" matrices. This can be a convenient way to assemble large sparse systems of equations for things like multidimensional PDEs.)

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