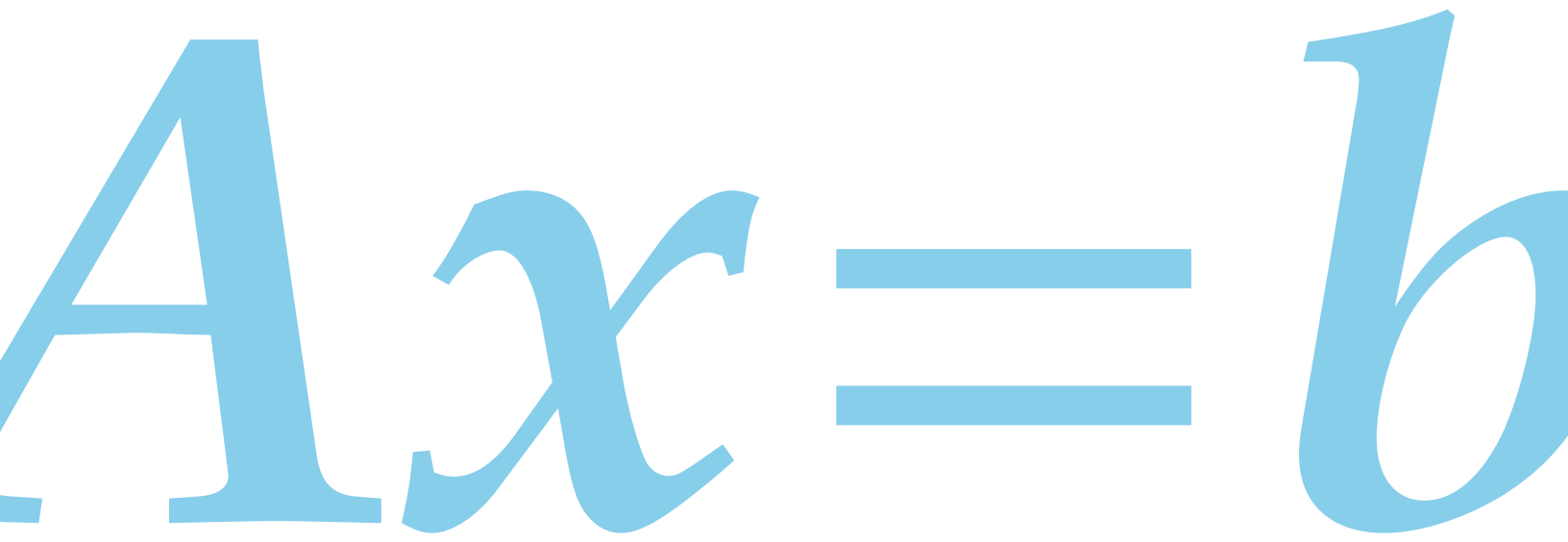
Matrix Methods for Computational Modeling and Data Analytics

Mark Embree
embree@vt.edu

Virginia Tech · Spring 2019

version of 21 January 2019



Matrix Methods for Computational Modeling and Data Analytics

Virginia Tech · Spring 2019

Mark Embree
embree@vt.edu

$Ax=b$

version of 21 January 2019

Chapter 1
Introduction

CMDA 3606 is about the linear algebra problem

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

in many wild and wonderful varieties. Most students are introduced to this equation in a sterile form, where $\mathbf{A}$ is a small, tidy matrix with integer entries whose origins are obscure, say

$$\mathbf{A} = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}.$$

We start this class by considering several physical situations that give rise to $\mathbf{A}\mathbf{x} = \mathbf{b}$ problems: these are physical systems at *equilibrium*, where $\mathbf{A}\mathbf{x} = \mathbf{b}$ encodes a *balance of forces*:

- $\mathbf{b}$ describes some applied force, *which we can usually measure*;
- $\mathbf{A}$ encodes the physical properties of the system (how it is connected, its material parameters), *which we also know, or can measure*;
- $\mathbf{x}$, the *unknown*, describes the system's reaction to the force $\mathbf{b}$.

This context gives physical meaning to otherwise abstract questions like, "Does $\mathbf{A}\mathbf{x} = \mathbf{b}$ have a unique solution?" When $\mathbf{A}$ models a structure,we will see how this question becomes roughly equivalent to "Is my building stable?"

When modeling realistic physical systems, the matrix $\mathbf{A}$ can be sufficiently large that the standard technique (Gaussian elimination) does not work. In this class we will study faster alternatives.

We will consider other variations of the $\mathbf{A}\mathbf{x} = \mathbf{b}$ problem. Sometimes we do not know the matrix $\mathbf{A}$, but we can conduct a series of physical experiments to apply known forces, $\mathbf{b}$ and *measure* the corresponding effects, $\mathbf{x}$. Can we use such experiments to *discover* $\mathbf{A}$?

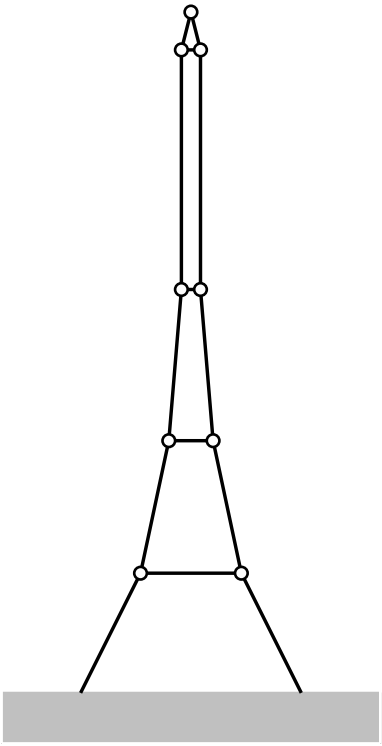


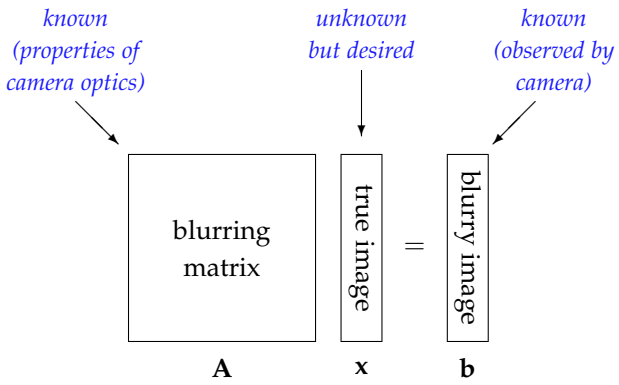
Figure 1.1: A primitive model of the Eiffel Tower. Is it stable?

This way of turning $\mathbf{Ax} = \mathbf{b}$ on its head is an example of an *inverse problem*. Often such problems are intimately connected to *least squares* and *optimization*: subjects that will occupy much of our semester.

Other situations arise where an exact solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$ exists: the theoreticians rest easy. But when you actually *compute* $\mathbf{x}$, you see that it looks like *garbage* (the polite term is *noise*): it obviously has no physical meaning!

The illustration below sketches one example of this phenomenon that arises in many problems in image science. The vector $\mathbf{x}$ contains some image that we want to see, while the vector $\mathbf{b}$ represents the image that we can measure with our camera. The matrix $\mathbf{A}$ encodes the blurring that inevitably occurs when the true image $\mathbf{x}$ is mapped to the observed image $\mathbf{b}$.

More specifically, $\mathbf{b}$ is a vector of pixel values. For example, each entry in $\mathbf{b}$ might be an integer between 0 and 255, representing a shade of 8-bit grayscale (0 = black, 255 = white). To make an image into a vector, we stack each column of pixel values, one on top of the other, scanning from left to right across the image.



We can estimate $\mathbf{A}$ based on properties of the camera's optics, and calibrate it by applying the camera to a few test images where we know $\mathbf{x}$.

We use the term *camera* generically; your camera might be a *telescope*: $\mathbf{A}$ can be calibrated by viewing well-studied objects, before turning your telescope to seek more interesting, unknown $\mathbf{b}$.

Here is a simple one-dimensional version of the blurring problem: scanning a UPC barcode. Consider the barcode shown below.

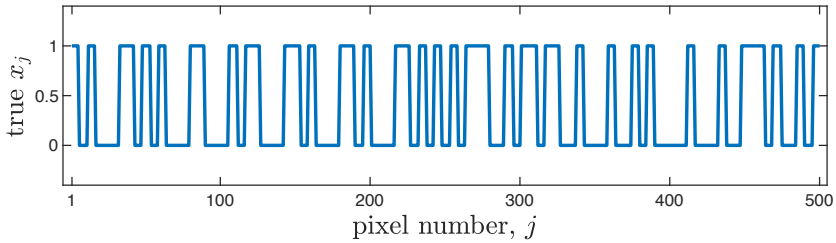
This application is mentioned by Per Christian Hansen in his book *Discrete Inverse Problems: Insight and Algorithms*, SIAM, 2010. Virginia Tech students can access the book for free: see the class website for a link.

Now take a horizontal slice of the barcode, shown in red below.

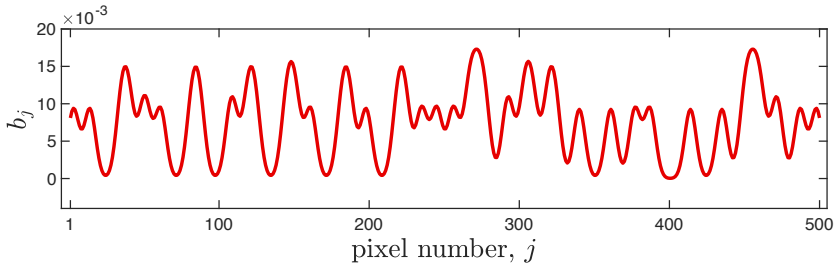
To turn this barcode into a length-n vector, we will *discretize* the red line into n pixels. Then the jth entry of $\mathbf{x}$ is determined by

$$x_j = \begin{cases} 0, & \text{the } j\text{th pixel is white;} \\ 1, & \text{the } j\text{th pixel is black.} \end{cases}$$

For $n = 500$, we obtain the function shown below.



Now we simulate the action of a supermarket barcode scanner, which detects the fuzzy version of this barcode shown in red in the plot below. This is the blurred vector, $\mathbf{b}$. From it we want to find the true barcode, $\mathbf{x}$.

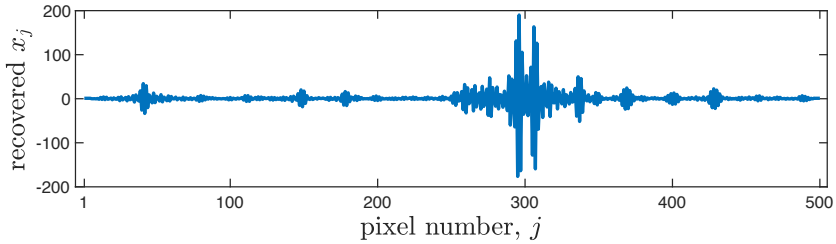


Suppose we know all about the blurring operation that made this image: we know the engineering behind the optics in the scanner, and so we know the matrix $\mathbf{A}$ exactly. Indeed, this $\mathbf{A}$ is *invertible*. Knowing the scanner $\mathbf{A}$ and the scanned image $\mathbf{b}$, we should be able to compute the original barcode:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

In MATLAB, compute `x = A\b`.

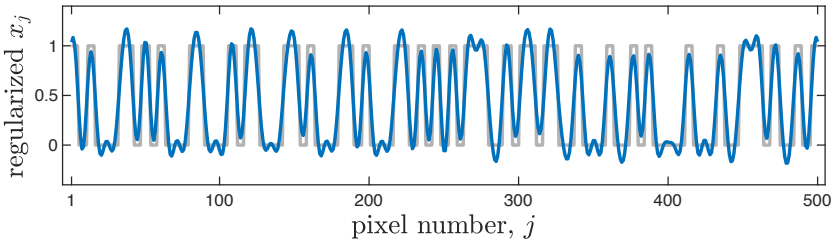
The resulting $\mathbf{x}$ *should* (in theory) exactly match the blue barcode plot shown above. Instead, MATLAB yields the following result: *garbage*!



Solutions to general $\mathbf{Ax} = \mathbf{b}$ problems are not always this poor! Deblurring problems typically give $\mathbf{A}$ matrices that are especially fragile and prone to this behavior. Later in the course we will understand how to identify such $\mathbf{A}$.

What could possibly go wrong? The matrix $\mathbf{A}$ is invertible, but it is very close to a matrix that is *not invertible*, and this makes the solution of $\mathbf{Ax} = \mathbf{b}$ very sensitive to the small mistakes that occur when solving real problems in floating-point arithmetic on a computer (and to noise that would normally pollute the vector $\mathbf{b}$). Unfortunately, many important, practical problems have this same structure. As a result, much research has gone into effective ways of tweaking this $\mathbf{Ax} = \mathbf{b}$ problem to discover a more robust solution. Later in the semester, we will learn about *regularization*. Applying this technique, we arrive at a much more satisfactory estimate of $\mathbf{x}$, shown in blue below. (The gray line underneath show the true $\mathbf{x}$ we are trying to find.)

Modern computers use a "floating point number system" for calculations that involve real numbers. The computer cannot represent all the (uncountably many) real numbers; instead it uses a carefully selected finite subset of the reals. Most of the time, this number system allows us to compute quickly and accurately (with small mistakes on the order of 10^{-16}, which we can ignore). However, when a problem is sensitive to small changes in the input data, you will get bad errors regardless of the cleverness of your number system.



This answer is not perfect, but to read the barcode we only need to know if our function is closer to 0 (white) or 1 (black) at a given pixel. The plot above is good enough to serve our purposes.

This example suggests that $\mathbf{Ax} = \mathbf{b}$ is much more interesting than you might have thought when you encountered this equation in your first linear algebra class. Throughout CMDA 3606 we will try to convince you with additional examples and applications. By the semester's end, you will:

- learn several important origins of $\mathbf{Ax} = \mathbf{b}$ problems;
- appreciate when $\mathbf{Ax} = \mathbf{b}$ can be solved;
- efficiently solve $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is very large;
- when $\mathbf{Ax} = \mathbf{b}$ cannot be solved, understand the approximation

  $$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|,$$

  which is often associated with *inverse problems*;
- use this approximation problem (with *regularization*) to solve inverse problems from applications, such as image deblurring;
- master the singular value decomposition (SVD);

- apply the SVD for data analytics, including principal component analysis and recommender systems;
- solve optimization problems like

$$\min_{\mathbf{x}} f(\mathbf{x}),$$

  where f depends on a vector of variables, using NEWTON's method (which requires the solution of an $\mathbf{Ax} = \mathbf{b}$ problem at each step).

Where possible, methods will be derived rigorously, but algorithms and applications will be a constant theme. Through this course (and CMDA 3605 before it), CMDA 3606 students will acquire a significant toolkit for solving a variety of applied problems in mathematical modeling and data science. *This course teaches empowering technology.*

1.1 Prerequisites

Upon starting CMDA 3606, all students are expected to have a basic working knowledge of:

- basic matrix-vector operations;
- subspaces, especially the column space and null space of a matrix;
- Gaussian elimination for solving $\mathbf{Ax} = \mathbf{b}$;
- eigenvalues and eigenvectors;
- MATLAB.

These concepts will be reviewed just-in-time, as need for them arises throughout the semester.

Students in the class are not expected to be expert MATLAB users, but should be able to write short programs on their own. MATLAB provides the best environment for experimenting with the concepts we will discuss throughout the semester. The instructor will provide sample codes from in-class demonstrations.

Python, with its NumPy package, is becoming an increasingly popular alternative to MATLAB.

1.2 Some notation and basic matrix-vector operations

We start modestly, establishing our basic conventions. Notation is the unsung hero of mathematics. The right notation clarifies, helping you see the essence of a problem. The conventions we describe have gradually evolved over linear algebra's 150 year history.

A *vector* is a column of numbers. We write $\mathbf{v} \in \mathbb{R}^n$ to indicate that $\mathbf{v}$ is a vector of length n whose entries are *real* numbers. If the entries

in $\mathbf{v}$ are *complex* numbers, we instead write $\mathbf{v} \in \mathbb{C}^n$. (In this class, most of our vectors will only contain real numbers.) In either case, we express $\mathbf{v}$ in terms of its entries:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

Vectors are typeset as bold Roman characters, while their entries (like all our scalars) are italic Roman (or Greek) letters.

Often you will need to turn a column vector into a row vector with the help of the *transpose*:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}.$$

Matrices are rectangular arrays of numbers. We write $\mathbf{A} \in \mathbb{R}^{m \times n}$ for a matrix with m rows and n columns made up of real entries; when those entries could be complex, we write $\mathbf{A} \in \mathbb{C}^{m \times n}$. In either case, we express $\mathbf{A}$ in terms of its entries:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}.$$

Matrices are denoted by bold capital letters, either Roman or Greek.

As you become a nimble manipulator of matrices, you will find it convenient to organize these entries in different ways. For example, you might write $\mathbf{A}$ by columns as

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix},$$

$$\mathbf{A} = \left[\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array}\right] \in \mathbb{R}^{m \times n}$$

where each $\mathbf{a}_k$ is a vector of length m:

$$\mathbf{a}_k = \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{m,k} \end{bmatrix}.$$

This bird's-eye view of $\mathbf{A}$ gives you a deeper appreciation for matrix-vector multiplication, for

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_{k=1}^{n} x_k \mathbf{a}_k = \sum_{k=1}^{n} x_k \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{m,k} \end{bmatrix},$$

revealing $\mathbf{A}\mathbf{x}$ to be a weighted sum of the columns of $\mathbf{A}$: the entry x_k reveals how much the vector $\mathbf{a}_k$ contributes to the product $\mathbf{A}\mathbf{x}$.

You can also build matrices from other matrices, as in

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

(assuming the dimensions match up properly). Then you can multiply against a conformally partitioned vector:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \\ \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{y} \end{bmatrix}.$$

Such composite matrices arise naturally in constrained multivariable optimization problems and in models of incompressible fluid dynamics.

The underlying mathematical model often suggests a natural way to partition a matrix like this.

Several distinguished matrices merit special mention. The *zero matrix* has all entries set to zero; we write it as $\mathbf{0}$, and its dimension will be clear from the context. The *identity* is a square matrix with zeros everywhere except the main diagonal, whose entries are all ones; we denote it by $\mathbf{I}$. We shall occasionally pick out kth column of the identity, which we denote by $\mathbf{e}_k$. Thus the $n \times n$ identity is

$$\mathbf{I} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \end{bmatrix}.$$

For example, when $n = 4$,

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{1.1}$$

Notice that you can use $\mathbf{e}_k$ to extract the kth column from $\mathbf{A}$:

$$\mathbf{A}\mathbf{e}_k = \mathbf{a}_k.$$

Notice that, for all n,

$$\mathbf{I}\mathbf{x} = \mathbf{x}, \qquad \mathbf{I}\mathbf{A} = \mathbf{A}, \qquad \mathbf{A}\mathbf{I} = \mathbf{A},$$

for all vectors $\mathbf{x}$ and matrices $\mathbf{A}$. More generally, we construct *diagonal* matrices as

$$\operatorname{diag}(d_1, \ldots, d_n) = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix},$$

where the unspecified off-diagonal elements are zero.

A square matrix $\mathbf{B}$ for which $\mathbf{A}\mathbf{B} = \mathbf{I}$ is called the *inverse* of $\mathbf{A}$, and is denoted by $\mathbf{A}^{-1}$. Not all matrices are invertible; for example, $\mathbf{A} = \mathbf{0}$ has no inverse. When the inverse exists, it is unique. It works on the right and the left sides of $\mathbf{A}$:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

These important facts are proved in most linear algebra books/courses. We shall not go into details here.

The inverse of a product of matrices is the product of the individual inverses, in reverse order. For example, if both $\mathbf{A}$ and $\mathbf{B}$ are invertible, then

$$(\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}.$$

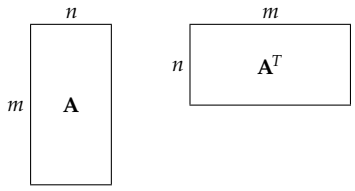
This fact is easily verified, since

$$(\mathbf{AB})(\mathbf{B}^{-1}\mathbf{A}^{-1}) = \mathbf{ABB}^{-1}\mathbf{A}^{-1} = \mathbf{AIA}^{-1} = \mathbf{AA}^{-1} = \mathbf{I}.$$

Hence $\mathbf{B}^{-1}\mathbf{A}^{-1}$ does what an inverse of $\mathbf{AB}$ is supposed to do, and since the inverse is unique, it is the only matrix that so affects $\mathbf{AB}$.

Just as we took the transpose of vectors, we do similarly for matrices:

$$\mathbf{A}^T = \begin{bmatrix} a_{1,1} & \cdots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \cdots & a_{m,n} \end{bmatrix}.$$

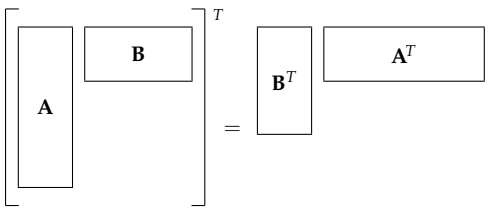


When $\mathbf{A}^T = \mathbf{A}$ we say $\mathbf{A}$ is *symmetric*. (Notice that only square matrices can be symmetric.)

The transpose distributes across addition, and (just like the inverse) distributes across a product, but reverses the order:

$$(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T, \qquad (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T.$$

You can recursively apply this idea to handle longer strings of matrices: $(\mathbf{ABCD})^T = \mathbf{D}^T\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$.



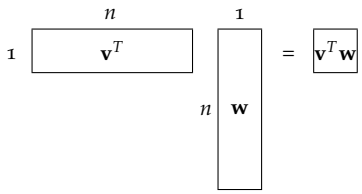
1.3 *Inner and outer products, geometry*

We often multiply vectors together in two special ways.

Definition 1 *The* inner product *(or* dot product*) of* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *is the scalar*

$$\mathbf{v}^T\mathbf{w} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \sum_{j=1}^{n} v_j w_j \in \mathbb{R}.$$

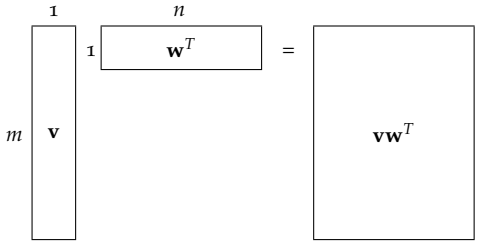
The *inner product* $\mathbf{v}^T\mathbf{w}$ can be viewed as the product of a $1 \times n$ matrix with an $n \times 1$ matrix, giving a 1×1 result.



Definition 2 *The* outer product *of* $\mathbf{v} \in \mathbb{R}^m$ *and* $\mathbf{w} \in \mathbb{R}^n$ *is the* $m \times n$ *matrix*

$$\mathbf{vw}^T = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} = \begin{bmatrix} v_1 w_1 & \cdots & v_1 w_n \\ \vdots & \ddots & \vdots \\ v_m w_1 & \cdots & v_m w_n \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

The *outer product* $\mathbf{vw}^T$ is the product of an $m \times 1$ matrix with a $1 \times n$ matrix, giving an $m \times n$ result.



We have casually spoken of the "length" of a vector, meaning its number of components (or *dimension*). However, there is another notion of the size of a vector, which generalizes the absolute value.

Definition 3 *The* norm *of a vector* $\mathbf{v} \in \mathbb{R}^n$ *is denoted*

$$\|\mathbf{v}\| = \sqrt{\sum_{j=1}^{n} |v_j|^2} = \sqrt{\mathbf{v}^T\mathbf{v}}.$$

The norm of a real vector is the usual Euclidean length that is familiar from geometry and physics.

The inner product also gives rise to another geometric notion, the angle between two vectors.

Definition 4 *The* angle *between the nonzero vectors* $\mathbf{v}$ *and* $\mathbf{w} \in \mathbb{R}^n$ *is*

$$\angle(\mathbf{v}, \mathbf{w}) = \cos^{-1}\left(\frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|\,\|\mathbf{w}\|}\right). \tag{1.2}$$

Definition 5 *Two vectors* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *are* orthogonal *provided*

$$\mathbf{v}^T\mathbf{w} = 0.$$

If $\mathbf{v}$ and $\mathbf{w}$ are nonzero, then orthogonality is equivalent to $\angle(\mathbf{v}, \mathbf{w}) = \pi/2$, so orthogonality implies that two vectors form a right angle.

Often the orthogonality of $\mathbf{v}$ and $\mathbf{w}$ is expressed as

$$\mathbf{v} \perp \mathbf{w}.$$

In this setting, the Pythagorean Theorem becomes quite simple.

Theorem 1 (Pythagoras) *If* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *are orthogonal, then*

$$\|\mathbf{v} + \mathbf{w}\|^2 = \|\mathbf{v}\|^2 + \|\mathbf{w}\|^2.$$

Proof. The orthogonality of $\mathbf{v}$ and $\mathbf{w}$ implies $\mathbf{v}^T\mathbf{w} = \mathbf{w}^T\mathbf{v} = 0$, so

$$\begin{aligned}\|\mathbf{v} + \mathbf{w}\|^2 &= (\mathbf{v} + \mathbf{w})^T(\mathbf{v} + \mathbf{w}) \\ &= \mathbf{v}^T\mathbf{v} + \mathbf{v}^T\mathbf{w} + \mathbf{w}^T\mathbf{v} + \mathbf{w}^T\mathbf{w} \\ &= \mathbf{v}^T\mathbf{v} + \mathbf{w}^T\mathbf{w} \\ &= \|\mathbf{v}\|^2 + \|\mathbf{w}\|^2. \qquad \blacksquare\end{aligned}$$

We shall make use of this ancient theorem at several key points in the semester. Another key result, which we shall not prove here, is the Cauchy–Schwarz inequality.

Theorem 2 (Cauchy–Schwarz Inequality) *For any* $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$,

$$|\mathbf{v}^T\mathbf{w}| \le \|\mathbf{v}\|\,\|\mathbf{w}\|.$$

Simple though it may seem, the Cauchy–Schwarz inequality is powerful and widely applicable. Its diverse proofs point to many directions in mathematics, forming the subject of an entire book:

J. Michael Steele. *The Cauchy–Schwarz Master Class*. Cambridge University Press, Cambridge, 2004

This theorem is trivial when $\mathbf{v}$ is orthogonal to $\mathbf{w}$, and it is sharp (i.e., it holds with equality) only when $\mathbf{v}$ and $\mathbf{w}$ are collinear. Notice that the Cauchy–Schwarz inequality ensures that

$$\frac{|\mathbf{v}^T\mathbf{w}|}{\|\mathbf{v}\|\,\|\mathbf{w}\|} \le 1,$$

and hence the argument of the arc cosine in (1.2) is permissible, making our definition of the *angle* between two vectors reasonable.