

LINEAR ALGEBRA

An Introduction to Data Science

Paul A. Jensen
University of Illinois at Urbana-Champaign

Contents

Introduction 7

0.1 Notation 7

1 Fields and Vectors 11

1.1 Algebra 11

1.2 The Field Axioms 11

1.2.1 Common Fields in Mathematics 13

1.3 Vector Addition 14

1.4 Vector Multiplication is not Elementwise 15

1.4.1 Do We Need Multiplication? 15

1.5 Linear Systems 15

1.6 Vector Norms 16

1.6.1 Normalized (Unit) Vectors 18

1.7 Scalar Vector Multiplication 18

1.8 Inner (Dot) Products 19

1.8.1 Computing the Dot Product 19

1.8.2 Dot Product Summary 21

2 Matrices 23

2.1 Matrix Multiplication 23

2.1.1 Generalized Multiplication 24

| | | |
|-------|---|----|
| 2.2 | <i>Identity Matrix</i> | 25 |
| 2.3 | <i>Matrix Transpose</i> | 25 |
| 2.4 | <i>Solving Linear Systems</i> | 26 |
| 2.5 | <i>Gaussian Elimination</i> | 29 |
| 2.6 | <i>Computational Complexity of Gaussian Elimination</i> | 30 |
| 2.7 | <i>Solving Linear Systems in MATLAB</i> | 31 |
| 3 | <i>The Finite Difference Method</i> | 33 |
| 3.1 | <i>Finite Differences</i> | 33 |
| 3.2 | <i>Linear Differential Equations</i> | 34 |
| 3.3 | <i>Discretizing a Linear Differential Equation</i> | 35 |
| 3.4 | <i>Boundary Conditions</i> | 36 |
| 4 | <i>Inverses, Solvability, and Rank</i> | 39 |
| 4.1 | <i>Matrix Inverses</i> | 39 |
| 4.2 | <i>Elementary Matrices</i> | 40 |
| 4.3 | <i>Proof of Existence for the Matrix Inverse</i> | 40 |
| 4.4 | <i>Computing the Matrix Inverse</i> | 42 |
| 4.5 | <i>Numerical Issues</i> | 43 |
| 4.6 | <i>Inverses of Elementary Matrices</i> | 45 |
| 4.7 | <i>Rank</i> | 45 |
| 4.8 | <i>Rank and Matrix Inverses</i> | 50 |
| 4.9 | <i>Summary</i> | 51 |
| 5 | <i>Linear Models and Regression</i> | 53 |
| 6 | <i>Applied Linear Regression in Matlab</i> | 55 |
| 7 | <i>Optimization, Convexity, and Hyperplanes</i> | 57 |
| 7.1 | <i>Optimization</i> | 57 |
| 7.1.1 | <i>Unconstrained Optimization</i> | 58 |
| 7.1.2 | <i>Constrained Optimization</i> | 58 |

| | | |
|-------|---|----|
| 7.2 | <i>Convexity</i> | 59 |
| 7.2.1 | <i>Convex sets</i> | 59 |
| 7.2.2 | <i>Convex functions</i> | 60 |
| 7.2.3 | <i>Convexity in Optimization</i> | 60 |
| 7.2.4 | <i>Convexity of Linear Systems</i> | 61 |
| 7.3 | <i>Geometry of Linear Equations</i> | 62 |
| 7.4 | <i>Geometry of Linear Systems</i> | 64 |
| 8 | <i>Vector Spaces, Span, and Basis</i> | 65 |
| 8.1 | <i>Vector Spaces</i> | 65 |
| 8.2 | <i>Span</i> | 65 |
| 8.3 | <i>Review: Linear Independence</i> | 67 |
| 8.4 | <i>Basis</i> | 67 |
| 8.4.1 | <i>Testing if vectors form a basis</i> | 68 |
| 8.4.2 | <i>Decomposing onto a basis</i> | 68 |
| 8.5 | <i>Orthogonal and Orthonormal Vectors</i> | 69 |
| 8.5.1 | <i>Decomposing onto orthonormal vectors</i> | 70 |
| 8.5.2 | <i>Checking an orthonormal set</i> | 71 |
| 8.5.3 | <i>Projections</i> | 71 |
| 8.5.4 | <i>Creating orthonormal basis vectors</i> | 72 |
| 9 | <i>Eigenvalues and Eigenvectors</i> | 75 |
| 9.1 | <i>Properties of Eigenvectors and Eigenvalues</i> | 76 |
| 9.2 | <i>Computing Eigenvectors and Eigenvalues</i> | 77 |
| 9.2.1 | <i>Eigenvalues and Eigenvectors in MATLAB</i> | 78 |
| 9.3 | <i>Applications</i> | 79 |
| 9.3.1 | <i>Solving Systems of ODEs</i> | 79 |
| 9.3.2 | <i>Stability of Linear ODEs</i> | 80 |
| 9.3.3 | <i>Positive Definite Matrices</i> | 80 |

Introduction

Updated 1/15/2017.

This class has three parts. In Part I, we analyze linear systems that transform an n -dimensional vector (\mathbf{x}) into another n -dimensional vector (\mathbf{y}). This transformation is often expressed as a linear system via matrix multiplication: $\mathbf{y} = \mathbf{A}\mathbf{x}$. In Part II, we expand the type of systems we can solve, including systems that transform vectors from n dimensions to m dimensions. We will also consider solution strategies that use alternative objectives when the system contains either too much or not enough information. Finally, Part III dispenses with linear systems altogether, focusing purely on observations of sets of n -dimensional vectors (matrices). We will learn how to analyze and extract information from matrices without a clear input/output relationship.

0.1 Notation

We will distinguish scalars, vectors, and matrices with the following typographic conventions:

| Object | Font and Symbol | Examples |
|----------|-------------------------------|--|
| Scalars | italicized, lowercase letters | x, α, y |
| Vectors | bold lowercase letters | $\mathbf{x}, \mathbf{y}, \mathbf{n}, \mathbf{w}$ |
| Matrices | bold uppercase | $\mathbf{A}, \mathbf{A}^{-1}, \mathbf{B}, \mathbf{\Gamma}$ |

There are many ways to represent rows or columns of a matrix \mathbf{A} . Since this course uses MATLAB, we think it is convenient to use a matrix addressing scheme that reflects Matlab's syntax. So, the i th row of matrix \mathbf{A} will be $\mathbf{A}(i, :)$, and the j th column will be $\mathbf{A}(:, j)$. Rows or columns of a matrix are themselves vectors, so we choose to keep the boldface font for the matrix \mathbf{A} even when it is subscripted. We could also use Matlab syntax for vectors ($\mathbf{x}(i)$, for example). However, the form x_i is standard across many fields of mathematics and engineering, so we retain the common notation. The lack of boldface font reminds us that elements of vectors are scalars in the field.

One goal of this class is to increase the precision of your mathematical writing. We will regularly use the following symbols to describe mathematical concepts and relations.

| Symbol | Read As | Description |
|--------------------|---|--|
| \Rightarrow | implies | $p \Rightarrow q$ means that whenever p is true, q must also be true. |
| \Leftrightarrow | if and only if | A symmetric, stronger version of \Rightarrow . The expression $p \Leftrightarrow q$ means $p \Rightarrow q$ and $q \Rightarrow p$. |
| \forall | for all | Remember this symbol as an upside down “A”, as in “for A ll”. |
| \exists | there exists | Remember this symbol as a backwards “E”, as in “there E xists”. |
| \in (\notin) | is (not) a member of | Used to state that a single element is a member of a set, i.e. $1 \in \mathbb{Z}$. To say that a set of multiple elements is a subset of another set, use \subset . |
| s.t. | such that | Other texts use the symbol $ $ (a vertical pipe) instead of “s.t.”. Note that “s.t.” is set in normal, not italicized font. |
| \mathbb{R} | the real numbers | The numbers along a line. The reals include both rational and irrational numbers. |
| \mathbb{R}^n | the set of n -dimensional vectors of real numbers | Each value of n is a different set – if $r \in \mathbb{R}^2$, then $r \notin \mathbb{R}^3$. Also, \mathbb{R}^2 is not a subset of \mathbb{R}^3 , etc. |
| \mathbb{Z} | the integers | The integers contain the “natural numbers” (1, 2, 3, ...), their negatives (-1, -2, -3, ...), and the number zero (0). The symbol comes from the German word for “number” (Zahlen). The word “integer” (Latin for “whole”) is used since integers have no fractional part. |
| \mathbb{Q} | the rationals | The rational numbers are all numbers that are the quotient of two integers. The symbol derives from the word “quotient”. |
| \mapsto | maps to | Describes the inputs and outputs of an operation. An operation that maps a vector of reals to a real number is $\mathbb{R}^n \mapsto \mathbb{R}$. An operation that maps two integers to a rational is $\mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{Q}$. |

These symbols can be used to succinctly write mathematical statements. For example, we can formally define the set of rational numbers as

*A number is rational if and only if it can be expressed
as the quotient of two integers.*

with the statement

$$r \in \mathbb{Q} \Leftrightarrow \exists p, q \in \mathbb{Z} \text{ s.t. } r = p/q$$

While the latter statement is shorter, it is more difficult to understand. So whenever possible we recommend writing statements with as few symbols as necessary. Rely on mathematical symbols only when a textual definition would be unwieldy or imprecise, or when brevity is important (like when writing on a chalkboard).

1

Fields and Vectors

1.1 Algebra

Updated 1/15/2017.

Algebra is a branch of mathematics that contains symbols and a set of rules to manipulate them.

You are probably familiar with the idea of symbols. We call them variables, and in previous algebra courses you used them to represent unknown (real) numbers. In this course, we will use variables to represent vectors. Vectors are collections of elements (e.g. real numbers). The number of elements in a vector is its dimension. We can write an n -dimensional vector as

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

While some vectors have elements that are real numbers, vectors themselves are not numbers. An n -dimensional vector does not belong to the set \mathbb{R} of real numbers; it belongs to a special set \mathbb{R}^n of all other vectors of dimension n with real elements.

We use the rules of algebra to manipulate elements. However, only certain sets of elements are amenable to the rules of algebra. These algebra-compatible sets are called *fields*. A set of conditions, or *axioms*, must be true about a set before we can consider it a field. These field axioms define the rules of algebra.

After spending years studying algebra, you might think that there are many byzantine rules that govern fields. In fact, there are only five. The five axioms describe only two operations (addition and multiplication) and define two special elements that must be in every field (0 and 1).

1.2 The Field Axioms

Given elements a , b , and c in a field:

When we say vector, we assume a *column vector* – a vertical array of elements. A *row vector* is a horizontal array of elements. We will see that column vectors are more convenient.

We can surround the elements of a vector with either parentheses () or square brackets []. Using straight lines || is not allowed, as this has a special meaning.

1. **Associativity.**

$$a + b + c = (a + b) + c = a + (b + c)$$

$$abc = (ab)c = a(bc)$$

2. **Commutativity.**

$$a + b = b + a$$

$$ab = ba$$

3. **Identity.** There exist elements 0 and 1, both in the field, such that

$$a + 0 = a$$

$$1 \times a = a$$

4. **Inverses.**

- For all a , there exists an element $(-a)$ in the field such that $a + (-a) = 0$.
- For all $a \neq 0$, there exists an element (a^{-1}) in the field such that $a \times a^{-1} = 1$.

5. **Distribution** of multiplication over addition.

$$a(b + c) = ab + ac$$

It might surprise you that only five axioms are sufficient to recreate everything you know about algebra. For example, nowhere do we state the special property of zero that $a \times 0 = 0$ for any number a . We don't need to state this property, as it follows from the field axioms:

Theorem. $a \times 0 = 0$

Proof.

$$\begin{aligned} a \times 0 &= a \times (1 - 1) \\ &= a \times 1 + a \times (-1) \\ &= a - a \\ &= 0 \end{aligned}$$

□

Similarly, we can prove corollaries from the field axioms.

Corollary. *If $ab = 0$, then either $a = 0$ or $b = 0$ (or both).*

Proof. Suppose $a \neq 0$. Then there exists a^{-1} such that

$$a^{-1}ab = a^{-1} \times 0$$

$$1 \times b = 0$$

$$b = 0$$

A similar argument follows when $b \neq 0$. □

The fundamental theorem of algebra relies on the above corollary when solving polynomials. If we factor a polynomial into the form $(x - r_1)(x - r_2) \cdots (x - r_k) = 0$, then we know the polynomial has roots r_1, r_2, \dots, r_k . This is only true because the left hand side of the factored expression only reaches zero when one of the factors is zero, i.e. when $x = r_i$.

1.2.1 Common Fields in Mathematics

The advantage of fields is that once a set is proven to obey the five field axioms, we can operate on elements in the field just like we would operate on real numbers. Besides the real numbers (which the concept of fields was designed to emulate), what are some other fields?

The rational numbers are a field. The numbers 0 and 1 are rational, so they are in the field. Since we add and multiply rational numbers just as we do real numbers, these operations commute, associate, and distribute. All that remains is to show that the rationals have additive and multiplicative inverses in the field. Let us consider a rational number p/q , where p and q are integers.

- We know that $-p/q$ is also rational, since $-p$ is still an integer. The additive inverse of a rational number is in the field of rational numbers.
- The additive inverse of p/q is q/p , which is also rational. The multiplicative inverse of a rational is also in the field.

So the rational numbers are a field. What does this mean? If we are given an algebraic expression, we can solve it by performing any algebraic manipulation and still be assured that the answer will be another rational number.

The integers, by contrast, are not a field. Every integer has a reciprocal ($2 \rightarrow 1/2$, $-100 \rightarrow -1/100$, etc.). However, the reciprocals are themselves not integers, so they are not in the same field. The field axioms require that the inverses for every element are members of the field. When constructing a field, every part of every axiom must be satisfied.

Let's see an example of this. Imagine the simple equation $y = ax + b$, which we solve for x to yield

$$x = \frac{y - b}{a}$$

If we wanted to solve this equation using only rational numbers, we would not need to change anything. So long as the values we input for the variables a , b , and y were rational, the value of x would also be rational. We solved the equation using field algebra, and the rationals constitute a field. Everything works out.

Now imagine you wanted only integer solutions. Even if the values for a , b , and y were integers, there is no guarantee that x would be an integer. ($a = 2$, $b = 4$, and $y = 3$ yields $x = -1/2$, for example). Because the integers are not a field, algebra does not work on them. In particular, the integers do not have integer multiplicative inverses (except for 1 and -1). When we divide by a , we assumed that the value $1/a$ exists in the field, which it does not.

Interestingly, the integers always have integer additive inverses, so the solution to the equation $y = x - b$ is always an integer (for integer y and b) since we could solve the equation with only additive inverses.

1.3 Vector Addition

Addition of two vectors is defined *elementwise*, or element-by-element.

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}$$

Since this is a direct extension of scalar addition, it is clear that vector addition commutes $[\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}]$ and is associative $[\mathbf{x} + \mathbf{y} + \mathbf{z} = (\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})]$.

The additive inverse of a vector \mathbf{x} (written as $-\mathbf{x}$) is also constructed elementwise:

$$-\mathbf{x} = \begin{pmatrix} -x_1 \\ -x_2 \\ \vdots \\ -x_n \end{pmatrix}$$

From our elementwise definition of vector addition, we can construct the zero element for the vector space. We know from the field axioms that $\mathbf{x} + \mathbf{0} = \mathbf{x}$, so the zero element must be a vector of the same dimension with all zero entries.

$$\mathbf{x} + \mathbf{0} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} x_1 + 0 \\ x_2 + 0 \\ \vdots \\ x_n + 0 \end{pmatrix} = \mathbf{x}$$

Notice that each set of n -dimensional vectors has its own zero element. In \mathbb{R}^2 , $\mathbf{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. In \mathbb{R}^3 , $\mathbf{0} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

1.4 Vector Multiplication is not Elementwise

What happens when we try to define multiplication as an elementwise operation? For example

$$\begin{pmatrix} -1 \\ 0 \\ 4 \end{pmatrix} \times \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \times 0 \\ 0 \times 2 \\ 4 \times 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \mathbf{0}$$

This is bad. Very bad. Here we have an example where $\mathbf{xy} = \mathbf{0}$, but neither \mathbf{x} nor \mathbf{y} is the zero element $\mathbf{0}$. This is a direct violation of a corollary of the field axioms, so **elementwise vector multiplication is not a valid algebraic operation**.

Sadly, vectors are not a field. There is no way to define multiplication using only vectors that satisfies the field axioms. Nor is there anything close to a complete set of multiplicative inverses, or even the element $\mathbf{1}$. Instead, we will settle for a weaker result – showing that vectors live in a *normed inner product space*. The concepts of a vector norm and inner product will let us create most of the operations and elements that vectors need to be a field.

On the bright side, if vectors were a field this class would be far too short.

1.4.1 Do We Need Multiplication?

When you were first taught to multiply, it was probably introduced as a “faster” method of addition, i.e. $4 \times 3 = 3 + 3 + 3 + 3$. If so, why do we need multiplication as a separate requirement for fields? Couldn’t we simply require the addition operator and construct multiplication from it? The answer is no, for two reasons. First, the idea of multiplication as a shortcut for addition only makes sense when discussing the non-negative integers. What, for example, does it mean to have -2.86×3 ? What does -2.86 groups look like in terms of addition?

Second, we must realize that multiplication is a much stronger relationship between numbers. To understand why, we should start talking about the “linear” part of linear algebra.

Also, the integers are not a field!

1.5 Linear Systems

Linear systems have two special properties.

1. **Proportionality.** If the input to a linear system is multiplied by a scalar, the output is multiplied by the same scalar: $f(kx) = kf(x)$.
2. **Additivity.** If two inputs are added, the result is the sum of the original outputs: $f(x_1 + x_2) = f(x_1) + f(x_2)$.

We can combine both of these properties into a single condition for linearity.

Definition. A system f is linear if and only if

$$f(k_1x_1 + k_2x_2) = k_1f(x_1) + k_2f(x_2)$$

for all inputs x_1 and x_2 and scalars k_1 and k_2 .

Consider a very simple function, $f(x) = x + 3$. Is this function linear? First we calculate the lefthand side of the definition of linearity.

$$f(k_1x_1 + k_2x_2) = k_1x_1 + k_2x_2 + 3$$

We compare this to the righthand side.

$$\begin{aligned} k_1f(x_1) + k_2f(x_2) &= k_1(x_1 + 3) + k_2(x_2 + 3) \\ &= k_1x_1 + k_2x_2 + 3(k_1 + k_2) \\ &\neq f(k_1x_1 + k_2x_2) \end{aligned}$$

This does not follow the definition of linearity. The function $f(x) = x + 3$ is not linear. Now let's look at a simple function involving multiplication: $f(x) = 3x$. Is this function linear?

$$\begin{aligned} f(k_1x_1 + k_2x_2) &= 3(k_1x_1 + k_2x_2) \\ &= k_1(3x_1) + k_2(3x_2) \\ &= k_1f(x_1) + k_2f(x_2) \end{aligned}$$

The function involving multiplication is linear.

These results might not be what you expected, at least concerning the nonlinearity of functions of the form $f(x) = x + b$. This is probably because in earlier math courses you referred to equations of straight lines ($y = mx + b$) as linear equations. In fact, any equation of this form (with $b \neq 0$) is called *affine*, not linear.

Truly linear functions have the property that $f(0) = 0$. Addition is, in a way, not “strong” enough to drive a function to zero. The expression $x + y$ is zero only when both x and y are zero. By contrast, the product xy is zero when either x or y is zero.

This follows from proportionality. If $f(k0) = kf(0)$ for all k , then $f(0)$ must equal zero.

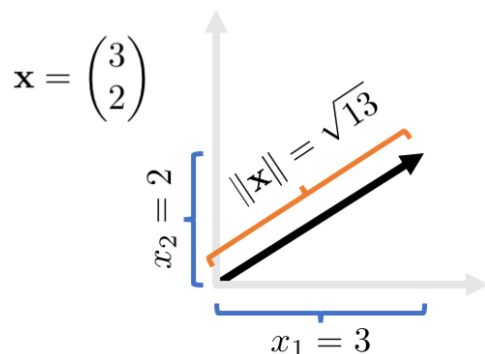
1.6 Vector Norms

One of the nice properties of the real numbers is that they are well ordered. Being well ordered means that for any two real numbers, we can determine which number is larger (or if the two numbers are equal). Well orderedness allows us to make all sorts of comparisons between the real numbers.

Vectors are not well ordered. Consider the vectors $(3, 4)$ and $(5, 2)$. Which one is larger? Each vector has one element that is larger than the other (4 in the first, 5 in the second). There is no unambiguous way to place all vectors in order.

This doesn't stop us from making comparisons between vector quantities. Consider velocity, which, contrary to how many people use the term, is a vector. Since vectors are not ordered, we should not be able to compare velocities. Instead, we often compare speeds, which are the magnitude of the velocity vectors. Traveling 30 mph due north and 30 mph due east are technically two different velocities. However, they have the same magnitude (30 mph), so most people consider them equivalent.

Vector magnitudes are calculated by taking a *norm* of the vector. There are many different kinds of norms, but the most commonly used norm is the 2-norm (or Cartesian or Pythagorean norm). We will refer to the 2-norm as simply “the norm” unless we state otherwise. If we treat a vector as a point in n -dimensional space, the norm is the length of the arrow drawn from the origin to that point.



In 2D, the norm corresponds to the hypotenuse of the right triangle with sides equivalent to the two elements in the vector – hence the name “Pythagorean norm” (since the norm can be calculated by the Pythagorean theorem). In higher dimensions, we simply generalize the Pythagorean definition of the norm to

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

In one dimension, taking the 2-norm yields the same result as taking the absolute value:

$$\|-3\| = \sqrt{(-3)^2} = 3 = |-3|$$

There are two useful properties of norms that derive directly from its definition. These properties must be true of all norms, not just the 2-norm.

1. **Nonnegativity.** $\|\mathbf{x}\| \geq 0$
2. **Zero Identity.** $\|\mathbf{x}\| = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$

We use a pair of two vertical bars ($\|\cdot\|$) to represent the norm. This differentiates the norm from the absolute value (which is, in fact, the 1-norm). Some texts use a subscript to identify which norm we are taking, i.e. $\|\mathbf{x}\|_2$ is the 2-norm of \mathbf{x} .

Be careful not to confuse the 2-norm and the absolute value, as they are not the same thing. Their equivalence in one dimension is a coincidence. However, the absolute value is a norm – it returns the magnitude of a number, but strips away the direction (negative or positive).

1.6.1 Normalized (Unit) Vectors

A vector contains information about both its magnitude and its orientation. We've seen how to extract the magnitude as a scalar from the vector by taking the norm. Is it possible to similarly separate out the vector's orientation? Yes, by *normalizing* the vector. A normalized vector (or unit vector) is any vector with magnitude equal to one. We can convert a vector to a normalized vector by dividing each element by the vector's magnitude. For example

$$\mathbf{x} = \begin{pmatrix} 3 \\ -4 \end{pmatrix} \Rightarrow \|\mathbf{x}\| = \sqrt{3^2 + (-4)^2} = 5$$

The normalized unit vector ($\hat{\mathbf{x}}$) is

$$\hat{\mathbf{x}} = \begin{pmatrix} 3/\|\mathbf{x}\| \\ -4/\|\mathbf{x}\| \end{pmatrix} = \begin{pmatrix} 3/5 \\ -4/5 \end{pmatrix}$$

Intuitively, the idea of a normalized unit vector as a direction makes sense. If a vector is a product of both a magnitude and a direction, then the vector divided by the magnitude (the norm) should equal a direction (a unit vector).

We use the hat symbol ($\hat{}$) over a unit vector to remind us that it has been normalized.

1.7 Scalar Vector Multiplication

We saw earlier that elementwise multiplication was a terrible idea. In fact, defining multiplication this way violates a corollary of the field axioms ($\mathbf{xy} = \mathbf{0}$ implies that $\mathbf{x} = \mathbf{0}$ or $\mathbf{y} = \mathbf{0}$). However, elementwise multiplication does work in one case – *scalar multiplication*, or the product between a scalar (real number) and a vector:

$$k\mathbf{x} = k \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} kx_1 \\ kx_2 \\ \vdots \\ kx_n \end{pmatrix}$$

where k is a scalar real number. Notice that scalar multiplication does not suffer from the same problem as elementwise vector multiplication. If $k\mathbf{x} = \mathbf{0}$, then either the scalar k equals zero or the vector \mathbf{x} must be the zero vector.

What happens when you multiply a vector by a scalar? For one, the norm changes:

$$\begin{aligned} \|k\mathbf{x}\| &= \sqrt{(kx_1)^2 + (kx_2)^2 + \cdots + (kx_n)^2} \\ &= \sqrt{k^2(x_1^2 + x_2^2 + \cdots + x_n^2)} \\ &= |k| \|\mathbf{x}\| \end{aligned}$$

Remember that $\sqrt{k^2} = |k|$, not k itself. We consider the square root to be the positive root.

Scalar multiplication scales the length of a vector by the scalar. If the scalar is negative, the direction of the vector “reverses”.

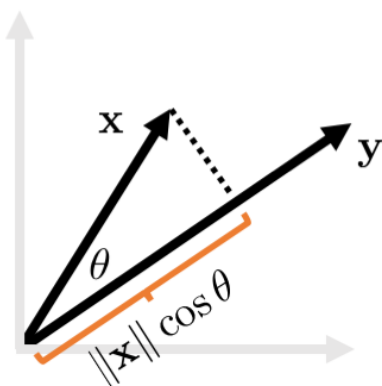
1.8 Inner (Dot) Products

One way to think of the product of two vectors is to consider the product of their norms (magnitudes). Such operations are common in mechanics. Work, for example, is the product of force and displacement. However, simply multiplying the magnitude of the force vector and the magnitude of the displacement vector disregards the orientation of the vectors. We know from physics that only the component of the force aligned with the displacement should count.

In general, we want an operation that multiplies the magnitude of one vector with the *projection* of a second vector onto the first. We call this operation the *inner product* or the *dot product*. Geometrically, the dot product is a measure of both the product of the vectors' magnitudes and how well they are aligned. For vectors \mathbf{x} and \mathbf{y} the dot product is defined

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$

where θ is the angle between the vectors.



Now we see why we use the symbol \times for multiplication; the dot (\cdot) is reserved for the dot product.

If two vectors are perfectly aligned, $\theta = 0^\circ$ and the dot product is simply the product of the magnitudes. If the two vectors point in exactly opposite directions, $\theta = 180^\circ$ and the dot product is -1 times the product of the magnitudes. If the vectors are *orthogonal*, the angle between them is 90° , so $\cos \theta = 0$ and the dot product is zero. Thus, **the dot product of two vectors is zero if and only if the vectors are orthogonal.**

1.8.1 Computing the Dot Product

We know how to calculate norms, but how do we calculate the angle between two n -dimensional vectors? The answer is that we don't need to. There is an easier way to calculate $\mathbf{x} \cdot \mathbf{y}$ than the formula $\|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$.

First, we need to define a special set of vectors – the unit vectors $\hat{\mathbf{e}}_i$. These are vectors that have only a single nonzero entry, a 1 at

element i . For example,

$$\hat{\mathbf{e}}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \hat{\mathbf{e}}_2 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \hat{\mathbf{e}}_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

Every vector can be written as a sum of scalar product with unit vectors. For example,

$$\begin{pmatrix} -3 \\ 6 \\ 2 \end{pmatrix} = -3 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 6 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\ = -3\hat{\mathbf{e}}_1 + 6\hat{\mathbf{e}}_2 + 2\hat{\mathbf{e}}_3$$

In general

$$\mathbf{x} = x_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + x_n \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \\ = \sum_{i=1}^n x_i \hat{\mathbf{e}}_i$$

Now let's compute the dot product using the unit vector expansion for \mathbf{x} and \mathbf{y} .

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y} &= (x_1 \hat{\mathbf{e}}_1 + x_2 \hat{\mathbf{e}}_2 + \cdots + x_n \hat{\mathbf{e}}_n) \cdot (y_1 \hat{\mathbf{e}}_1 + y_2 \hat{\mathbf{e}}_2 + \cdots + y_n \hat{\mathbf{e}}_n) \\ &= x_1 \hat{\mathbf{e}}_1 \cdot (y_1 \hat{\mathbf{e}}_1 + y_2 \hat{\mathbf{e}}_2 + \cdots + y_n \hat{\mathbf{e}}_n) \\ &\quad + x_2 \hat{\mathbf{e}}_2 \cdot (y_1 \hat{\mathbf{e}}_1 + y_2 \hat{\mathbf{e}}_2 + \cdots + y_n \hat{\mathbf{e}}_n) \\ &\quad + \cdots \\ &\quad + x_n \hat{\mathbf{e}}_n \cdot (y_1 \hat{\mathbf{e}}_1 + y_2 \hat{\mathbf{e}}_2 + \cdots + y_n \hat{\mathbf{e}}_n) \end{aligned}$$

Consider each of the terms $x_i \hat{\mathbf{e}}_i \cdot (y_1 \hat{\mathbf{e}}_1 + y_2 \hat{\mathbf{e}}_2 + \cdots + y_n \hat{\mathbf{e}}_n)$. By distribution, this is equivalent to

$$x_i y_1 \hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_1 + \cdots + x_i y_j \hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_j + \cdots + x_i y_n \hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_n$$

The only nonzero term in this entire summation is $x_i y_i \hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_i$, which equals $x_i y_i$. The dot product reduces to

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y} &= x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \\ &= \sum_{i=1}^n x_i y_i \end{aligned}$$

Although the above formula is convenient for computing dot products, it lacks the intuition of our previous method ($\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$). Whenever you use the latter method, be sure to remember that you're really calculating the product magnitudes after one vector is projected onto the other.

Think about the dot product $\hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_j$. If $i = j$, this product is 1 since $\|\hat{\mathbf{e}}_i\| = \|\hat{\mathbf{e}}_j\| = 1$ and $\theta = 0^\circ$. However, if $i \neq j$, the vectors are always orthogonal and the dot product is 0.

1.8.2 Dot Product Summary

- Dot products are defined between two vectors with the same dimension.
- Dot products return a scalar from two vectors. This is the projected product of the two magnitudes.
- $\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$
- $\mathbf{x} \cdot \mathbf{y} = 0 \Leftrightarrow \mathbf{x}$ and \mathbf{y} are orthogonal.
- Dot products are associative $[\mathbf{x} \cdot \mathbf{y} \cdot \mathbf{z} = (\mathbf{x} \cdot \mathbf{y}) \cdot \mathbf{z} = \mathbf{x} \cdot (\mathbf{y} \cdot \mathbf{z})]$, commutative $[\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}]$, and distributive over addition $[\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}]$.

2

Matrices

Updated 1/31/2018.

2.1 Matrix Multiplication

Let's take stock of the operations we've defined so far.

- The **norm** (magnitude) maps a vector to a scalar. ($\mathbb{R}^n \mapsto \mathbb{R}$)
- The **scalar product** maps a scalar and a vector to a new vector ($\mathbb{R} \times \mathbb{R}^n \mapsto \mathbb{R}^n$), but can only scale the magnitude of the vector (or flip it if the scalar is negative).
- The **dot product** maps two vectors to a scalar ($\mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$) by projecting one onto the other and multiplying the resulting magnitudes.

All of these operations appeared consistent with the field axioms. Unfortunately, we still do not have a true multiplication operation – one that can transform any vector into any other vector. Can we construct such an operation using only the above methods?

Let's construct a new vector \mathbf{y} from vector \mathbf{x} . To be as general as possible, we should let each element in \mathbf{y} be an arbitrary linear combination of the elements in \mathbf{x} . This implies that

$$\begin{aligned}y_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\y_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\&\vdots \\y_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n\end{aligned}$$

where the scalars a_{ij} determine the relative weight of x_j when constructing y_i . There are n^2 scalars required to unambiguously map \mathbf{x} to \mathbf{y} . For convenience, we collect the set of weights into an n by n numeric grid called a *matrix*.

If \mathbf{A} is a real-valued matrix with dimensions $m \times n$, we say $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\dim(\mathbf{A}) = m \times n$.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

What we have been calling “vectors” all along are really just matrices with only one column. Thinking of vectors as matrices lets us write a simple, yet powerful, definition of multiplication.

Definition. The product of matrices \mathbf{AB} is a matrix \mathbf{C} where each element c_{ij} in \mathbf{C} is the dot product between the i th row in \mathbf{A} and the j th column in \mathbf{B} :

$$c_{ij} = \mathbf{A}(i,:) \cdot \mathbf{B}(:,j)$$

Using this definition of matrix multiplication, the previous system of n equations becomes the matrix equation

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

or, more succinctly

$$\mathbf{y} = \mathbf{Ax}$$

2.1.1 Generalized Multiplication

In the previous example, both \mathbf{x} and \mathbf{y} were n -dimensional. This does not need to be the case. In general, the vector \mathbf{y} could have $m \neq n$ dimensions. The matrix \mathbf{A} would have m rows, each used to construct an element y_i in \mathbf{y} . However, the matrix \mathbf{A} would still need n columns to match the n rows in \mathbf{x} . (Each row in \mathbf{A} is “dotted” with the n -dimensional vector \mathbf{x} , and dot products require the two vectors have the same dimension.)

Any matrices \mathbf{A} and \mathbf{B} are *conformable* for multiplication if the number of columns in \mathbf{A} matches the number of rows in \mathbf{B} . If the dimensions of \mathbf{A} are $m \times n$ and the dimensions of \mathbf{B} are $n \times p$, then the product will be a matrix of dimensions $m \times p$.

Matrix multiplication is associative [$\mathbf{ABC} = (\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$] and distributive over addition [$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$], provided \mathbf{A} , \mathbf{B} , and \mathbf{C} are all conformable. However, it is **not** commutative. To see why, consider $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$. The product \mathbf{AB} is an $m \times p$ matrix, but the product \mathbf{BA} is not conformable since $p \neq m$. Even if \mathbf{BA} were conformable, it is not the same as the product \mathbf{AB} .

MATLAB returns an error that “matrix dimensions must agree” when multiplying non-conformable objects.

For the system $\mathbf{y} = \mathbf{Ax}$, if $\dim(\mathbf{A}) = m \times n$ and $\dim(\mathbf{x}) = n \times 1$, $\dim(\mathbf{y}) = m \times 1$, i.e. \mathbf{y} is a column vector in \mathbb{R}^m .

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix}$$

$$\begin{aligned}\mathbf{AB} &= \begin{pmatrix} 1 \times 0 + 2 \times (-1) & 1 \times 1 + 2 \times 2 \\ 3 \times 0 + 4 \times (-1) & 3 \times 1 + 4 \times 2 \end{pmatrix} = \begin{pmatrix} -2 & 5 \\ -4 & 11 \end{pmatrix} \\ \mathbf{BA} &= \begin{pmatrix} 0 \times 1 + 1 \times 3 & 0 \times 2 + 1 \times 4 \\ -1 \times 1 + 2 \times 3 & -1 \times 2 + 2 \times 4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}\end{aligned}$$

2.2 Identity Matrix

We need to find an element that serves as $\mathbf{1}$ for vectors. The field axioms define this element by the property that $1 \times x = x$ for all x in the field. For vectors, we defined multiplication to involve matrices, so the element $\mathbf{1}$ will be a matrix, which we will call the *identity matrix* \mathbf{I} . We require that

$$\mathbf{Ix} = \mathbf{xI} = \mathbf{x}$$

for all \mathbf{x} . Assuming that \mathbf{x} is n -dimensional, \mathbf{I} must have n columns to be conformable. Also, the output of \mathbf{Ix} has n elements, so \mathbf{I} must have n rows. Therefore, we know that \mathbf{I} is a square $n \times n$ matrix whenever \mathbf{x} has dimension n .

Consider the first row of \mathbf{I} , i.e. $\mathbf{I}(1, :)$. We know from the definition of \mathbf{I} that $\mathbf{I}(1, :) \cdot \mathbf{x} = x_1$, so $\mathbf{I}(1, :) = (1 \ 0 \ 0 \ \cdots \ 0)$. For the second row, $\mathbf{I}(2, :) \cdot \mathbf{x} = x_2$, so $\mathbf{I}(2, :) = (0 \ 1 \ 0 \ \cdots \ 0)$. In general, the i th row of \mathbf{I} has a 1 at position i and zeros everywhere else

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

The identity matrix \mathbf{I} for a vector in \mathbb{R}^n is an $n \times n$ matrix with ones along the diagonal and zeroes everywhere else.

Our definition of the identity matrix also works for matrix multiplication. For any square matrix \mathbf{A}

$$\mathbf{IA} = \mathbf{AI} = \mathbf{A}$$

The identity matrix also works for non-square matrices; however, the dimensions of the identity matrix change if the multiplication is on the left or right side. If \mathbf{A} is an $m \times n$ matrix, then $\mathbf{IA} = \mathbf{A}$ if \mathbf{I} is an $m \times m$ identity matrix, and $\mathbf{AI} = \mathbf{A}$ if \mathbf{I} is an $n \times n$ identity matrix.

2.3 Matrix Transpose

The transpose operator flips the rows and columns of a matrix. The element a_{ij} in the original matrix becomes element a_{ji} in the transposed matrix. The transpose operator is a superscript “T”, as in \mathbf{A}^T .

In \mathbb{R}^1 , $\mathbf{I} = (1)$, which behaves like the real number 1.

Other notations for the matrix transpose include \mathbf{A}^t and \mathbf{A}' . The latter is used in MATLAB.

A transposed matrix is reflected about a diagonal drawn from the upper left to the lower right corner.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

Transposing an $m \times n$ matrix creates an $n \times m$ matrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

Transposing a column vector creates a row vector, and vice versa.

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}^T = (1 \quad 2 \quad 3), \quad (1 \quad 2 \quad 3)^T = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

2.4 Solving Linear Systems

Remember back to algebra, when you were asked to solve small systems of equations like

$$a_{11}x_1 + a_{12}x_2 = y_1$$

$$a_{21}x_1 + a_{22}x_2 = y_2$$

Your strategy was to manipulate the equations until they reach the form

$$x_1 = y'_1$$

$$x_2 = y'_2$$

In matrix form, this process transforms a matrix \mathbf{A} into the identity matrix

$$\begin{pmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y'_1 \\ y'_2 \end{pmatrix}$$

This leads us to our first strategy for solving linear systems of the form $\mathbf{Ax} = \mathbf{y}$. We manipulate both sides of the equation (\mathbf{A} and \mathbf{y}) until \mathbf{A} becomes the identity matrix. The vector \mathbf{x} then equals the transformed vector \mathbf{y}' . Because we will be applying the same transformations to both \mathbf{A} and \mathbf{y} , it is convenient to collect them both into an *augmented matrix* $(\mathbf{A} \quad \mathbf{y})$. For 2×2 system above, the augmented matrix is

$$\begin{pmatrix} a_{11} & a_{12} & y_1 \\ a_{21} & a_{22} & y_2 \end{pmatrix}$$

What operations can we use to transform \mathbf{A} into the identity matrix? There are three operations, called the *elementary row operations*, or EROs.

We often use the prime symbol (') to indicate that an unspecified new value is based on an old one. For example, y'_1 is a new value calculated from y_1 . In this case,

$$y'_1 = \frac{y_1 a_{22} - a_{12} y_2}{a_{11} a_{22} - a_{12} a_{21}}$$

1. Exchanging two rows. Since the order of the equations in our system is arbitrary, we can re-order the rows of the augmented matrix at will. By working with the augmented matrix, we ensure that both the left- and right-hand sides move together.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \xrightarrow{R_2 \leftrightarrow R_3} \begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{pmatrix}$$

2. Multiply any row by a scalar. Again, since we are working with the augmented matrix, multiplying a row by a scalar multiplies both the left- and right-hand sides of the equation by the same factor.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \xrightarrow{3R_2} \begin{pmatrix} 1 & 2 & 3 \\ 12 & 15 & 18 \\ 7 & 8 & 9 \end{pmatrix}$$

3. Add a scalar multiple of any row to any other row.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \xrightarrow{R_1 + 3R_2} \begin{pmatrix} 13 & 17 & 21 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Let's solve the system of equations

$$4x_1 + 8x_2 - 12x_3 = 44$$

$$3x_1 + 6x_2 - 8x_3 = 32$$

$$-2x_1 - x_2 = -7$$

This is a linear system of the form $\mathbf{Ax} = \mathbf{y}$ where the matrix \mathbf{A} and the vector \mathbf{y} are

$$\mathbf{A} = \begin{pmatrix} 4 & 8 & -12 \\ 3 & 6 & -8 \\ -2 & -1 & 0 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 44 \\ 32 \\ -7 \end{pmatrix}$$

The augmented matrix is therefore

$$\begin{pmatrix} 4 & 8 & -12 & 44 \\ 3 & 6 & -8 & 32 \\ -2 & -1 & 0 & -7 \end{pmatrix}$$

Now we apply the elementary row operations.

$$\begin{aligned} &\xrightarrow{\frac{1}{4}R_1} \begin{pmatrix} 1 & 2 & -3 & 11 \\ 3 & 6 & -8 & 32 \\ -2 & -1 & 0 & -7 \end{pmatrix} \\ &\xrightarrow{R_2-3R_1} \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 0 & 1 & -1 \\ -2 & -1 & 0 & -7 \end{pmatrix} \\ &\xrightarrow{R_3+2R_1} \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 0 & 1 & -1 \\ 0 & 3 & -6 & 15 \end{pmatrix} \end{aligned}$$

Notice that after three steps we have a zero at position (2,2). We need to move this row farther down the matrix to continue; otherwise we can't cancel out the 3 below it. This operation is called a "pivot".

$$\begin{aligned} &\xrightarrow{R_2 \leftrightarrow R_3} \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 3 & -6 & 15 \\ 0 & 0 & 1 & -1 \end{pmatrix} \\ &\xrightarrow{\frac{1}{3}R_2} \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{aligned}$$

At this point we have a matrix in *row echelon form*. The bottom triangle looks like the identity matrix. We could stop here and solve the system using back substitution:

$$\begin{aligned} x_3 &= -1 \\ x_2 + -2(-1) &= 5 \Rightarrow x_2 = 3 \\ x_1 + 2(3) - 3(-1) &= 11 \Rightarrow x_1 = 2 \end{aligned}$$

Or, we could keep going and place the augmented matrix into *reduced row echelon form*.

$$\begin{aligned} &\xrightarrow{R_1-2R_2} \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix} \\ &\xrightarrow{R_1-R_3} \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix} \\ &\xrightarrow{R_2+2R_3} \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{pmatrix} \end{aligned}$$

The left three columns are the identity matrix, so the resulting system of equations has been simplified to

$$\begin{aligned}x_1 &= 2 \\x_2 &= 3 \\x_3 &= -1\end{aligned}$$

2.5 Gaussian Elimination

Using EROs to transform the augmented matrix into the identity matrix is called *Gaussian elimination*. Let's develop an algorithm for Gaussian elimination for a general system of equations $\mathbf{Ax} = \mathbf{y}$ when \mathbf{A} is an $n \times n$ matrix. We begin with the augmented matrix

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & y_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & y_2 \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & y_n \end{pmatrix}$$

We need a 1 in the a_{11} position.

$$\xrightarrow{a_{11}^{-1}R_1} \begin{pmatrix} 1 & a_{11}^{-1}a_{12} & \cdots & a_{11}^{-1}a_{1n} & a_{11}^{-1}y_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & y_2 \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & y_n \end{pmatrix}$$

Now we zero out the a_{21} position using the first row multiplied by $-a_{21}$.

$$\xrightarrow{R_2 - a_{21}R_1} \begin{pmatrix} 1 & a_{11}^{-1}a_{12} & \cdots & a_{11}^{-1}a_{1n} & a_{11}^{-1}y_1 \\ 0 & a_{22} - a_{21}a_{11}^{-1}a_{12} & \cdots & a_{2n} - a_{21}a_{11}^{-1}a_{1n} & y_2 - a_{21}a_{11}^{-1}y_1 \\ \vdots & & \ddots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & y_n \end{pmatrix}$$

We keep zeroing out the entries a_{31} through a_{n1} using the first row. We end up with the matrix

$$\xrightarrow{R_n - a_{n1}R_1} \begin{pmatrix} 1 & a_{11}^{-1}a_{12} & \cdots & a_{11}^{-1}a_{1n} & a_{11}^{-1}y_1 \\ 0 & a_{22} - a_{21}a_{11}^{-1}a_{12} & \cdots & a_{2n} - a_{21}a_{11}^{-1}a_{1n} & y_2 - a_{21}a_{11}^{-1}y_1 \\ \vdots & & \ddots & & \vdots \\ 0 & a_{n2} - a_{n1}a_{11}^{-1}a_{12} & \cdots & a_{nn} - a_{n1}a_{11}^{-1}a_{1n} & y_n - a_{n1}a_{11}^{-1}y_1 \end{pmatrix}$$

This is looking a little complicated, so let's rewrite the matrix as

$$\begin{pmatrix} 1 & a'_{12} & \cdots & a'_{1n} & y'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & y'_2 \\ \vdots & & \ddots & & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} & y'_n \end{pmatrix}$$

The first column looks like the identity matrix, which is exactly what we want. Our next goal is to put the lower half of an identity matrix in column 2 by setting $a'_{22} = 1$ and $a_{32}, \dots, a_{n2} = 0$. Notice that this is the same as applying the above procedure to the sub-matrix

$$\begin{pmatrix} a'_{22} & \cdots & a'_{2n} & y'_2 \\ \vdots & \ddots & & \vdots \\ a'_{n2} & \cdots & a'_{nn} & y'_n \end{pmatrix}$$

After that, we can continue recursively until the left part of the augmented matrix is the identity matrix. We can formalize the Gaussian elimination algorithm as follows:

```

function GAUSSIAN ELIMINATION
  for  $j = 1$  to  $n$  do                                ▷ For every column
     $a_{jj}^{-1} R_j$                                        ▷ Set the element on the diagonal to 1
    for  $i = j + 1$  to  $n$  do                            ▷ For every row below the diagonal
       $R_i - a_{ij} R_j$                                    ▷ Zero the below-diagonal element
    end for
  end for
end function

```

We're ignoring pivoting here. In general, we need to check that $a_{jj} \neq 0$; if it is, we swap the row for one below that has a nonzero term in the j th column.

2.6 Computational Complexity of Gaussian Elimination

How many computational operations are needed to perform Gaussian elimination on an $n \times n$ matrix? Let's start by counting operations when reducing the first column. The scaling of the first row ($a_{11}^{-1} R_1$) requires n operations. (There are $n + 1$ entries in the first row of the augmented matrix if you include the value y_1 ; however, we know the result in the first column, $a_{11}^{-1} a_{11}$, will always equal 1, so we don't need to compute it.) Similarly, zeroing out a single row below requires n multiplications and n subtractions. (Again, there are $n + 1$ columns, but we know the result in column 1 will be zero.) In the first column, there are $n - 1$ rows below to zero out, so the total number of operations is

$$\underbrace{n}_{a_{11}^{-1} R_1} + \underbrace{2(n-1)n}_{R_i - a_{i1} R_1} = 2n^2 - n$$

After we zero the bottom of the first row, we repeat the procedure on the $(n - 1) \times (n - 1)$ submatrix, and so on until we reach the 1×1 "submatrix" that includes only a_{nn} . We add up the number of

Remember that

$$\begin{aligned} \sum_{k=1}^n 1 &= n \\ \sum_{k=1}^n k &= \frac{n(n+1)}{2} \\ \sum_{k=1}^n k^2 &= \frac{n(n+1)(2n+1)}{6} \end{aligned}$$

operations for each of these n submatrices

$$\begin{aligned}
 &= \sum_{k=1}^n (2k^2 - k) \\
 &= 2 \sum_{k=1}^n k^2 - \sum_{k=1}^n k \\
 &= \frac{n(n+1)(2n+1)}{3} - \frac{n(n+1)}{2} \\
 &= \mathcal{O}(n^3)
 \end{aligned}$$

Thus the number of operations required to bring an $n \times n$ matrix into row-echelon form is on the order of n^3 . The number of operations needed to perform back substitution and solve the system is $\mathcal{O}(n^2)$. This raises two important points.

The \mathcal{O} , or “big-O” notation indicates the rate of growth of a function for large values. Any polynomial of degree d is $\mathcal{O}(d)$.

1. Gaussian elimination scales cubically. A system with twice as many equations takes eight times longer to solve.
2. The computational bottleneck is generating the row echelon matrix. Back substitution (or creating the reduced row echelon matrix) is significantly faster.

2.7 Solving Linear Systems in MATLAB

MATLAB has multiple functions for solving linear systems. Given variables **A** and **y**, you can use

- `linsolve(A,y)` to solve using LU decomposition, a variant of Gaussian elimination.
- `(A \ y)` to let MATLAB choose the best algorithm based on the size and structure of **A**.
- `rref([A y])` to compute the reduced row echelon form of the augmented matrix.

The Finite Difference Method

When you first learned about derivatives, they were probably introduced as the limit of a finite difference

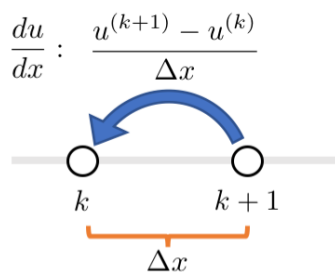
$$\frac{du}{dx} = \lim_{a \rightarrow b} \frac{u(b) - u(a)}{b - a}$$

As the distance between points a and b shrinks to zero, the finite difference becomes infinitesimal. The resulting differential equations must be solved with integral calculus. This chapter presents an alternative, numerical method for solving differential equations. Rather than shrink the finite differences all the way to zero, we leave a small but finite gap. The resulting algebraic equations approximate the differential equation and can be solved without any tools from calculus.

Updated 1/22/2018.

3.1 Finite Differences

Solving a differential equation analytically yields a solution over the entire domain (the region between the boundary conditions). By contrast, numerical methods find solutions to a differential equation at only a discrete set of points, or *nodes*, in the domain. The derivatives in the differential equation are discretized by converting them into *finite differences*. For example, we can approximate a first derivative as the change between two nodes divided by the distance between the nodes

$$\frac{du}{dx} : \frac{u^{(k+1)} - u^{(k)}}{\Delta x}$$


where $u^{(k)}$ is the value of the variable at the k th node. To approxi-

We write the value of u at node k as $u^{(k)}$. Using a superscript with parentheses avoids confusion with expressions u^k (the k th power of u) and u_k (the k th element of a vector named \mathbf{u}).

mate a second derivative, we calculate the finite difference between nodes $u^{(k+1)}$ and $u^{(k)}$; and nodes $u^{(k)}$ and $u^{(k-1)}$. We divide this “difference between differences” by the distance between the centers of the nodes:

$$\frac{d^2u}{dx^2} : \frac{u^{(k+1)} - 2u^{(k)} + u^{(k-1)}}{\Delta x^2}$$

$$\frac{du}{dx} : \frac{u^{(k)} - u^{(k-1)}}{\Delta x} \quad \frac{u^{(k+1)} - u^{(k)}}{\Delta x}$$

$k-1 \quad k \quad k+1$
 $\Delta x \quad \Delta x$
 Δx^2

3.2 Linear Differential Equations

In order to generate a set of linear algebraic equations, the starting ODE or PDE must be linear. For differential equations, linearity means that the dependent variable (i.e. u) only appears in linear expressions; there can be nonlinearities involving only the independent variables. Remember that differentiation is a linear operator, so all derivatives of u are linear!

For example, consider the PDE with dependent variable $u(t, x)$:

$$t^2 \frac{\partial u}{\partial t} = c_1 \frac{\partial^2 u}{\partial x^2} + c_2 \sin(x) \frac{\partial u}{\partial x} + c_3 e^{tx}$$

This PDE is linear in u . However, the ODE

$$u \frac{du}{dx} = 0$$

is not linear. In general, for a variable $u(t, x)$, any terms of the form

$$f(t, x) \frac{\partial^n u}{\partial t^n} \quad \text{or} \quad f(t, x) \frac{\partial^n u}{\partial x^n}$$

are linear.

You might be wondering why we only require that a PDE be linear in the dependent variable. Why do nonlinearities in the independent variables not lead to nonlinear algebraic equations? Remember that in the finite difference method we discretize the independent variables

The finite difference method will still work on nonlinear PDEs; however, the resulting set of equations are nonlinear.

As a rule of thumb, you can tell if a PDE is linear in u by ignoring the derivative operators and seeing if the resulting algebraic equation is linear in u .

across a grid and solve for the dependent variable at each node. Before solving, the value of the dependent variable is unknown. However, the value of all the independent variables are known, i.e. we know the location of every node in space and time. So, we can evaluate all terms involving the independent variables when setting up our equations.

3.3 Discretizing a Linear Differential Equation

Converting a linear differential equation into a set of linear algebraic equations requires three steps.

1. Divide the domain into n equally-sized intervals. Creating n intervals requires $n + 1$ points, or *nodes*, labeled $0, 1, \dots, n$. The spacing between each node is $\Delta x = l/n$.
2. Starting with the interior nodes $(1, 2, \dots, n - 1)$, we rewrite the differential equation at each node using finite differences.

$$\begin{aligned} u &\rightarrow u^{(k)} \\ \frac{du}{dx} &\rightarrow \frac{u^{(k+1)} - u^{(k)}}{\Delta x} \\ \frac{d^2u}{dx^2} &\rightarrow \frac{u^{(k+1)} - 2u^{(k)} + u^{(k-1)}}{\Delta x^2} \end{aligned}$$

3. Add equations to enforce the boundary conditions at the boundary nodes.

For example, consider the ordinary differential equation

$$\frac{d^2u}{dx^2} + \frac{du}{dx} - 6u = 0, \quad u(0) = 0, \quad u(1) = 3$$

If we divide the domain $[0, 1]$ into four sections, then $n = 4$ and $\Delta x = n/l = 1/4 = 0.25$. We have five nodes $(0, 1, 2, 3, 4)$, three of which are interior nodes $(1, 2, 3)$. We rewrite the ODE using finite differences at each of the interior nodes.

$$\begin{aligned} \frac{u^{(2)} - 2u^{(1)} + u^{(0)}}{(0.25)^2} + \frac{u^{(2)} - u^{(1)}}{0.25} - 6u^{(1)} &= 0 \quad [\text{node 1}] \\ \frac{u^{(3)} - 2u^{(2)} + u^{(1)}}{(0.25)^2} + \frac{u^{(3)} - u^{(2)}}{0.25} - 6u^{(2)} &= 0 \quad [\text{node 2}] \\ \frac{u^{(4)} - 2u^{(3)} + u^{(2)}}{(0.25)^2} + \frac{u^{(4)} - u^{(3)}}{0.25} - 6u^{(3)} &= 0 \quad [\text{node 3}] \end{aligned}$$

which simplifies to the equations

$$\begin{aligned} 16u^{(0)} - 42u^{(1)} + 20u^{(2)} &= 0 \\ 16u^{(1)} - 42u^{(2)} + 20u^{(3)} &= 0 \\ 16u^{(2)} - 42u^{(3)} + 20u^{(4)} &= 0 \end{aligned}$$

Now we can add equations for the boundary nodes. Node 0 corresponds to $x = 0$, so the boundary condition tells us that $u^{(0)} = 0$. Similarly, node 4 corresponds to $x = 1$, so $u^{(4)} = 3$. Combining these two boundary equations with the equations for the interior nodes yields the final linear system

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 16 & -42 & 20 & 0 & 0 \\ 0 & 16 & -42 & 20 & 0 \\ 0 & 0 & 16 & -42 & 20 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u^{(0)} \\ u^{(1)} \\ u^{(2)} \\ u^{(3)} \\ u^{(4)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 3 \end{pmatrix}$$

Solving by Gaussian elimination yields

$$\begin{pmatrix} u^{(0)} \\ u^{(1)} \\ u^{(2)} \\ u^{(3)} \\ u^{(4)} \end{pmatrix} = \begin{pmatrix} 0.00 \\ 0.51 \\ 1.07 \\ 1.84 \\ 3.00 \end{pmatrix}$$

We can compare our numerical solution to the exact solution for this expression, which is

$$u(x) = \frac{3}{e^2 - e^{-3}} (e^{2x} - e^{-3x})$$

| x | Numerical Solution | Exact Solution | Error |
|------|--------------------|----------------|-------|
| 0 | 0.00 | 0.00 | 0.0% |
| 0.25 | 0.51 | 0.48 | 5.7% |
| 0.50 | 1.07 | 1.02 | 4.7% |
| 0.75 | 1.84 | 1.79 | 2.6% |
| 1 | 3.00 | 3.00 | 0.0% |

We used only five nodes to solve this ODE, yet our relative error is less than 6%!

3.4 Boundary Conditions

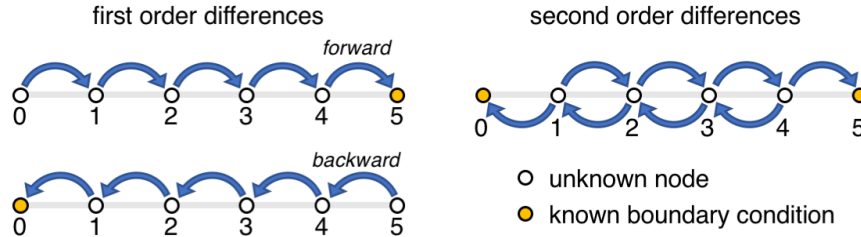
There are two ways to write a finite difference approximation for the first derivative at node k . We can write the *forward difference* using node $k + 1$:

$$\frac{du}{dx} \rightarrow \frac{u^{(k+1)} - u^{(k)}}{\Delta x}$$

or we can use the *backward difference* using the previous node at $k - 1$:

$$\frac{du}{dx} \rightarrow \frac{u^{(k)} - u^{(k-1)}}{\Delta x}$$

The choice of forward or backward differences depends on the boundary conditions. For a first order ODE, we are given a boundary condition at either $x = 0$ or $x = l$. If we are given $u(0)$, we use backward differences. If we are given $u(l)$, we use forward differences. Otherwise, we run out of nodes when writing equations for the finite differences.



You cannot avoid the issue by using the difference $u^{(0)} - u^{(1)}$ for node 0 and $u^{(1)} - u^{(0)}$ for node 1. These equations are *linearly dependent*, which leaves us with too little information when solving the system.

Second order finite differences require nodes on both sides. This is related to the necessity of two boundary conditions, since we cannot write finite difference approximations for nodes at either end. If your ODE comes with two boundary conditions, you can choose either forward or backward differences to approximate any first order derivatives.

4

Inverses, Solvability, and Rank

4.1 Matrix Inverses

Updated 2/5/2018.

So far we've demonstrated how Gaussian elimination can solve linear systems of the form $\mathbf{Ax} = \mathbf{y}$. Gaussian elimination involves a series of elementary row operations to transform the coefficient matrix \mathbf{A} into the identity matrix. While Gaussian elimination works well, our initial goal of defining an algebra for vectors requires something stronger – a multiplicative inverse. For vectors, the multiplicative inverse is called a *matrix inverse*. For any square matrix, a matrix inverse (if it exists) is a square matrix \mathbf{A}^{-1} such that

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

If we could prove the existence of a *matrix inverse* for \mathbf{A} , we could solve a wide variety of linear systems, including $\mathbf{Ax} = \mathbf{y}$.

This definition is analogous to the field axiom that there exists a^{-1} such that $a^{-1}a = 1$ for all nonzero a . Since scalar multiplication always commutes, $a^{-1}a = aa^{-1}$. Since matrix multiplication doesn't commute, we need to state this property separately.

$$\begin{aligned}\mathbf{Ax} &= \mathbf{y} \\ \mathbf{A}^{-1}\mathbf{Ax} &= \mathbf{A}^{-1}\mathbf{y} \\ \mathbf{Ix} &= \mathbf{A}^{-1}\mathbf{y} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{y}\end{aligned}$$

The existence of the matrix inverse and being amenable to Gaussian elimination are related. While the end result is the same (a transformation of the coefficient matrix into the identity matrix), the processes are different. The Gaussian elimination algorithm applies a series of elementary row operations, including pivoting to avoid numerical issues. For a matrix inverse to exist, it must be able to capture all of these operations in a single matrix multiplication. This condensing of Gaussian elimination is not a trivial task.

Our first goal of this chapter is to prove the existence of the matrix inverse for any coefficient matrix that can be solved by Gaussian elimination. Then we will derive a method to construct the matrix inverse

if it exists. Finally, this chapter formalizes the requirements for solvability of a linear system of equations, which is related to the existence of the matrix inverse.

4.2 Elementary Matrices

Before proving the existence of the matrix inverse, we need to add another matrix manipulation tool to our arsenal – the *elementary matrix*. An elementary matrix is constructed by applying any single elementary row operation to the identity matrix. For example, consider swapping the second and third rows of the 3×3 identity matrix:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_2 \leftrightarrow R_3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \mathbf{E}_{R_2 \leftrightarrow R_3}$$

We use the notation \mathbf{E}_r to denote an elementary matrix constructed using the elementary row operation r .

Notice what happens when we left multiply a matrix with an elementary matrix.

$$\mathbf{E}_{R_2 \leftrightarrow R_3} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{pmatrix}$$

Multiplication by the elementary matrix exchanges the second and third rows – the same operation that created the elementary matrix. The same idea works for other elementary row operations, such as scalar multiplication

$$\mathbf{E}_{3R_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{E}_{3R_2} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 12 & 15 & 18 \\ 7 & 8 & 9 \end{pmatrix}$$

Multiplication by an elementary matrix on the right applies the same operation to the columns of the matrix.

and addition by a scaled row

$$\mathbf{E}_{R_2+2R_3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{E}_{R_2+2R_3} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 18 & 21 & 24 \\ 7 & 8 & 9 \end{pmatrix}$$

4.3 Proof of Existence for the Matrix Inverse

We are now ready to prove the existence of the inverse for any coefficient matrix that is solvable by Gaussian elimination, i.e. any square

matrix that can be transformed into the identity matrix with elementary row operations. We will prove existence in three steps.

1. Construct a matrix \mathbf{P} that looks like a left inverse ($\mathbf{PA} = \mathbf{I}$).
2. Show that this left inverse is also a right inverse ($\mathbf{AP} = \mathbf{I}$).
3. Show that the matrix inverse is unique, implying that \mathbf{P} must be the inverse of \mathbf{A} .

Theorem. *Suppose matrix \mathbf{A} can be reduced to the identity matrix \mathbf{I} by elementary row operations. Then there exists a matrix \mathbf{P} such that $\mathbf{PA} = \mathbf{I}$.*

Proof. We assume that reducing \mathbf{A} to \mathbf{I} requires k elementary row operations. Let $\mathbf{E}_1, \dots, \mathbf{E}_k$ be the associated elementary matrices. Then

$$\begin{aligned}\mathbf{E}_k \mathbf{E}_{k-1} \cdots \mathbf{E}_2 \mathbf{E}_1 \mathbf{A} &= \mathbf{I} \\ (\mathbf{E}_k \mathbf{E}_{k-1} \cdots \mathbf{E}_2 \mathbf{E}_1) \mathbf{A} &= \mathbf{I} \\ \mathbf{PA} &= \mathbf{I}\end{aligned}$$

where $\mathbf{P} = \mathbf{E}_k \mathbf{E}_{k-1} \cdots \mathbf{E}_2 \mathbf{E}_1$. □

Theorem. *If $\mathbf{PA} = \mathbf{I}$ then $\mathbf{AP} = \mathbf{I}$.*

Proof.

$$\begin{aligned}\mathbf{PA} &= \mathbf{I} \\ \mathbf{PAP} &= \mathbf{IP} \\ \mathbf{P}(\mathbf{AP}) &= \mathbf{P}\end{aligned}$$

Since \mathbf{P} multiplied by \mathbf{AP} gives \mathbf{P} back again, \mathbf{AP} must equal the identity matrix ($\mathbf{AP} = \mathbf{I}$). □

Theorem. *The inverse of a matrix is unique.*

Proof. Let \mathbf{A}^{-1} be the inverse of \mathbf{A} , i.e. $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \mathbf{AA}^{-1}$. Suppose there exists a matrix $\mathbf{P} \neq \mathbf{A}^{-1}$ such that $\mathbf{PA} = \mathbf{I} = \mathbf{AP}$.

$$\begin{aligned}\mathbf{PA} &= \mathbf{I} \\ \mathbf{PAA}^{-1} &= \mathbf{IA}^{-1} \\ \mathbf{PI} &= \mathbf{A}^{-1} \\ \mathbf{P} &= \mathbf{A}^{-1}\end{aligned}$$

This contradicts our supposition that $\mathbf{P} \neq \mathbf{A}^{-1}$, so \mathbf{A}^{-1} must be unique. □

4.4 Computing the Matrix Inverse

Our proof of existence of the matrix inverse also provided a method of construction. We performed Gaussian elimination on a matrix, constructing an elementary matrix for each step. These elementary matrices were multiplied together to form the matrix inverse. In practice, this method would be wildly inefficient. Transforming an $n \times n$ matrix to reduced row echelon form requires $\mathcal{O}(n^2)$ elementary row operations, so we would need to construct and multiply $\mathcal{O}(n^2)$ elementary matrices. Since naive matrix multiplication requires $\mathcal{O}(n^3)$ operations per matrix, constructing a matrix inverse with this method requires $\mathcal{O}(n^5)$ operations! Since Gaussian elimination is $\mathcal{O}(n^3)$, we would be far better off avoiding the matrix inverse entirely.

Fortunately, there are better methods for constructing matrix inverses. One of the best is called the *side-by-side method*. To see how the side-by-side method works, consider constructing an inverse for the square matrix \mathbf{A} . We can use Gaussian elimination to transform \mathbf{A} into the identity matrix, which we can represent with a series of k elementary matrices.

$$\underbrace{\mathbf{E}_k \mathbf{E}_{k-1} \cdots \mathbf{E}_2 \mathbf{E}_1}_{\mathbf{A}^{-1}} \mathbf{A} = \mathbf{I}$$

What would happen if we simultaneously apply the same elementary row operations to another identity matrix?

$$\underbrace{\mathbf{E}_k \mathbf{E}_{k-1} \cdots \mathbf{E}_2 \mathbf{E}_1}_{\mathbf{A}^{-1}} \mathbf{I} = \mathbf{A}^{-1} \mathbf{I} = \mathbf{A}^{-1}$$

In the side-by-side method, we start with an augmented matrix containing the $n \times n$ matrix \mathbf{A} and an $n \times n$ identity matrix. Then we apply Gaussian elimination to transform \mathbf{A} into \mathbf{I} . The augmented matrix ensure that the same elementary row operations will be applied to the identity matrix, yielding the inverse of \mathbf{A} :

$$(\mathbf{A} \quad \mathbf{I}) \xrightarrow{\text{EROs}} (\mathbf{I} \quad \mathbf{A}^{-1})$$

Let's solve the following system by constructing the matrix inverse.

$$\begin{aligned} 3x_1 + 2x_2 &= 7 \\ x_1 + x_2 &= 4 \end{aligned}$$

In matrix form,

$$\mathbf{A} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$$

The fastest matrix multiplication algorithm is $\mathcal{O}(n^{2.7373})$, although this is not a big help in practice.

Like the augmented matrix $(\mathbf{A} \mathbf{y})$, there is no direct interpretation of $(\mathbf{A} \mathbf{I})$. It is simply a convenient way to apply the same EROs to both \mathbf{A} and \mathbf{I} .

We start with the augmented matrix for the side-by-side method.

$$\begin{aligned}
 \begin{pmatrix} 3 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} &\xrightarrow{R_1 \leftrightarrow R_2} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix} \\
 &\xrightarrow{R_2 - 3R_1} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & -1 & 1 & -3 \end{pmatrix} \\
 &\xrightarrow{-R_2} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 3 \end{pmatrix} \\
 &\xrightarrow{R_1 - R_2} \begin{pmatrix} 1 & 0 & 1 & -2 \\ 0 & 1 & -1 & 3 \end{pmatrix}
 \end{aligned}$$

Thus, the matrix inverse is

$$\mathbf{A}^{-1} = \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix}$$

and we can compute the solution by matrix multiplication.

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} = \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \end{pmatrix}$$

We can verify that \mathbf{A}^{-1} is a matrix inverse

$$\begin{aligned}
 \mathbf{A}^{-1}\mathbf{A} &= \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{I} \\
 \mathbf{A}\mathbf{A}^{-1} &= \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{I}
 \end{aligned}$$

The nice thing about having a matrix inverse is that if only the right hand side of a system of equations change, we do not need to retransform the coefficient matrix. For example, to solve

$$\begin{aligned}
 3x_1 + 2x_2 &= 1 \\
 x_1 + x_2 &= 3
 \end{aligned}$$

we can re-use \mathbf{A}^{-1} since \mathbf{A} is unchanged (only \mathbf{y} is different).

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} = \begin{pmatrix} 1 & -2 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} -5 \\ 8 \end{pmatrix}$$

4.5 Numerical Issues

Matrix inverses are a powerful method for solving linear systems.

However, calculating a matrix inverse should always be your last resort. There are far more efficient and numerically stable methods for solving linear systems. Reasons against using a matrix inverse include:

1. **Computation time.** While the side-by-side method is more efficient for constructing inverses than multiplying elementary matrices, it is still slower than Gaussian elimination for solving linear systems of the form $\mathbf{Ax} = \mathbf{y}$. Both side-by-side and Gaussian elimination reduce an augmented matrix. For an $n \times n$ matrix \mathbf{A} , the augmented matrix for Gaussian elimination ($\mathbf{A} \mathbf{y}$) is $n \times (n+1)$. The augmented matrix for the side-by-side method ($\mathbf{A} \mathbf{I}$) is $n \times 2n$. Solving for the inverse requires nearly twice the computations as solving the linear system directly. Having the inverse allows us to “resolve” the system for a new right hand side (\mathbf{y}) for only the cost of a matrix multiplication. However, there are variants of Gaussian elimination – such as LU decomposition – that allow resolving without repeating the entire reduction of the coefficient matrix \mathbf{A} .
2. **Memory.** Most large matrices in engineering are *sparse*. Sparse matrices contain very few nonzero entries; matrices with less than 0.01% nonzero entries are not uncommon. Examples of sparse matrices include matrices generated from finite difference approximations or matrices showing connections between nodes in large networks. Computers store sparse matrices by only storing the nonzero entries and their locations. However, there is no guarantee that the inverse of a sparse matrix will also be sparse. Consider the arrow matrix, a matrix with ones along the diagonal and last column and row.

Imagine the connection matrix for Facebook. It would have hundreds of millions of rows and columns, but each person (row) would only have nonzero entries for a few hundred people (columns) that they knew.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

An $n \times n$ arrow matrix has n^2 entries but only $3n - 2$ nonzeros. However, the inverse of an arrow matrix always has 100% nonzeros. For example, the inverse of the 8×8 matrix above is

A 1000×1000 arrow matrix has less than 0.3% nonzeros.

$$\mathbf{A}^{-1} = \frac{1}{6} \begin{pmatrix} 5 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & 5 & -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & 5 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & 5 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 5 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 5 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 5 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 \end{pmatrix}$$

Calculating the inverse of a large, sparse matrix could require orders of magnitude more memory than the original matrix. For some matrices, storing – much less computing – the inverse is impossible.

Despite its disadvantages, the matrix inverse is still a powerful construct. A multiplicative inverse for matrices is necessary for many algebraic manipulations, and the inverse can be used to simply or prove many matrix equations. Just remember to think critically about the need for a matrix inverse before calculating one.

4.6 Inverses of Elementary Matrices

We conclude with another interesting property of elementary matrices. We said before that left multiplication by an elementary matrix performs an elementary row operation (the same ERO that was used to construct the elementary matrix). Left multiplication by the inverse of an elementary matrix “undoes” the operation of the elementary matrix. For example, the elementary matrix \mathbf{E}_{3R_2} scales the second row by two. The inverse $\mathbf{E}_{3R_2}^{-1}$ would scale the second row by $1/2$, undoing the scaling by two. Similarly, $\mathbf{E}_{R_2 \leftrightarrow R_3}$ swaps rows two and three, and $\mathbf{E}_{R_2 \leftrightarrow R_3}^{-1}$ swaps them back. The proof of this property is straightforward.

Theorem. *If the elementary matrix \mathbf{E}_r performs the elementary row operation r , then left multiplication by the inverse \mathbf{E}_r^{-1} undoes this operation.*

Proof.

$$\mathbf{E}_r^{-1}(\mathbf{E}_r \mathbf{A}) = (\mathbf{E}_r^{-1} \mathbf{E}_r) \mathbf{A} = (\mathbf{I}) \mathbf{A} = \mathbf{A}$$

□

4.7 Rank

Consider the linear system

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 2 & -4 \\ -2 & 0 & -6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -2 \\ -4 \end{pmatrix}$$

and the row echelon form of the associated augmented matrix

$$\begin{pmatrix} 1 & 0 & 3 & 2 \\ 0 & 2 & -4 & -2 \\ -2 & 0 & -6 & -4 \end{pmatrix} \xrightarrow{\frac{1}{2}R_2} \xrightarrow{R_3+2R_1} \begin{pmatrix} 1 & 0 & 3 & 2 \\ 0 & 1 & -2 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Notice that the last row is all zeros. We have no information about the last entry (x_3). However, this does not mean we cannot solve the

linear system. Since x_3 is unknown, let us assign its value the symbol α . Then, by back substitution

$$\begin{aligned} x_3 &= \alpha \\ x_2 - 2x_3 &= -1 \Rightarrow x_2 = 2\alpha - 1 \\ x_1 + 3x_3 &= 2 \Rightarrow x_1 = 2 - 3\alpha \end{aligned}$$

The above linear system has not one solution, but infinitely many. There is a solution for every value of the parameter α , so we say the system has a *parameterized solution*.

Parameterized solutions are necessary any time row echelon reduction of a matrix leads to one or more rows with all zero entries. The number of nonzero rows in the row echelon form of a matrix is the matrix's *rank*. The rank of a matrix can be calculated by counting the number of nonzero rows after the matrix is transformed into row echelon form by Gaussian elimination. In general, if a matrix with n columns has rank n , it is possible to find a unique solution to the system $\mathbf{Ax} = \mathbf{y}$. If $\text{rank}(\mathbf{A}) < n$, there may be infinitely many solutions. These solutions require that we specify $n - \text{rank}(\mathbf{A})$ parameters.

We denote the rank of a matrix \mathbf{A} as $\text{rank}(\mathbf{A})$.

Matrices has both a *row rank* (the number of nonzero rows in row-reduced echelon form) and a *column rank* (the number of nonzero columns in a column-reduced echelon form). Thus the concept of rank also applies to nonsquare matrices. However, the row and column ranks are always equivalent, even if the matrix is not square:

Theorem. *The row rank of a matrix equals the column rank of the matrix, i.e. $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^T)$.*

Proof. We will defer the proof of this theorem until our discussion of the Fundamental Theorem of Linear Algebra. \square

The equivalence of the row and column ranks implies an upper bound on the rank of nonsquare matrices.

Theorem. *The rank of a matrix is less than or equal to the smallest dimension of the matrix, i.e. $\text{rank}(\mathbf{A}) \leq \min(\dim \mathbf{A})$.*

Proof. The row rank of \mathbf{A} is the number of nonzero rows in the row-reduced \mathbf{A} , so the rank of \mathbf{A} must be less than the number of rows in \mathbf{A} . Since the row rank is also equal to the column rank, there must also be $\text{rank}(\mathbf{A})$ nonzero columns in the column-reduced \mathbf{A} . So the rank of \mathbf{A} must never be larger than either the number of rows or number of columns in \mathbf{A} . \square

A matrix that has the maximum possible rank (rank n for an $n \times n$ square matrix or rank $\min(m, n)$ for an $m \times n$ rectangular matrix), we say the matrix is *full rank*. A matrix that is not full rank is *rank deficient*.

Linear Independence

The notion of rank is deeply tied to the concept of *linear independence*. A vector \mathbf{x}_i is linearly dependent on a set of n vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ if there exists coefficients c_1, c_2, \dots, c_n such that

$$\mathbf{x}_i = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_n\mathbf{x}_n$$

A set of vectors are *linearly dependent* if one of the vectors can be expressed as a linear combination of some of the others. This is analogous to saying there exists a set of coefficients c_1, \dots, c_n , not all equal to zero, such that

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \dots + c_n\mathbf{x}_n = \mathbf{0}$$

If a matrix with n rows has rank $k < n$, then $n - k$ of the rows are linearly dependent on the other k rows. Going back to our previous example, the matrix

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 2 & -4 \\ -2 & 0 & -6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

has rank 2 since there are two nonzero rows in the row-reduced matrix. Therefore, one of the rows must be linearly dependent on the other rows. Indeed, we see that the last row $\begin{pmatrix} -2 & 0 & -6 \end{pmatrix}$ is -2 times the first row $\begin{pmatrix} 1 & 0 & 3 \end{pmatrix}$. During row reduction by Gaussian elimination, any linearly dependent rows will be completely zeroed out, revealing the rank of the matrix.

Rank and linear dependence tell us about the information content of a coefficient matrix. If some of the rows of the coefficient matrix are linearly dependent, then matrix is rank deficient and no unique solution exists. These matrices are also information deficient – we do not have one independent expression for each variable. Without a separate piece of information for each variable, we cannot unique map between the input \mathbf{x} and the output \mathbf{y} . However, if we introduce a separate parameter for each zeroed row, we are artificially providing the missing information. We can find a new solution every time we specify values for the parameters.

Homogeneous Systems ($\mathbf{Ax} = \mathbf{0}$)

A linear systems of equations is *homogeneous* if and only if the right hand side vector (\mathbf{y}) is equal to the zero vector ($\mathbf{0}$). Homogeneous systems always have at least one solution, $\mathbf{x} = \mathbf{0}$, since $\mathbf{A}\mathbf{0} = \mathbf{0}$. The zero solution to a homogeneous system is called the *trivial solution*.

Note the difference between $\mathbf{x}_1, \dots, \mathbf{x}_n$, a set of n vectors; and x_1, \dots, x_n , a set of n scalars that form the elements of a vector \mathbf{x} .

Some homogeneous systems have a nontrivial solution, i.e. a solution $\mathbf{x} \neq \mathbf{0}$. If a homogeneous system has a nontrivial solution, then it has infinitely many solutions, a result we state as follows

Theorem. *Any linear combination of nontrivial solutions to a homogeneous linear system is also a solution.*

Proof. Suppose we had two solutions, \mathbf{x} and \mathbf{x}' to the homogeneous system $\mathbf{Ax} = \mathbf{0}$. Then

$$\begin{aligned}\mathbf{A}(k\mathbf{x} + k'\mathbf{x}') &= \mathbf{A}(k\mathbf{x}) + \mathbf{A}(k'\mathbf{x}') \\ &= k(\mathbf{Ax}) + k'(\mathbf{Ax}') \\ &= k(\mathbf{0}) + k'(\mathbf{0}) \\ &= \mathbf{0}\end{aligned}$$

This proof is equivalent to showing that $\mathbf{Ax} = \mathbf{0}$ satisfies our definition of linear systems: $f(k_1x_1 + k_2x_2) = k_1f(x_1) + k_2f(x_2)$.

Since there are infinitely many scalars k and k' , we can generate infinitely many solutions to the homogeneous system $\mathbf{Ax} = \mathbf{0}$. \square

There is a connection between the solvability of nonhomogeneous systems $\mathbf{Ax} = \mathbf{y}$ and the corresponding homogeneous system $\mathbf{Ax} = \mathbf{0}$. If there exists at least one solution to $\mathbf{Ax} = \mathbf{y}$ and a nontrivial solution to $\mathbf{Ax} = \mathbf{0}$, then there are infinitely many solutions to $\mathbf{Ax} = \mathbf{y}$. To see why, let \mathbf{x}_{nh} be the solution to the nonhomogeneous system ($\mathbf{Ax}_{\text{nh}} = \mathbf{y}$) and \mathbf{x}_{h} be a nontrivial solution to the homogeneous system $\mathbf{Ax}_{\text{h}} = \mathbf{0}$. Then any of the infinite linear combinations $\mathbf{x}_{\text{nh}} + k\mathbf{x}_{\text{h}}$ is also a solution to $\mathbf{Ax} = \mathbf{y}$ since

$$\mathbf{A}(\mathbf{x}_{\text{nh}} + k\mathbf{x}_{\text{h}}) = \mathbf{Ax}_{\text{nh}} + k\mathbf{Ax}_{\text{h}} = \mathbf{y} + k\mathbf{0} = \mathbf{y}$$

General Solvability

For any linear system of equations, we can use the rank of the coefficient matrix and the augmented matrix to determine the existence and number of solutions. The relationship between solvability and rank is captured by the Rouché-Capelli theorem:

Theorem. *A linear system $\mathbf{Ax} = \mathbf{y}$ where $\mathbf{x} \in \mathbb{R}^n$ has a solution if and only if the rank of the coefficient matrix equals the rank of the augmented matrix, i.e. $\text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A} \ \mathbf{y}])$. Furthermore, the solution is unique if $\text{rank}(\mathbf{A}) = n$; otherwise there are infinitely many solutions.*

Proof. We will sketch several portions of this proof to give intuition about the theorem. A more rigorous proof is beyond the scope of this class.

1. **Homogeneous systems.** For a homogeneous system $\mathbf{Ax} = \mathbf{0}$, we know that $\text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A} \ \mathbf{0}])$. (Since the rank of \mathbf{A} is equal to the number of nonzero columns, adding another column of zeros will never change the rank.) Thus, we know that homogeneous systems are always solvable, at least by the trivial solution $\mathbf{x} = \mathbf{0}$. If $\text{rank}(\mathbf{A}) = n$, then the trivial solution is unique and is the only solution. If $\text{rank}(\mathbf{A}) < n$, there are infinitely many parameterized solutions.
2. **Full rank, nonhomogeneous systems.** For a nonhomogeneous system ($\mathbf{Ax} = \mathbf{y}$, $\mathbf{y} \neq \mathbf{0}$), we expect a unique solution if and only if adding the column \mathbf{y} to the coefficient matrix doesn't change the rank. For this to be true, \mathbf{y} must be linearly dependent on the other columns in \mathbf{A} ; otherwise, adding a new linearly independent column would increase the rank. If \mathbf{y} is linearly dependent on the n columns of \mathbf{A} , it must be true that there exists weights c_1, c_2, \dots, c_n such that

$$c_1 \mathbf{A}(:, 1) + c_2 \mathbf{A}(:, 2) + \dots + c_n \mathbf{A}(:, n) = \mathbf{y}$$

based on the definition of linear dependence. But the above expression can be rewritten in matrix form as

$$(\mathbf{A}(:, 1) \ \mathbf{A}(:, 2) \ \dots \ \mathbf{A}(:, n)) \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \mathbf{Ac} = \mathbf{y}$$

which shows that the system has a unique solution $\mathbf{x} = \mathbf{c}$.

3. **Rank deficient, nonhomogeneous systems.** Let $\text{rank}(\mathbf{A}) = k < n$. Then the row-reduced form of \mathbf{A} will have k rows that resemble the identity matrix and $n - k$ rows of all zeros:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2k} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} & \dots & a_{kn} \\ a_{k+1,1} & a_{k+1,2} & \dots & a_{k+1,k} & \dots & a_{k+1,n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nk} & \dots & a_{nn} \end{pmatrix} \xrightarrow{\text{EROs}} \begin{pmatrix} 1 & a'_{12} & \dots & a'_{1k} & \dots & a'_{1n} \\ 0 & 1 & \dots & a'_{2k} & \dots & a'_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & \dots & a'_{kn} \\ 0 & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{pmatrix}$$

Now imagine we performed the same row reduction on the augmented matrix (\mathbf{Ay}) . We would still end up with $n - k$ rows with

zeros in the first n columns (the columns of \mathbf{A}):

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} & \cdots & a_{1n} & y_1 \\ a_{21} & a_{22} & \cdots & a_{2k} & \cdots & a_{2n} & y_2 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & \cdots & a_{kn} & y_k \\ a_{k+1,1} & a_{k+1,2} & \cdots & a_{k+1,k} & \cdots & a_{k+1,n} & y_{k+1} \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} & \cdots & a_{nn} & y_n \end{pmatrix} \xrightarrow{\text{EROs}} \begin{pmatrix} 1 & a'_{12} & \cdots & a'_{1k} & \cdots & a'_{1n} & y'_1 \\ 0 & 1 & \cdots & a'_{2k} & \cdots & a'_{2n} & y'_2 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & a'_{kn} & y'_k \\ 0 & 0 & \cdots & 0 & \cdots & 0 & y'_{k+1} \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & y'_n \end{pmatrix}$$

We know that if $y'_{k+1}, \dots, y'_n = 0$, we can solve this system by designating $n - k$ parameters for the variables x_{k+1}, \dots, x_n for which we have no information. However, notice what happens if any of the values y'_{k+1}, \dots, y'_n are nonzero. Then we have an expression of the form $0 = y'_i \neq 0$, which is nonsensical. Therefore, the only way we can solve this system is by requiring that $y'_{k+1}, \dots, y'_n = 0$. This is exactly the requirement that the rank of the augmented matrix equal k , the rank of the matrix \mathbf{A} by itself. If any of the y'_{k+1}, \dots, y'_n are nonzero, then the augmented matrix has one fewer row of zeros, so the rank of the augmented matrix would be greater than the rank of the original matrix. There are two ways to interpret this result. First, we require that the right hand side \mathbf{y} doesn't "mess up" our system by introducing a nonsensical expression. Second, if a row i in the matrix \mathbf{A} is linearly dependent on the other rows in \mathbf{A} , the corresponding values y_i must have the same dependency on the other values in \mathbf{y} . If so, when the row i is zeroed out during row reduction, the value y_i will also be zeroed out, avoiding any inconsistency.

□

4.8 Rank and Matrix Inverses

For a general nonhomogeneous system $\mathbf{Ax} = \mathbf{y}$ with $\mathbf{A} \in \mathbb{R}^{n \times n}$, we know that a unique solution only exists if $\text{rank}(\mathbf{A}) = \text{rank}([\mathbf{A} \ \mathbf{y}]) = n$. If $\text{rank}(\mathbf{A}) = n$, we know that \mathbf{A} can be transformed into reduced row form without generating any rows with all zero entries. We also know that if an inverse \mathbf{A}^{-1} exists for \mathbf{A} , we can use the inverse to uniquely solve for $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$. Putting these facts together, we can now definitely state necessary and sufficient conditions for matrix inversion:

An $n \times n$ matrix \mathbf{A} has an inverse if and only if $\text{rank}(\mathbf{A}) = n$.

4.9 *Summary*

We've shown in this chapter the tight connections between matrix inversion, solvability, and the rank of a matrix. We will use these concepts many times to understand the solution of linear systems. However, we've also argued that despite their theoretical importance, these concepts have limited practical utility for solving linear systems. For example, computing the rank of a matrix requires transforming the matrix into reduced echelon form. This requires the same computations as solving a linear system involving the matrix, so one would rarely check the rank of a coefficient matrix before attempting to solve a linear system. Instead, we will see rank emerge as a useful tool only when considering matrices by themselves in Part III of this course.

5

Applied Linear Regression in Matlab

Applied Linear Regression in Matlab

```
rng(2017); % set the Random Number Generator
x = linspace(1,15,100)';
y = 2*x + (x+randn(size(x))).^2;
```

Calculating Pseudoinverses

We saw before how the general linear model $y = X\beta + \epsilon$ can be solved for β by finding the pseudoinverse X^+ of the design matrix X . Then our estimate b for β can be found via matrix multiplication

$$b = X^+y$$

For example, given a 5x3 design matrix X

```
X = rand(5,3)
```

```
X =
    0.5978    0.6104    0.7408
    0.8466    0.9087    0.8940
    0.4716    0.8236    0.2554
    0.8076    0.2589    0.9636
    0.5071    0.3713    0.1046
```

and a 5x1 response vector y

```
y = rand(5,1)
```

```
y =
    0.8001
    0.6658
    0.7738
    0.3867
    0.3185
```

we can estimate the parameters b using the `pinv` function to calculate the pseudoinverse.

```
b = pinv(X)*y
```

```
b =
   -0.0260
    0.8084
    0.1795
```

Matlab also offers the backslash operator (`\`) to solve linear systems.

```
b = X \ y
```

```
b =
   -0.0260
    0.8084
```

0.1795

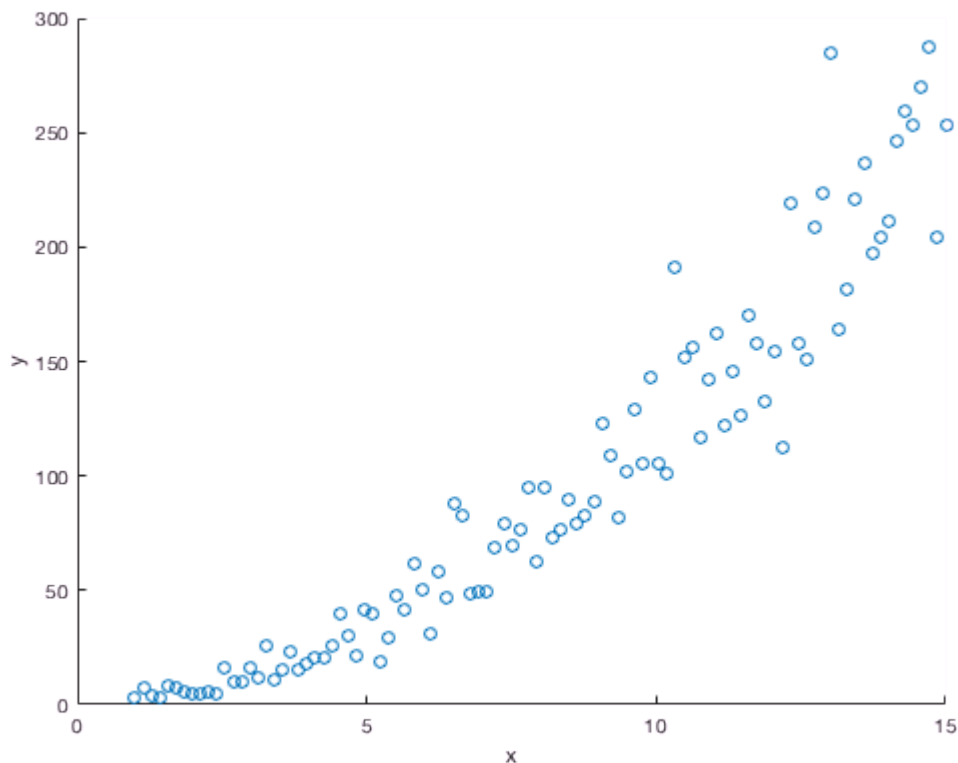
The backslash operator will always choose the appropriate method for solving the system based on the structure of the coefficient matrix. If the coefficient matrix is square and full rank, an efficient variant of Gaussian elimination is selected. If the coefficient matrix is not square, pseudoinversion is used to find the least-squared solution.

Simple Linear Regression

```
rng(2017); % set the Random Number Generator
x = linspace(1,15,100)';
y = 2*x + (x+randn(size(x))).^2;
```

Imagine you are given a set of 100 pairs of data (x, y) . We believe that y is some function of x ; can we identify this function using linear regression? If possible, we always start by plotting the data. In Matlab, we can use the `scatter` function to visualize individual points.

```
scatter(x,y)
xlabel('x')
ylabel('y')
```



Clearly there is some positive relationship between y and x . Let's begin by fitting the simplest linear model

$$y = \beta_1 x$$

In this case, we only have one parameter to estimate (β_1), and the design matrix \mathbf{X} has only a single column with the 100 x values.

```
X = [x];
```

Let's solve for the parameter estimates by pseudoinversion ($\mathbf{b} = \mathbf{X}^+ \mathbf{y}$), or, equivalently, using the backslash operator.

```
b = X \ y
```

```
b = 13.3924
```

Let's plot our model on the same plot as the original data. First, we plot the data.

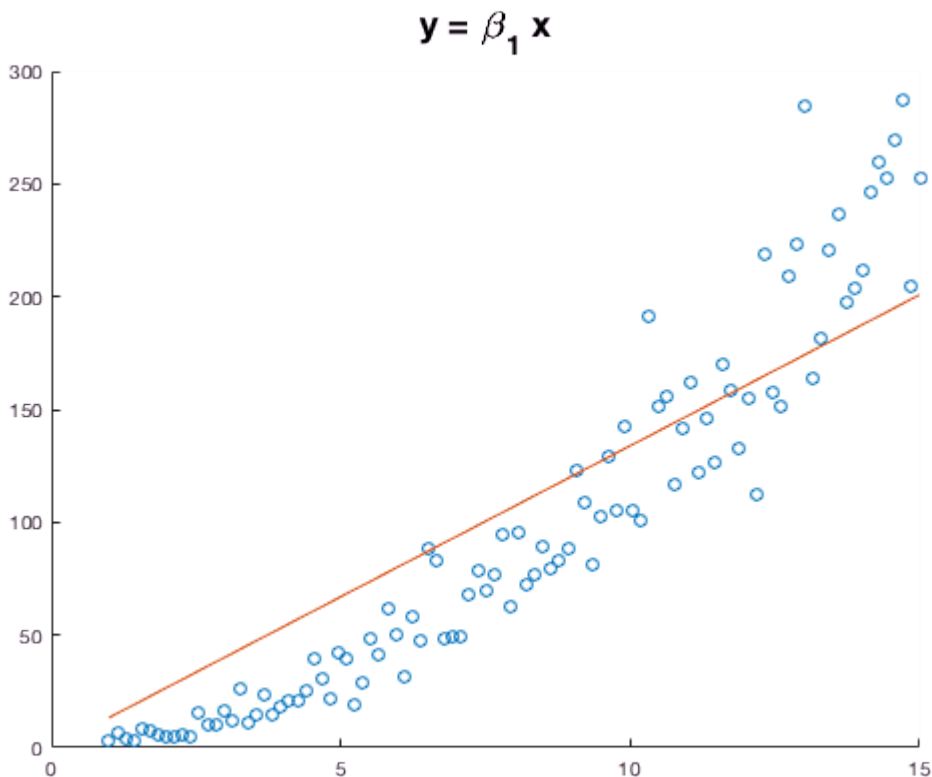
```
scatter(x,y)
```

Then, we tell Matlab to "hold on". This prevents Matlab from making a new figure for subsequent plots (until we tell it to "hold off").

```
hold on
```

Now we can plot a line with our model. The easiest way to multiply the design matrix by the parameter estimates.

```
plot(x, X*b)  
title('y = \beta_1 x', 'FontSize',18)  
hold off
```



This does not seem to be a great fit. Maybe we need an intercept term, making our model

$$y = \beta_0 + \beta_1 x$$

How do we add an intercept term? We want our final model to be of the form

$$\mathbf{y} = \mathbf{X} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

This tells us that the design matrix \mathbf{X} should be

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$$

The column of ones on the left allows the parameter β_0 to stand alone as an intercept. Let's construct this design matrix, solve for the parameters, and plot the new model.

```
X = [ones(size(x)) x];
```

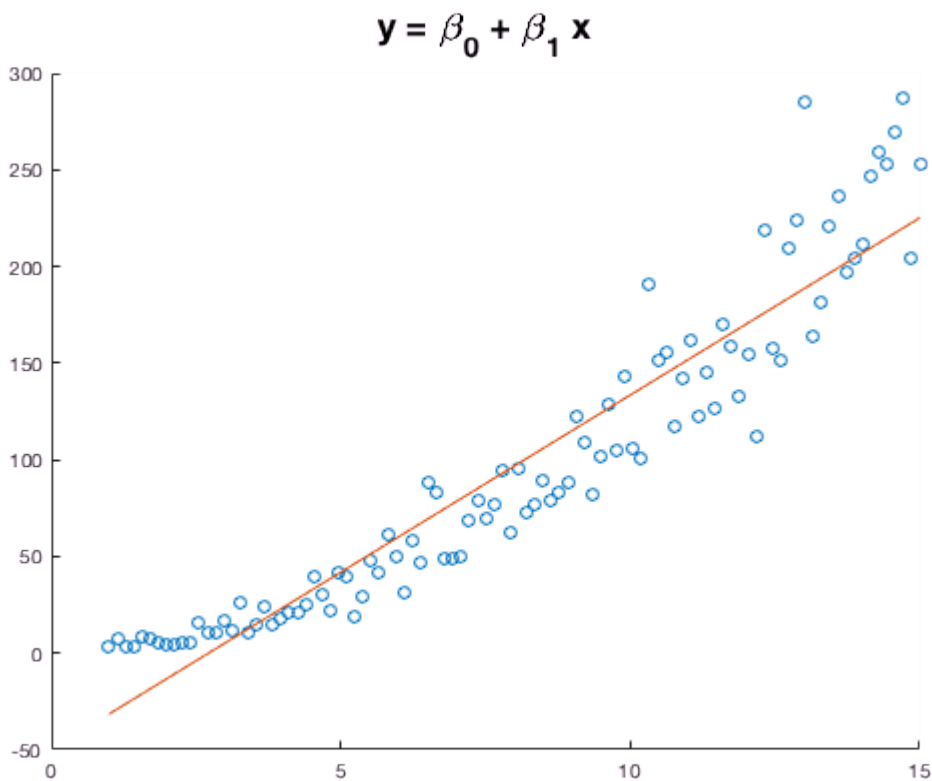
We use the `ones` function to create a column of ones. The `ones` function accepts either two values giving the dimensions (e.g. `ones(3,4)`) or the size of a similar matrix (`ones(size(x))`). Yes, in case you're wondering, there is a `zeros` function.

```
b = X \ y
```

```
b =  
-49.8172  
18.3332
```

The vector `b` now has two entries. The first is our estimate for β_0 , the second is the estimate for β_1 .

```
scatter(x,y)  
hold on  
plot(x, X*b)  
title('y = \beta_0 + \beta_1 x', 'FontSize',18)  
hold off
```



To plot the line representing our model, we could have constructed the point manually as $b(1) + b(2) \cdot x$. Notice two things. First, Matlab indexes vectors starting at one, so $b(1)$ is actually the estimate for β_0 , not for β_1 . Second, Matlab distinguishes between matrix multiplication ($*$) and element-by-element multiplication ($.*$).

This is still not a great fit. Based on the slight upward curve in the data, a quadratic model may be more appropriate. This model would be

$$y = \beta_0 + \beta_1 x + \beta_2 x^2$$

Fortunately, we can still fit quadratic polynomials (and, in fact, all polynomials) since **all polynomials are linear with respect to the parameters**. To fit a quadratic, we add a column to the design matrix that contains the square of each element in the vector x . (We will use the element-by-element exponentiation operator $.^$ here; matrix exponentiation is completely different.)

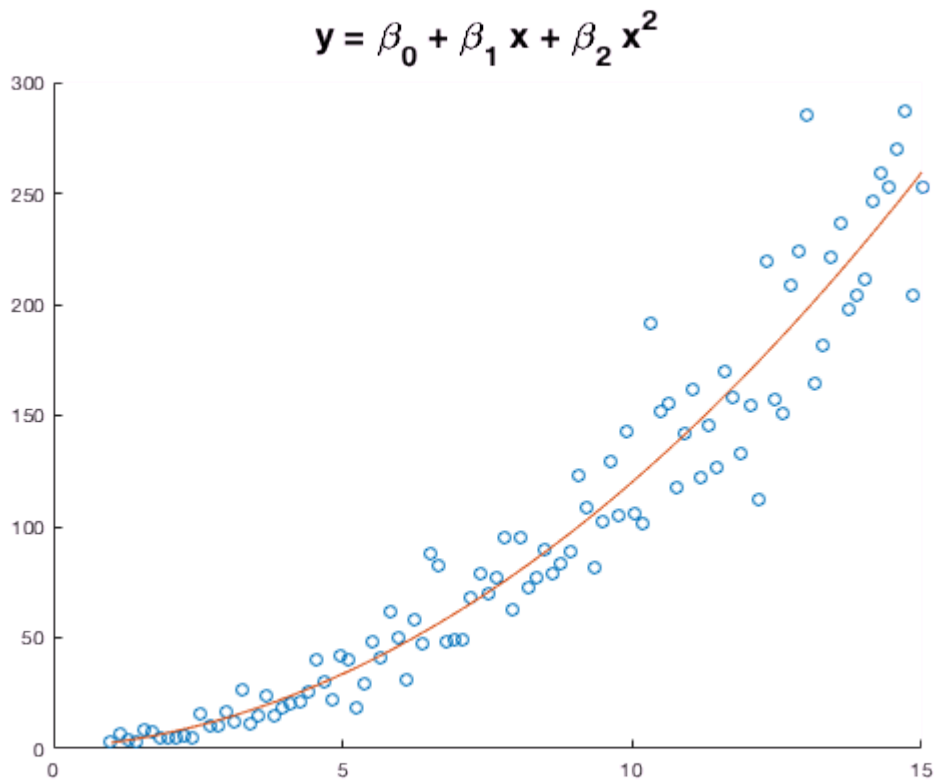
```
X = [ones(size(x)) x x.^2];
b = X \ y
```

```
b =
    0.3349
    1.3816
    1.0595
```

Our vector of estimates now has three entries corresponding to β_0 , β_1 , and β_2 . Let's plot the quadratic model.

```
scatter(x,y)
hold on
plot(x, X*b)
```

```
title('y = \beta_0 + \beta_1 x + \beta_2 x^2', 'FontSize',18)
hold off
```



That looks like a much better fit. These data appear to have a quadratic relationship.

Linear Regression with `fitlm`

Matlab offers an easier method for fitting linear models -- the `fitlm` function. To use `fitlm`, we start by placing our data in a Matlab table.

```
tbl = table(x,y);
head(tbl) % head shows only a few entries of large tables
```

```
ans = 8x2 table
      x      y
-----
      1    3.357
  1.1414    7.0774
  1.2828    3.6488
  1.4242    2.8731
  1.5657    8.2716
  1.7071    7.6963
  1.8485    5.3921
  1.9899    4.6634
```

Each variable given to the `table` function becomes a column in the table. The names of the columns are the names of the variables in the call to `table`. (If Matlab can't find the names, for example if you call `table(log(x), y+3)`, it names the columns "Var1", "Var2", etc.). You can change the names of an existing table by setting `tbl.Properties.VariableNames = {'name1', 'name2', ...}`.

The column names are important, as we refer to variables by these names when specifying our linear model.

Now that we have our data in a table, we call `fitlm`.

```
modell = fitlm(tbl, 'y ~ x')
```

```
modell =
```

```
Linear regression model:
```

```
y ~ 1 + x
```

```
Estimated Coefficients:
```

| | Estimate | SE | tStat | pValue |
|-------------|----------|---------|---------|------------|
| | ----- | ----- | ----- | ----- |
| (Intercept) | -49.817 | 5.7739 | -8.6281 | 1.1389e-13 |
| x | 18.333 | 0.64288 | 28.517 | 3.0026e-49 |

```
Number of observations: 100, Error degrees of freedom: 98
```

```
Root Mean Squared Error: 26.2
```

```
R-squared: 0.892, Adjusted R-Squared 0.891
```

```
F-statistic vs. constant model: 813, p-value = 3e-49
```

The first argument to `fitlm` is the table containing your data. The second argument is a string specifying the formula for the linear model. To specify a formula:

- Use a tilde to separate the response variable (y) from the input variables (x_1, x_2 , etc.).
- Do not include the names of the parameters; only the input variables. For example, we say ' $y \sim x$ ' not ' $y \sim b*x$ '. The `fitlm` function adds a parameter for each term in the model.
- `fitlm` always adds an intercept by default. To turn off this behavior, call `fitlm` with the '`intercept`' option set to false, i.e. `fitlm(..., 'intercept', false)`.
- The names of the response and input variables must match column names in the table.

The output from `fitlm` begins by stating the specified model. Again, no parameters appear. If an intercept was added, it appears as a 1 (just like the column of ones in the design matrix). Next is a table of the fit parameters -- an estimate of each coefficient in the model, including the intercept. Along with providing the numerical value of the coefficient, `fitlm` also reports the standard error for the estimate. The standard errors are calculated using the degrees of freedom left over in the data. So long as the number of observations is greater than the number of variables, `fitlm` will be able to provide errors for each coefficient.

Using the standard errors, we can construct confidence intervals around the parameter estimates. It is important to know if the confidence interval includes zero. If so, then we cannot reject the possibility that the true value of the parameter is zero. If the true value is zero, we should not draw any inferences regarding the estimated coefficient, since we cannot statistically distinguish the parameter from zero. To help identify statistically significant parameters, `fitlm` performs a modified t -test on the parameter estimates. Using the t -statistic ("tStat" in the `fitlm` output), a p -value is calculated. Only those estimates with p -values below out significance threshold (e.g. 0.05) should be interpreted.

`fitlm` also provides summary statistics on the model as a whole. The most commonly used metris is the coefficient of determination (R^2). Values near one are ideal; however, there is not widely accepted cutoff for "good" vs. "bad" R^2 values. Instead, you should consider the F -statistic. This test measures if the model performs better than a null model -- a model that discards the inputs and returns only a constant value.

Finally, `fitlm` provides the root mean squared error (RMSE). This measures the accuracy of the model and summarizes how closely estimates made with the model match the observed responses. To calculate the RMSE, we use the observed responses (y_i) and the predicted responses (\hat{y}_i) from our model on the corresponding inputs (x_i). Then the RMSE is

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^m (y_i - \hat{y}_i)^2}$$

The RMSE is the square root of the average penalty (the squared error) found during the least squares estimation.

Let's use `fitlm` to fit our quadratic model.

```
model2 = fitlm(tbl, 'y ~ x^2')
```

```
model2 =
```

```
Linear regression model:  
y ~ 1 + x + x^2
```

```
Estimated Coefficients:
```

| | Estimate | SE | tStat | pValue |
|-------------|----------|---------|----------|------------|
| (Intercept) | 0.33485 | 8.0944 | 0.041369 | 0.96709 |
| x | 1.3816 | 2.3069 | 0.59887 | 0.55065 |
| x^2 | 1.0595 | 0.14057 | 7.537 | 2.5514e-11 |

```
Number of observations: 100, Error degrees of freedom: 97  
Root Mean Squared Error: 20.9  
R-squared: 0.932, Adjusted R-Squared 0.931  
F-statistic vs. constant model: 667, p-value = 2.1e-57
```

For convenience, adding a quadratic term in the `fitlm` formula expands to include the linear term. This behavior can be disabled using the `'purequadratic'` option.

Exploratory Data Analysis with Linear Models

We can use linear modeling to identify factors that significantly affect an outcome. For example, let's use Matlab's builtin set of simulated hospital records.

```
load hospital_records  
hospital(1:10,:)
```

```
ans =
```

| | LastName | Sex | Age | Weight | Smoker | BloodPressure |
|---------|------------|--------|-----|--------|--------|---------------|
| YPL-320 | 'SMITH' | Male | 38 | 176 | true | 124 93 |
| GLI-532 | 'JOHNSON' | Male | 43 | 163 | false | 109 77 |
| PNI-258 | 'WILLIAMS' | Female | 38 | 131 | false | 125 83 |
| MIJ-579 | 'JONES' | Female | 40 | 133 | false | 117 75 |
| XLK-030 | 'BROWN' | Female | 49 | 119 | false | 122 80 |
| TFP-518 | 'DAVIS' | Female | 46 | 142 | false | 121 70 |
| LPD-746 | 'MILLER' | Female | 33 | 142 | true | 130 88 |
| ATA-945 | 'WILSON' | Male | 40 | 180 | false | 115 82 |
| VNL-702 | 'MOORE' | Male | 28 | 183 | false | 115 78 |
| LQW-768 | 'TAYLOR' | Female | 31 | 132 | false | 118 86 |

We want to identify any factors (sex, age, weight, or smoking) that increase blood pressure. We will build a linear model that combines these factors to predict blood pressure. First, let's average the systolic and diastolic blood pressures into a single value that we can use as our response variable.

```
hospital.meanBP = mean(hospital.BloodPressure,2);
hospital(1:10,:)
```

```
ans =
```

| | LastName | Sex | Age | Weight | Smoker | BloodPressure | meanBP |
|---------|------------|--------|-----|--------|--------|---------------|--------|
| YPL-320 | 'SMITH' | Male | 38 | 176 | true | 124 93 | 108.5 |
| GLI-532 | 'JOHNSON' | Male | 43 | 163 | false | 109 77 | 93 |
| PNI-258 | 'WILLIAMS' | Female | 38 | 131 | false | 125 83 | 104 |
| MIJ-579 | 'JONES' | Female | 40 | 133 | false | 117 75 | 96 |
| XLK-030 | 'BROWN' | Female | 49 | 119 | false | 122 80 | 101 |
| TFP-518 | 'DAVIS' | Female | 46 | 142 | false | 121 70 | 95.5 |
| LPD-746 | 'MILLER' | Female | 33 | 142 | true | 130 88 | 109 |
| ATA-945 | 'WILSON' | Male | 40 | 180 | false | 115 82 | 98.5 |
| VNL-702 | 'MOORE' | Male | 28 | 183 | false | 115 78 | 96.5 |
| LQW-768 | 'TAYLOR' | Female | 31 | 132 | false | 118 86 | 102 |

We added another column, "meanBP" to the `hospital` table. The second argument to `mean` tells the function to calculate the means along the second dimension. This gives us the mean for each row.

Now we build and fit a linear model using `fitlm`.

```
fitlm(hospital, 'meanBP ~ Sex + Age + Weight + Smoker')
```

```
ans =
```

```
Linear regression model:
meanBP ~ 1 + Sex + Age + Weight + Smoker
```

```
Estimated Coefficients:
```

| | Estimate | SE | tStat | pValue |
|-------------|------------|----------|-----------|------------|
| (Intercept) | 97.09 | 5.4093 | 17.949 | 2.7832e-32 |
| Sex_Male | 0.51095 | 2.0897 | 0.24451 | 0.80737 |
| Age | 0.058337 | 0.047726 | 1.2224 | 0.2246 |
| Weight | -0.0008026 | 0.039503 | -0.020317 | 0.98383 |
| Smoker_1 | 10.088 | 0.73786 | 13.672 | 3.6239e-24 |

```
Number of observations: 100, Error degrees of freedom: 95
Root Mean Squared Error: 3.41
R-squared: 0.683, Adjusted R-Squared 0.67
F-statistic vs. constant model: 51.2, p-value = 6.55e-23
```

Looking at the coefficient estimates, we find that only smoking is a significant predictor. The coefficient is 10.1, so smokers on average have a blood pressure that is 10 mmHg higher than non-smokers.

Be careful in interpreting the intercept of this model. The intercept is the response value when all inputs are zero. For this example, all zero inputs would be a female non-smoker of age zero that weighs nothing. Instead, we can find an average blood pressure for a 30 year old, 100 pound female nonsmoker as

$$BP = 97.1 + 0.51(0) + 0.058(30) - 0.00080(100) + 10.1(0) = 98.8$$

For a 50 year old, 260 pound male smoker, the estimate would be

$$BP = 97.1 + 0.51(1) + 0.058(50) - 0.00080(260) + 10.1(1) = 110.4$$

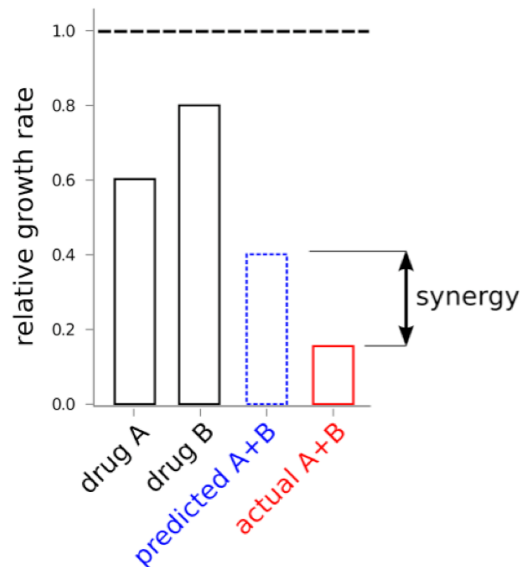
Interactions

```
rng(61112);

conc_A = rand(15,1);
conc_B = rand(15,1);
response = 3.*conc_A + 4.5.*conc_B + 12.5.*conc_A.*conc_B + 0.5.*randn(15,1);
drug = table(response, conc_A, conc_B);
```

Linear models can test for significant interactions between variables. To include interactions between variables x_i and x_j , we add a new term that is the product of the two variables ($x_i x_j$). We choose multiplication to combine the variables since this interaction term will only be nonzero when **both** variables are nonzero; thus, the interaction is suppressed when either variable is not observed.

Interactions are commonly used to test for *synergy* between drugs. For drugs to synergize, the observed response must be greater than the expected additive effects of the two drugs alone. (If the drugs response is lower than expected, the drugs are *antagonistic*.)



We modeled the response to the drug (y) as a linear combination of the two drugs independently plus an interaction term:

$$y = \beta_0 + \beta_1[c_1] + \beta_2[c_2] + \beta_{12}[c_1][c_2]$$

where $[c_1]$ is the concentration of drug 1 (or log-concentration, as dose responses are often log-linear), and $[c_2]$ is the concentration of drug 2. The intercept (β_0) is the basal response of the assay when neither drug is added. The parameter β_{12} corresponds to the strength of the interaction. If we fit the above model and find a significant, nonzero β_{12} term, we can conclude that some nonlinear interaction exists between the drugs.

Let's test for synergy with the following drug response data (with concentration is log scale).

```
drug
```

```
drug = 15x3 table
```

| response | conc_A | conc_B |
|----------|----------|----------|
| 2.9375 | 0.78812 | 0.056482 |
| 6.9199 | 0.48957 | 0.51783 |
| 2.6542 | 0.53637 | 0.075296 |
| 9.3316 | 0.32535 | 0.88539 |
| 8.567 | 0.82917 | 0.34905 |
| 16.306 | 0.98211 | 0.80744 |
| 1.6428 | 0.63671 | 0.017543 |
| 5.6819 | 0.070839 | 0.93119 |
| 2.5009 | 0.54116 | 0.079158 |
| 3.9707 | 0.89685 | 0.029386 |
| 2.2057 | 0.62384 | 0.043192 |
| 13.62 | 0.60518 | 0.98991 |
| 14.015 | 0.892 | 0.74739 |
| 4.924 | 0.37901 | 0.39126 |
| 4.993 | 0.15946 | 0.6222 |

In `fitlm` formulas, interactions are specified by combining the two variable names with a colon (`x1:x2`). As a shortcut, we can add both linear terms and an interaction term using the `*` operator. The formula `response ~ x1 + x2 + x1:x2` is equivalent to the formula `response ~ x1 * x2`.

First, let's fit a linear model without interaction.

```
model_lin = fitlm(drug, 'response ~ conc_A + conc_B')
```

```
model_lin =
```

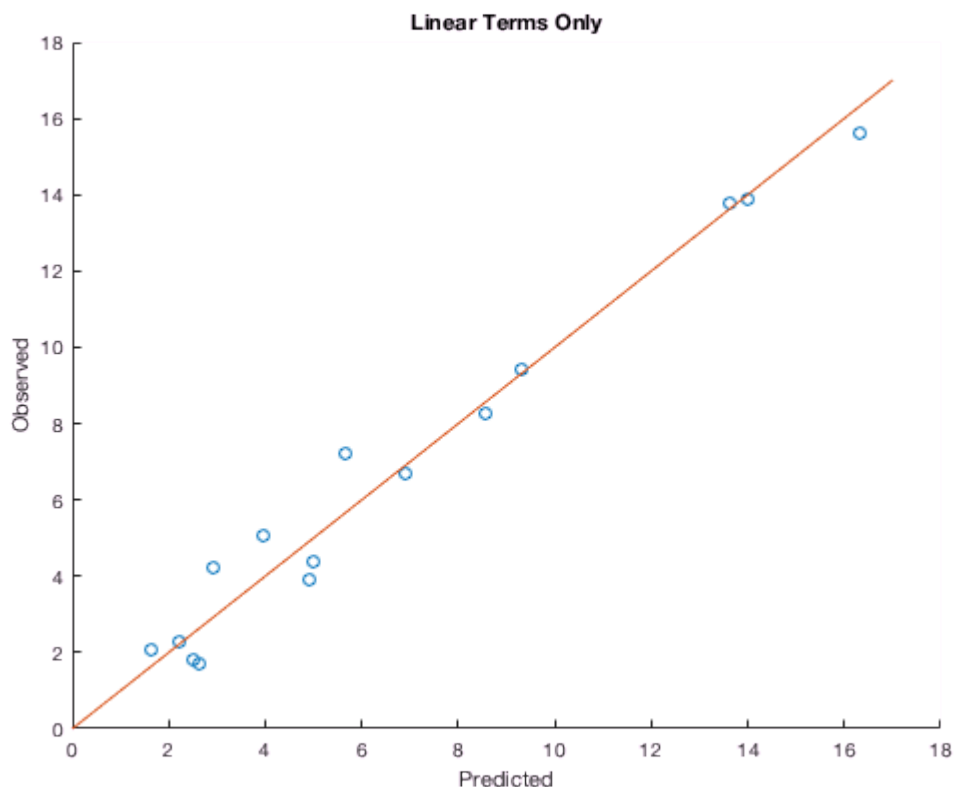
```
Linear regression model:  
response ~ 1 + conc_A + conc_B
```

```
Estimated Coefficients:  
              Estimate      SE      tStat      pValue  
-----  
(Intercept)  -5.0689    0.69951   -7.2464   1.0197e-05  
conc_A        10.887    0.89216   12.202   4.0126e-08  
conc_B        12.378    0.64862   19.083   2.4105e-10
```

```
Number of observations: 15, Error degrees of freedom: 12  
Root Mean Squared Error: 0.861  
R-squared: 0.972, Adjusted R-Squared 0.967  
F-statistic vs. constant model: 205, p-value = 5.28e-10
```

We would like to plot the data and our model. Unfortunately, the multiple input variables makes visualization difficult. Instead, we can get a sense for our model's accuracy by plotting the observed response values vs. the model's predicted response values. (We use the `predict` function to find the predicted values from a fitted model and the original data table. We also add a diagonal line corresponding to perfect correlation.)

```
scatter(drug.response, predict(model_lin, drug))  
xlabel('Predicted');  
ylabel('Observed');  
title('Linear Terms Only');  
hold on  
plot([0 17], [0 17])  
hold off
```

This isn't a bad model fit, but it can clearly be improved. Maybe the interaction terms will help.

```
model_int = fitlm(drug, 'response ~ conc_A * conc_B')
```

```
model_int =
```

```
Linear regression model:  
response ~ 1 + conc_A*conc_B
```

```
Estimated Coefficients:
```

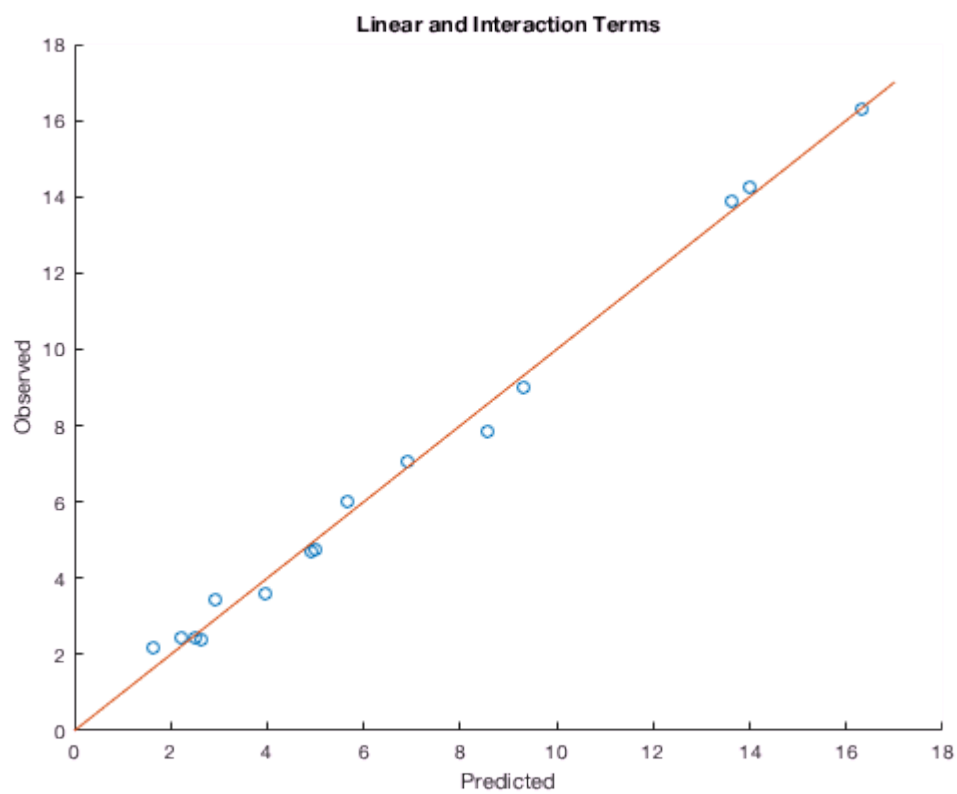
| | Estimate | SE | tStat | pValue |
|---------------|----------|---------|--------|------------|
| (Intercept) | -0.94009 | 0.71872 | -1.308 | 0.21755 |
| conc_A | 4.5727 | 1.0626 | 4.3034 | 0.0012489 |
| conc_B | 6.44 | 0.96668 | 6.662 | 3.5531e-05 |
| conc_A:conc_B | 9.4829 | 1.4631 | 6.4815 | 4.5414e-05 |

```
Number of observations: 15, Error degrees of freedom: 11  
Root Mean Squared Error: 0.409  
R-squared: 0.994, Adjusted R-Squared 0.992  
F-statistic vs. constant model: 618, p-value = 1.55e-12
```

The coefficient of the interaction term is statistically significant at $p < 0.05$. The coefficient is also positive, indicating synergy between the drugs. Let's see if taking the synergy into account improves our predictions.

```
scatter(drug.response, predict(model_int,drug))  
xlabel('Predicted');  
ylabel('Observed');
```

```
title('Linear and Interaction Terms');  
hold on  
plot([0 17], [0 17])  
hold off
```



The model with the interaction term is a better predictor of the drug response.

6

Optimization, Convexity, and Hyperplanes

We formulated the least squares method and linear regression as optimization problems. Our goal was to minimize the sum of the squared errors by choosing parameters for the linear model. Optimization problems have enormous utility in data science, and most model fitting techniques are cast as optimizations. In this chapter, we will develop a general framework for describing and solving several classes of optimization problems. We begin by reviewing the fundamentals of optimization. Next, we discuss convexity, a property that greatly simplifies the search for optimal solutions. Finally we derive vector expressions for common geometric constructs and show how linear systems give rise to convex problems.

6.1 Optimization

Optimization is the process of minimizing or maximizing a function by selecting values for a set of variables or parameters (called *decision variables*). If we are free to choose any values for the decision variables, the optimization problem is *unconstrained*. If our solutions must obey a set of constraints, the problem is a *constrained optimization*. In constrained optimization, any set of values for the decision variables that satisfies the constraints is called a *feasible solution*. The goal of constrained optimization is to select the “best” feasible solution.

Optimization problems are formulated as either minimizations or maximizations. We don’t need to discuss minimization and maximization separately, since minimizing $f(x)$ is equivalent to maximizing $-f(x)$. Any algorithm for minimizing can be used for maximizing by multiplying the objective by -1 , and vice versa. For the rest of this chapter, we’ll talk about minimizing functions. Keep in mind that everything we discuss can be applied to maximization problems by switching the sign of the objective.

During optimization we search for minima. A minimum can either be *locally* or *globally* minimal. A global minimum is has the

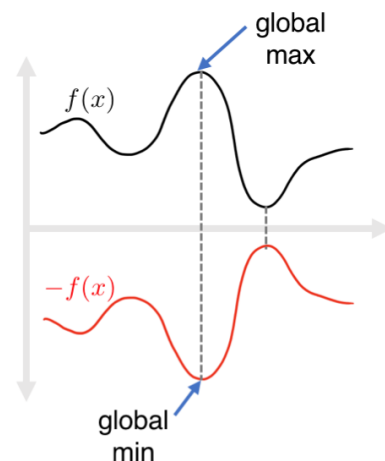


Figure 6.1: The maximum of a function $f(x)$ can be found by minimizing $-f(x)$.

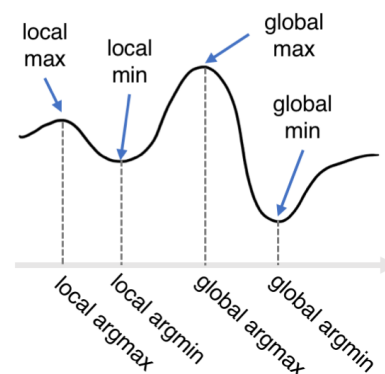


Figure 6.2: Minima and maxima of a function can be local or global.

smallest objective value of any feasible solution. A local minimum has the smallest objective value for any of the feasible solutions in the surrounding area. The input to a function that yields the minimum is called the *argmin*, since it is the argument to the function that gives the minimum. Similarly, the *argmax* of a function is the input that gives the function's maximum. Consider the function $f(x) = 3 + (x - 2)^2$. This function has a single minimum, $f(2) = 3$. The minimum is 3, while the argmin is $x = 2$, the value of the decision variable at which the minimum occurs. For optimization problems, the minimum (or maximum) is called the *optimal objective value*. The argmin (or argmax) is called the *optimal solution*.

6.1.1 Unconstrained Optimization

You already know how to solve unconstrained optimization problems in a single variable: set the derivative to the function equal to zero and solve. This method of solution relies on the observation that both maxima and minima occur when the slope of a function is zero. However, it is important to remember that not all roots of the derivative are maxima or minima. Inflection points (where the derivative changes sign) also have derivatives equal to zero. You must always remember to test the root of the derivative to see if you've found a minimum, maximum, or inflection point. The easiest test involves the sign of the second derivative. If the second derivative at the point is positive, you've found a minimum. If it's negative, you've found a maximum. If the second derivative is zero, you've found an inflection point.

A similar approach works for optimizing multivariate functions. In this case one solves for points where the gradient is equal to zero, checking that you've not found an inflection point (called "saddle points" in higher dimensions).

6.1.2 Constrained Optimization

Constrained optimization problems cannot be solved by finding roots of the derivatives of the objective. Why? It is possible that the minima or maxima of the unconstrained problem lie outside the feasible region of the constrained problem. Consider our previous example of $f(x) = 3 + (x - 2)^2$, which we know has an argmin at $x = 2$. Say we want to solve the constrained problem

$$\min f(x) = 3 + (x - 2)^2 \quad \text{s.t.} \quad x \leq 1$$

The root of the derivative of f is still at $x = 2$, but values of x greater than one are not feasible. From the graph we can see that the minimum feasible value occurs at $x = 1$. The value of the derivative at $x = 1$ is -2 , not zero.

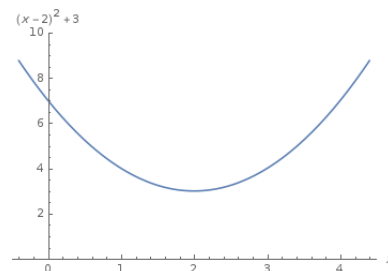


Figure 6.3: The function $f(x) = 3 + (x - 2)^2$ has a minimum of $f = 3$ at argmin $x = 2$.

Any point where the derivative of a function equals zero is called an *extreme point* or *extremum*. Setting the derivative of a function equal to zero and solving for the extrema is called *extremizing* a function.

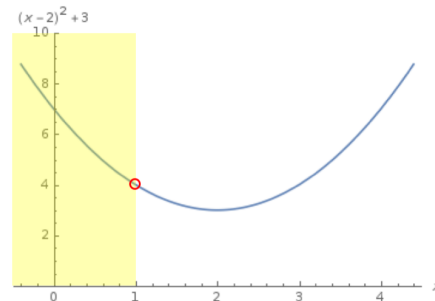


Figure 6.4: The yellow region is the feasible space ($x \leq 1$). The global argmin occurs at $x = 1$. The derivative of the function is not zero at this point.

In general, constrained optimization is a challenging field. Finding global optima for constrained problems is an unsolved area or research, one which is beyond the scope of this course. However, there are classes of problems that we can solve to optimality using the tools of linear algebra. These problems form the basis of many advanced techniques in data science.

6.2 Convexity

Many “solvable” optimization problems rely on a property called *convexity*. Both sets and functions can be convex.

6.2.1 Convex sets

A set of points is *convex* if given any two points in the set, the line segment connecting these points lies entirely in the set. You can move from any point in the set to any other point in the set without leaving the set. Circles, spheres, and regular polygons are examples of convex sets.

To formally define convexity, we construct the line segment between any two points in the set.

Definition. A set S is convex if and only if given any $\mathbf{x} \in S$ and $\mathbf{y} \in S$ the points $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}$ are also in S for all scalars $\lambda \in [0, 1]$.

The expression $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}$ is called a *convex combination* of \mathbf{x} and \mathbf{y} . A convex combination of two points contains all points on the line segment between the two points. To see why, consider the 1-dimensional line segment between points 3 and 4.

$$\lambda(3) + (1 - \lambda)(4) = 4 - \lambda, \quad \lambda \in [0, 1]$$

When $\lambda = 0$, the value of the combination is 4. As λ moves from 0 to 1, the value of the combination moves from 4 to 3, covering all values in between.

Convex combinations work in higher dimensions as well. The convex combination of the vectors $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is

$$\lambda \begin{pmatrix} 1 \\ 0 \end{pmatrix} + (1 - \lambda) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda \\ 1 - \lambda \end{pmatrix}$$

The combination goes from the first point $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ when $\lambda = 0$ to the second point $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ when $\lambda = 1$. Halfway in between, $\lambda = 1/2$ and

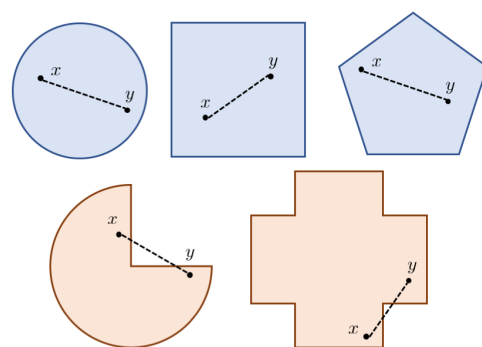


Figure 6.5: The blue shapes are convex. The red shapes are not convex.

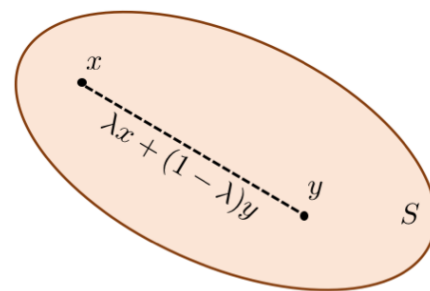


Figure 6.6: The segment connecting x and y can be defined as $\lambda x + (1 - \lambda)y$ for $\lambda \in [0, 1]$.

the combination is $\begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$, which is midway along the line connecting $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Sometimes it is helpful to think of a convex combination as a weighted sum of \mathbf{x} and \mathbf{y} . The weighting (provided by λ) moves the combination linearly from \mathbf{y} to \mathbf{x} as λ goes from 0 to 1.

6.2.2 Convex functions

There is a related definition for *convex functions*. This definition formalizes our visual idea of convexity (lines that curve upward) and concavity (lines that curve downward).

Definition 1. A function f is convex if and only if

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}), \quad \lambda \in [0, 1]$$

This definition looks complicated, but the intuition is simple. If we plot a convex (upward curving) function, any chord – a segment drawn between two points on the line – should lie above the line. We can define the chord between any two points on the line, say $f(\mathbf{x})$ and $f(\mathbf{y})$ as a convex combination of these points, i.e. $\lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y})$. This is the right hand side of the above definition. For convex functions, we expect this cord to be greater than or equal to the function itself over the same interval. The interval is the segment from \mathbf{x} to \mathbf{y} , or the convex combination $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}$. The values of the function over this interval are therefore $f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y})$, which is the left hand side of the definition.

6.2.3 Convexity in Optimization

Why do we care about convexity? In general, finding local optima during optimization is easy; just pick a feasible point and move downward (during minimization) until you arrive at a local minimum. The truly hard part of optimization is finding global optima. How can you be assured that your local optimum is a global optimum unless you try out all points in the feasible space?

Fortunately, convexity solves the local vs. global challenge for many important problems, as we see with the following theorem.

Theorem. When minimizing a convex function over a convex set, all local minima are global minima.

Convex functions defined over convex sets must have a special shape where no *strictly* local minima exist. There can be multiple local minima, but all of these local minima must have the same value (which is the global minimum).

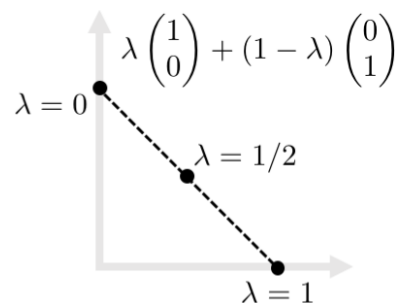


Figure 6.7: A convex combination in 2D.

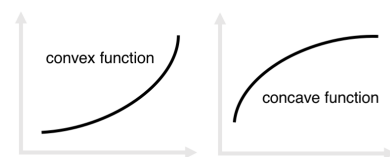


Figure 6.8: Convex functions curve upward. Concave functions curve downward.

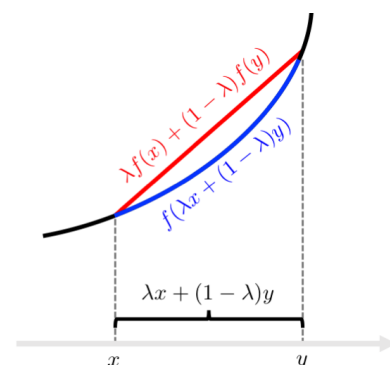


Figure 6.9: The chord connecting any two points of a convex function (red) lies above the function (blue).

All local minima are *less than or equal to* the global minimum. Strictly local minima must be *less than* the global minimum.

Let's prove that all local minima are global minima when minimizing a convex function over a convex set.

Proof. Suppose the convex function f has a local minimum at \mathbf{x}' that is not the global minimum (which is at \mathbf{x}^*). By the convexity of f ,

$$f(\lambda \mathbf{x}' + (1 - \lambda) \mathbf{x}^*) \leq \lambda f(\mathbf{x}') + (1 - \lambda) f(\mathbf{x}^*)$$

Since \mathbf{x}' is at a local, but not global, minimum, we know that $f(\mathbf{x}') > f(\mathbf{x}^*)$. If we replace $f(\mathbf{x}^*)$ on the right hand side by the larger quantity $f(\mathbf{x}')$, the inequality (\leq) becomes a strict inequality ($<$). (Even if both sides were equal, adding a small amount to the right hand side would still make it larger.) We now have

$$f(\lambda \mathbf{x}' + (1 - \lambda) \mathbf{x}^*) < \lambda f(\mathbf{x}') + (1 - \lambda) f(\mathbf{x}')$$

which, by simplifying the right hand side, becomes

$$f(\lambda \mathbf{x}' + (1 - \lambda) \mathbf{x}^*) < f(\mathbf{x}')$$

This statement says that the value of the function f on any point on the line segment from \mathbf{x}' to \mathbf{x}^* is less than the value of the function at \mathbf{x}' . If this is true, we can find a point arbitrarily close to \mathbf{x}' that is below our supposed local minimum $f(\mathbf{x}')$. Clearly, $f(\mathbf{x}')$ cannot be a local minimum if we can find a lower point arbitrarily closer to it. Our conclusion contradicts our original supposition. No local minimum can exist that are not equal to the global minimum. \square

For a simpler, yet less intuitive argument, let $\lambda = 1$. Then the inequality becomes $f(\mathbf{x}') < f(\mathbf{x}')$, which is nonsense.

The previous proof seemed to rely only on the convexity of the objective function, not on the convexity of the solution set. The role of convexity of the set is hidden. When we make an argument about a line drawn from the local to the global minimum, we assume that all the points on the line are feasible. Otherwise, it does not matter if they have a lower objective than the local minimum, since they would not be allowed. By assuming the solution set is convex, we are assured that any point on this line is also feasible.

6.2.4 Convexity of Linear Systems

This course focuses on linear functions and systems of linear equations. It would be enormously helpful if linear functions and the solution set of linear systems were convex. Then we can look for local optima during optimization and know that we've found global optima.

Let's first prove the convexity of linear functions. For a function to be convex, we require that a line segment connecting any two points in the line lie above or on the line. For linear functions, this is intuitively true. The line segment connecting any two points is the line

itself, so it always lies on the line. As a more formal argument, we describe a linear function as the product between a vector of coefficients \mathbf{c} and \mathbf{x} , i.e. $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$. Let's start with the values of the function over the range spanned by arbitrary points \mathbf{x} and \mathbf{y} . The segment of the domain corresponds to the convex combination $\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}$. The values of the function over this interval are

$$\begin{aligned} f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) &= \mathbf{c}^T (\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \\ &= \mathbf{c}^T \lambda \mathbf{x} + \mathbf{c}^T (1 - \lambda) \mathbf{y} \\ &= \lambda \mathbf{c}^T \mathbf{x} + (1 - \lambda) \mathbf{c}^T \mathbf{y} \\ &= \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}) \end{aligned}$$

which satisfies the definition of convexity: $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$.

Now let's turn to a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. We want to show that the set of all solutions for this system (the *solution space*) is convex. Let's assume we have two points in the solution space, \mathbf{x} and \mathbf{y} . Since \mathbf{x} and \mathbf{y} are solutions, we know that $\mathbf{A}\mathbf{x} = \mathbf{b}$ and $\mathbf{A}\mathbf{y} = \mathbf{b}$. If the solution set is convex, any point in the convex combination of \mathbf{x} and \mathbf{y} is also a solution.

$$\begin{aligned} \mathbf{A}(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) &= \mathbf{A} \lambda \mathbf{x} + \mathbf{A} (1 - \lambda) \mathbf{y} \\ &= \lambda \mathbf{A} \mathbf{x} + (1 - \lambda) \mathbf{A} \mathbf{y} \\ &= \lambda \mathbf{b} + (1 - \lambda) \mathbf{b} \\ &= \mathbf{b} \end{aligned}$$

Since $\mathbf{A}(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) = \mathbf{b}$, we know that all points on the line between \mathbf{x} and \mathbf{y} are solutions, so the solution set is convex.

6.3 Geometry of Linear Equations

Why do linear systems have convex solution spaces? Before answering, we should understand the shape of individual equations (rows) in the systems $\mathbf{A}\mathbf{x} = \mathbf{b}$. The equation corresponding to the i^{th} row is

$$\mathbf{A}(i, :) \cdot \mathbf{x} = b_i$$

which we will simplify by using the notation

$$\mathbf{a} \cdot \mathbf{x} = b$$

where \mathbf{a} and \mathbf{x} are vectors and the value b is a scalar. In two dimensions, this expression defines a line

$$a_1 x_1 + a_2 x_2 = b$$

By convention, all vectors are column vectors, including \mathbf{c} ; this requires a transposition to be conformable for multiplication by \mathbf{x} .

Following the conventions of the optimization field, we call the right hand side of linear systems the column vector \mathbf{b} , not \mathbf{y} as we have said previously.

The above representation of a line is the *standard form*, which differs from the *slope-intercept* form you remember from algebra ($y = mx + b$). It seems intuitive why the slope-intercept form is a line; a change in x produces a corresponding change $m\Delta x$ in y , with an intercept b when $x = 0$. What is the analogous reasoning for why $\mathbf{a} \cdot \mathbf{x} = b$ is a line?

First, we note that the vector \mathbf{a} always point perpendicular, or *normal* to the line. For the horizontal line $y = 3$, the vector $\mathbf{a} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ points vertically. For the vertical line $x = 3$, the vector $\mathbf{a} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ point horizontal. For the line $x_1 + x_2 = 1$, $\mathbf{a} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, which is still perpendicular to the original line.

To help visualize the equation $\mathbf{a} \cdot \mathbf{x} = b$, we need to know the length of \mathbf{a} . The easiest solution is to normalize \mathbf{a} by dividing both sides of the equation by the norm of \mathbf{a} .

$$\frac{1}{\|\mathbf{a}\|} \mathbf{a} \cdot \mathbf{x} = b / \|\mathbf{a}\|$$

If we use our previous notation of $\hat{\mathbf{a}}$ for the normalized form of \mathbf{a} and define scalar $d = b / \|\mathbf{a}\|$, we have

$$\hat{\mathbf{a}} \cdot \mathbf{x} = d$$

We know that $\hat{\mathbf{a}}$ is a unit vector normal to the line. What is the meaning of d ? Let's compute the dot product $\hat{\mathbf{a}} \cdot \mathbf{x}$ using an arbitrary point \mathbf{x} on the line.

$$d = \hat{\mathbf{a}} \cdot \mathbf{x} = \|\hat{\mathbf{a}}\| \|\mathbf{x}\| \cos \theta = \|\mathbf{x}\| \cos \theta$$

Thus, d is the projection of the magnitude of \mathbf{x} onto the normal vector. For any point on the line, this projection is always the same length – the distance between the origin and the nearest point on the line. Conversely, a line is the set of all vectors whose projection against a vector $\hat{\mathbf{a}}$ is a constant distance (d) from the origin.

The same interpretation follows in higher dimensions. In 3D, the expression $\hat{\mathbf{a}} \cdot \mathbf{x} = d$ defines a plane with normal vector $\hat{\mathbf{a}}$ at a distance d from the origin. This definition fits with the algebraic definition of a plane that you may have seen previously: $a_1x_1 + a_2x_2 + a_3x_3 = d$. In higher dimensions (four or more), this construct is called a *hyperplane*.

Remember that when analyzing an expression of the form $\hat{\mathbf{a}} \cdot \mathbf{x} = d$, the constant on the right hand side (d) is only equal to the distance between the line and the origin if the vector $\hat{\mathbf{a}}$ is normalized. For example, the line

$$3x_1 + 4x_2 = 7$$

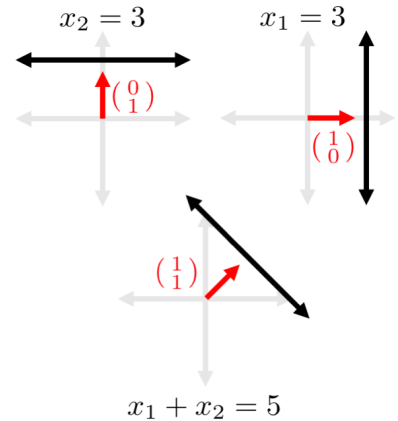


Figure 6.10: The vector \mathbf{a} is always normal (perpendicular) to the line $\mathbf{a} \cdot \mathbf{x} = b$.

The equation $\hat{\mathbf{a}} \cdot \mathbf{x} = d$ is called the Hesse normal form of a line, plane, or hyperplane.

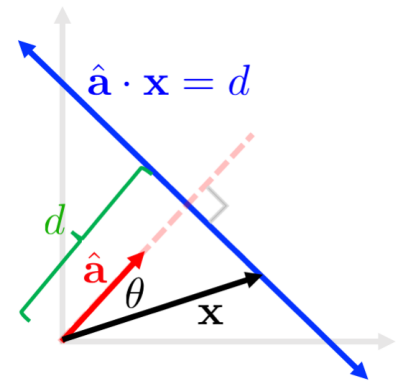


Figure 6.11: A line is the set of all points \mathbf{x} whose projection onto $\hat{\mathbf{a}}$ is the distance d .

has coefficient vector $\mathbf{a} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$, which is not normalized. To normalize \mathbf{a} , we divide both sides by $\|\mathbf{a}\| = \sqrt{3^2 + 4^2} = 5$, yielding

$$\frac{3}{5}x_1 + \frac{4}{5}x_2 = \frac{7}{5}$$

Now we can say that the distance between this line and the origin is $7/5$.

6.4 Geometry of Linear Systems

The equation $\hat{\mathbf{a}} \cdot \mathbf{x} = d$ defines a hyperplane. It is also a single row in the linear system $\mathbf{Ax} = \mathbf{b}$. What does the entire system of equations look like? First, let's consider a set of three equations in two dimensions (so we can visualize them as lines). Solutions to $\mathbf{Ax} = \mathbf{b}$ are points of intersection of all three equations. If the lines are parallel, no solutions exist. If the lines all intersect at one point, there is a unique solution. If the lines are *colinear* (all fall upon the same line), infinitely many solutions exist. Note that these are the only options – zero, one, or infinitely many solutions, just as predicted by the grand solvability theorem. It is impossible to draw three straight lines that intersect in only two places.

If linear systems $\mathbf{Ax} = \mathbf{b}$ are a set of intersecting lines in 2D, what do the inequalities $\mathbf{Ax} \leq \mathbf{b}$ represent? Each inequality states that the projection of \mathbf{x} onto the normal vector \mathbf{a} must be less than d . These points form a *half-plane* – all the points on one side of a hyperplane. The system $\mathbf{Ax} \leq \mathbf{b}$ has a solution space that is the overlap of multiple half-planes (one for each row in \mathbf{A}). As we proved earlier, this solution set is a convex set.

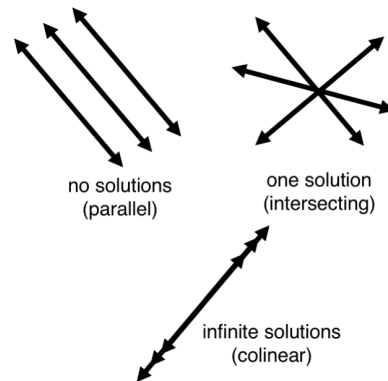


Figure 6.12: A system of linear equations can have zero, one, or infinitely many points of intersection.

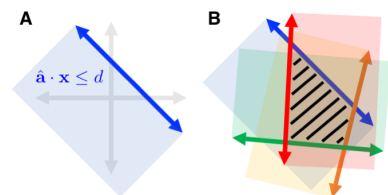


Figure 6.13: **A.** A single inequality defines a half-plane. **B.** Multiple half-planes intersect to form a convex solution set for the system $\mathbf{Ax} \leq \mathbf{b}$.

γ

Vector Spaces, Span, and Basis

7.1 Vector Spaces

Vector spaces are collections of vectors. The most common spaces are \mathbb{R}^2 , \mathbb{R}^3 , and \mathbb{R}^n – the spaces that include all 2-, 3-, and n -dimensional vectors. We can construct *subspaces* by specifying only a subset of the vectors in a space. For example, the set of all 3-dimensional vectors with only integer entries is a subspace of \mathbb{R}^3 .

Remember that \mathbb{R}^2 is not a subspace of \mathbb{R}^3 ; they are completely separate, non-overlapping spaces.

7.2 Span

A set of m vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ is said to *span* a space V if any vector \mathbf{u} in V can be written as a linear combination of the vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$. This is equivalent to saying there exist scalars a_1, a_2, \dots, a_m such that

$$\mathbf{u} = a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \dots + a_m \mathbf{v}_m$$

Writing a vector \mathbf{u} as a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ is called *decomposing* \mathbf{u} over $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$. If a set of vectors spans a space, they can be used to decompose any other vector in the space.

We've already seen vector composition using a special set of vectors $\hat{\mathbf{e}}_j$, the unit vectors with only one nonzero entry at element j . For example, the vector

$$\begin{pmatrix} -2 \\ 4 \\ 5 \end{pmatrix} = -2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 4 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 5 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Thus the vectors $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$ spans \mathbb{R}^3 . In general, the set of vectors $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \dots, \hat{\mathbf{e}}_n$ spans the space \mathbb{R}^n . Are these the only sets of vectors that span these spaces?

No, there are infinitely many sets of vectors that span each space. Consider the vectors $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$. We can show that these vectors

span \mathbb{R}^2 by showing that any vector \mathbf{u} in \mathbb{R}^2 can be written as a linear combination

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = a_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + a_2 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Finding the coefficients a_1 and a_2 is akin to solving the system of linear equations

$$a_1 - a_2 = u_1$$

$$a_1 + a_2 = u_2$$

which has the unique solution

$$a_1 = \frac{u_1 + u_2}{2}, \quad a_2 = \frac{u_2 - u_1}{2}$$

To demonstrate, let $\mathbf{u} = \begin{pmatrix} -2 \\ 4 \end{pmatrix}$. Then $a_1 = 1$ and $a_2 = 3$. Then

$$a_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + a_2 \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3 \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 - 3 \\ 1 + 3 \end{pmatrix} = \begin{pmatrix} -2 \\ 4 \end{pmatrix}$$

We've shown that there are least two sets of vectors that span \mathbb{R}^2 , $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$ and $\left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\}$. How can we say there are infinitely many? If vectors \mathbf{v}_1 and \mathbf{v}_2 span a space V , then the vectors $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$ also span V for any scalars k_1 and k_2 . To prove this, remember that any vector \mathbf{u} can be decomposed onto \mathbf{v}_1 and \mathbf{v}_2 , i.e.

$$\begin{aligned} \mathbf{u} &= a_1\mathbf{v}_1 + a_2\mathbf{v}_2 \\ &= \frac{a_1}{k_1}(k_1\mathbf{v}_1) + \frac{a_2}{k_2}(k_2\mathbf{v}_2) \end{aligned}$$

Since (a_1/k_1) and (a_2/k_2) are simply scalars, we've shown that \mathbf{u} can be decomposed onto the vectors $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$. Therefore, $k_1\mathbf{v}_1$ and $k_2\mathbf{v}_2$ must also span \mathbb{R}^2 . For example, the vectors $\begin{pmatrix} 3 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$ are scalar multiples of $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The former two vectors must

therefore span \mathbb{R}^2 , so we can decompose the vector $\begin{pmatrix} -2 \\ 4 \end{pmatrix}$ onto them.

$$\begin{pmatrix} -2 \\ 4 \end{pmatrix} = -\frac{2}{3} \begin{pmatrix} 3 \\ 0 \end{pmatrix} - 8 \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}$$

Similarly, if \mathbf{v}_1 and \mathbf{v}_2 span a space V , the vectors \mathbf{v}_1 and $(\mathbf{v}_1 + \mathbf{v}_2)$ also span V :

$$\begin{aligned} \mathbf{u} &= a_1\mathbf{v}_1 + a_2\mathbf{v}_2 \\ &= a_1\mathbf{v}_1 + a_2(\mathbf{v}_1 + \mathbf{v}_2) - a_2\mathbf{v}_1 \\ &= (a_1 - a_2)\mathbf{v}_1 + a_2(\mathbf{v}_1 + \mathbf{v}_2) \end{aligned}$$

Since k_1 and k_2 are arbitrary, this allows us to generate infinitely many sets of vectors that span any space from a single spanning set.

If scalars a_1 and a_2 decompose \mathbf{u} over \mathbf{v}_1 and \mathbf{v}_2 , then $(a_1 - a_2)$ and a_2 decompose \mathbf{u} over \mathbf{v}_1 and $(\mathbf{v}_1 + \mathbf{v}_2)$.

7.3 Review: Linear Independence

We said before that a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ are *linearly independent* if and only if

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_n\mathbf{v}_n = \mathbf{0}$$

implies that all coefficients a_1, a_2, \dots, a_n are zero. No linear combination of a set of linearly independent vectors can be the zero vector except for the trivial case where all the coefficients are zero. We often say that a set of vectors are linearly dependent if one of the vectors can be written as a linear combination of the others, i.e.

$$\mathbf{v}_i = a_1\mathbf{v}_1 + \dots + a_{i-1}\mathbf{v}_{i-1} + a_{i+1}\mathbf{v}_{i+1} + \dots + a_n\mathbf{v}_n$$

Moving the vector \mathbf{v}_i to the right hand side

$$\mathbf{0} = a_1\mathbf{v}_1 + \dots + a_{i-1}\mathbf{v}_{i-1} - \mathbf{v}_i + a_{i+1}\mathbf{v}_{i+1} + \dots + a_n\mathbf{v}_n$$

we see 1.) a linear combination of the vectors sums to the zero vector on the left, and 2.) at least one of the coefficients (the -1 in front of \mathbf{v}_i) is nonzero. This is consistent with the above definition of linear independence. We said these vectors were not linearly independent, so it is possible for a linear combination to sum to zero using at least one nonzero coefficient.

7.4 Basis

The concepts of span and linear independence are a powerful combination. Any linearly independent set of vectors that span a space V are called a *basis* for V . Any vector in a space can be decomposed over a set of vectors that span the space. **However, every vector in a space has a unique decomposition over an associated basis.** Said another way, for every vector in a space, there are only one set of coefficients a_1, a_2, \dots, a_n that decompose it over a basis $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$.

We can prove that a decomposition over a basis is unique by contradiction. Suppose there were two sets of coefficients – a_1, a_2, \dots, a_n and b_1, b_2, \dots, b_n – that decomposed a vector \mathbf{u} over a basis $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$. Then

$$\mathbf{u} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_n\mathbf{v}_n = b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \dots + b_n\mathbf{v}_n$$

We can move all the right hand side over to the left and group terms to give

$$(a_1 - b_1)\mathbf{v}_1 + (a_2 - b_2)\mathbf{v}_2 + \dots + (a_n - b_n)\mathbf{v}_n = \mathbf{0}$$

Remember that $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is a basis, so the vectors must be linearly independent. By the definition of linear independence, the only way the above equation can be true is if all the coefficients are zero. This implies that $a_1 = b_1$, $a_2 = b_2$, and so on. Clearly this is a violation of our original statement that a_1, a_2, \dots, a_n and b_1, b_2, \dots, b_n were different. Therefore, there can only be one way to decompose any vector onto a basis.

7.4.1 Testing if vectors form a basis

Every basis for a vector space has the same number of vectors. This number is called the *dimension* of the vector space. For standard vectors spaces like \mathbb{R}^n , the dimension is n . – the dimension of \mathbb{R}^2 is 2, and the dimension of \mathbb{R}^3 is 3.

Most people think of dimension as the number of elements in a vector. While the true definition of dimension is the number of vectors in the basis, counting elements in a vector works for spaces like \mathbb{R}^n . To see why, remember that the Cartesian unit vectors $\hat{\mathbf{e}}_i$ form a basis for \mathbb{R}^n , but we need n of these vectors, one per element.

Any set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is a basis for a space V if and only if:

1. The number of vectors (n) matches the dimension of V .
2. The vectors span V .
3. The vectors are linearly independent.

Proving any two of the above statements automatically implies the third is true.

We get to choose which two of the above three statements to prove when verifying that $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is a basis. The first statement is usually trivial – does the number of vectors match the dimension? – so we almost always choose to prove the first statement. Proving that vectors are linearly independent is always easier than proving the vectors span the space. If we collect the vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ into a matrix, the rank of this matrix should be n if the vectors are linearly independent.

7.4.2 Decomposing onto a basis

How do we decompose a vector onto a basis? Remember that decomposing \mathbf{u} over $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is equivalent to finding a set of coefficients a_1, a_2, \dots, a_n such that

$$a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \dots + a_n \mathbf{v}_n = \mathbf{u}$$

Let's collect the vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ into a matrix \mathbf{V} , where each vector \mathbf{v}_i is the i th column in \mathbf{V} . Then

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \mathbf{V}\mathbf{a} = \mathbf{u}$$

We see that finding the coefficients a_1, a_2, \dots, a_n that decompose a vector \mathbf{u} onto a basis $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is equivalent to solving the linear system $\mathbf{V}\mathbf{a} = \mathbf{u}$.

By formulating vector decomposition as a linear system, we can easily see why the decomposition of a vector over a basis is unique. In \mathbb{R}^n , the basis contains n vectors, each with n elements. So, the matrix \mathbf{V} is a square, $n \times n$ matrix. Since the vectors in the basis (and therefore the columns in \mathbf{V}) are linearly independent, the matrix \mathbf{V} has full rank. Thus, the solution to $\mathbf{V}\mathbf{a} = \mathbf{u}$ must be unique, implying that the decomposition of every vector onto a basis is unique.

Since \mathbf{V} is square and full rank, its inverse (\mathbf{V}^{-1}) must exist. The system must have a unique solution $\mathbf{a} = \mathbf{V}^{-1}\mathbf{u}$.

7.5 Orthogonal and Orthonormal Vectors

A set of vectors is an *orthogonal set* if every vector in the set is orthogonal to every other vector in the set. If every vector in an orthogonal set has been normalized, we say the vectors form an *orthonormal set*. Orthogonal and orthonormal sets are ideal candidates for basis vectors. Since there is no “overlap” among the vectors, we can easily decompose other vectors onto orthogonal basis vectors.

Imagine you have an orthogonal set of vectors you want to use as a basis. We'll assume you have the correct number of vectors (equal to the dimension of your space) for this to be possible. Based on the above requirements for basis vectors, we need only to show that these vectors are linearly independent. If so, they are a basis. For a set of n orthogonal vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$, linear independence requires that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{0}$$

if and only if all the coefficients a_1, a_2, \dots, a_n are equal to zero. Let's take the dot product of both sides of the above equation with the vector \mathbf{v}_1

$$(a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{0}) \cdot \mathbf{v}_1 = \mathbf{0} \cdot \mathbf{v}_1$$

On the right hand side, we know that $\mathbf{0} \cdot \mathbf{v}_1 = 0$ for any vector \mathbf{v}_1 . We also distribute the dot product on the left hand side to give

$$a_1\mathbf{v}_1 \cdot \mathbf{v}_1 + a_2\mathbf{v}_2 \cdot \mathbf{v}_1 + \cdots + a_n\mathbf{v}_n \cdot \mathbf{v}_1 = 0$$

Since all the vectors are orthogonal, $\mathbf{v}_i \cdot \mathbf{v}_1$ is zero except when $i = 1$. Canceling out all the dot products equal to zero shows that

$$a_1 \mathbf{v}_1 \cdot \mathbf{v}_1 = a_1 \|\mathbf{v}_1\|^2 = 0$$

We know that $\|\mathbf{v}_1\|^2 \neq 0$, so the only way $a_1 \mathbf{v}_1 \cdot \mathbf{v}_1$ can equal zero is if a_1 is zero. If we repeat this entire process by taking the dot product with \mathbf{v}_2 instead of \mathbf{v}_1 , we will find that $a_2 = 0$. This continues with $\mathbf{v}_3, \dots, \mathbf{v}_n$ until we can say that $a_1 = a_2 = \dots = a_n = 0$. Therefore, if the vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ are an orthogonal (or orthonormal) set, they are linearly independent.

7.5.1 Decomposing onto orthonormal vectors

We saw previously that finding the coefficients to decompose a vector onto a basis requires solving a system of linear equations. For high-dimensional spaces, solving such a system can be computationally expensive. Fortunately, decomposing a vector onto an orthonormal basis is far more efficient.

Theorem. *The decomposition of a vector \mathbf{u} onto an orthonormal basis $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots, \hat{\mathbf{v}}_n$ given by*

$$\mathbf{u} = a_1 \hat{\mathbf{v}}_1 + a_2 \hat{\mathbf{v}}_2 + \dots + a_n \hat{\mathbf{v}}_n$$

has coefficients

$$a_1 = \mathbf{u} \cdot \hat{\mathbf{v}}_1$$

$$a_2 = \mathbf{u} \cdot \hat{\mathbf{v}}_2$$

$$\vdots$$

$$a_n = \mathbf{u} \cdot \hat{\mathbf{v}}_n$$

We use the symbol $\hat{\mathbf{v}}_i$ for vectors in an orthonormal set to remind us that each vector has been normalized.

Proof. We use a similar strategy as when we proved the linear independence of orthogonal sets. Let's start with the formula for vector decomposition

$$\mathbf{u} = a_1 \hat{\mathbf{v}}_1 + a_2 \hat{\mathbf{v}}_2 + \dots + a_n \hat{\mathbf{v}}_n$$

Taking the dot product of both sides with the vector $\hat{\mathbf{v}}_1$ yields (after distributing the right hand side)

$$\mathbf{u} \cdot \hat{\mathbf{v}}_1 = a_1 \hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1 + a_2 \hat{\mathbf{v}}_2 \cdot \hat{\mathbf{v}}_1 + \dots + a_n \hat{\mathbf{v}}_n \cdot \hat{\mathbf{v}}_1$$

Because all the vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots, \hat{\mathbf{v}}_n$ are orthogonal, the only nonzero term on the right hand side is $a_1 \hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1$, so

$$\mathbf{u} \cdot \hat{\mathbf{v}}_1 = a_1 \hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1$$

By definition of the dot product, $\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{v}}_1 = \|\hat{\mathbf{v}}_1\|^2$. Since $\hat{\mathbf{v}}_1$ is a unit vector, $\|\hat{\mathbf{v}}_1\|^2 = 1$. Thus $a_1 = \mathbf{u} \cdot \hat{\mathbf{v}}_1$. By repeating the same procedure with $\hat{\mathbf{v}}_2$, we find that $a_2 = \mathbf{u} \cdot \hat{\mathbf{v}}_2$, and so on. \square

Decomposing vectors over an orthonormal basis is efficient, requiring only a series of dot products to compute the coefficients. For example, we can decompose the vector $\mathbf{u} = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix}$ over the orthonormal

$$\text{basis } \left\{ \hat{\mathbf{v}}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \hat{\mathbf{v}}_2 = \begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix}, \hat{\mathbf{v}}_3 = \begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} \right\}.$$

$$a_1 = \mathbf{u} \cdot \hat{\mathbf{v}}_1 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 7 + 0 + 0 = 7$$

$$a_2 = \mathbf{u} \cdot \hat{\mathbf{v}}_2 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix} = 0 - 3 + 8 = 5$$

$$a_3 = \mathbf{u} \cdot \hat{\mathbf{v}}_3 = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} = 0 - 4 - 6 = -10$$

The decomposition of \mathbf{u} is

$$\mathbf{u} = 7 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 5 \begin{pmatrix} 0 \\ 3/5 \\ 4/5 \end{pmatrix} - 10 \begin{pmatrix} 0 \\ 4/5 \\ -3/5 \end{pmatrix} = \begin{pmatrix} 7 + 0 + 0 \\ 0 + 3 - 8 \\ 0 + 4 + 6 \end{pmatrix} = \begin{pmatrix} 7 \\ -5 \\ 10 \end{pmatrix}$$

7.5.2 Checking an orthonormal set

Given a set of vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots, \hat{\mathbf{v}}_n$, how can we verify that they are orthonormal? We need to test two things.

1. All vectors are normalized ($\|\hat{\mathbf{v}}_i\| = 1$ for all $\hat{\mathbf{v}}_i$).
2. All vectors are mutually orthogonal ($\hat{\mathbf{v}}_i \cdot \hat{\mathbf{v}}_j = 0$ for all $i \neq j$).

The first test is straightforward. The second can be a little cumbersome, as we need to test all $n^2 - n/2$ pairs of vectors for orthogonality. A simpler, albeit more sometimes more computationally intensive approach, is to collect the vectors into a matrix $\mathbf{V} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix}$. Then the set of vectors is orthonormal if and only if $\mathbf{V}^{-1} = \mathbf{V}^T$. While inverting a matrix is “expensive”, for small to medium size vector sets this method avoids the need to iterate over all pairs of vectors to test for orthonormality.

Interestingly, the proof of this method (not shown here) reveals that if the columns of \mathbf{V} are an orthonormal set, the rows of \mathbf{V} are also an orthonormal set!

7.5.3 Projections

Our next goal will be to create orthonormal sets of vectors from sets that are not orthogonal. Before introducing such an algorithm, we

need to develop a geometric tool – the vector *projection*. The projection of vector \mathbf{v} onto vector \mathbf{u} is a vector that points along \mathbf{u} with length equal to the “shadow” of \mathbf{v} onto \mathbf{u} . Previously we used the dot product to calculate the magnitude of the projection of \mathbf{v} onto \mathbf{u} , which was a scalar equal to $\|\mathbf{v}\| \cos \theta$, where θ is the angle between \mathbf{v} and \mathbf{u} . To calculate the actual projection, we multiply the magnitude of the projection ($\|\mathbf{v}\| \cos \theta$) by a unit vector that points along \mathbf{u} . Thus the projection of \mathbf{v} onto \mathbf{u} is defined as

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = (\|\mathbf{v}\| \cos \theta) \hat{\mathbf{u}}$$

By definition, $\hat{\mathbf{u}} = \mathbf{u} / \|\mathbf{u}\|$. Also, we note that $\mathbf{v} \cdot \mathbf{u} = \|\mathbf{v}\| \|\mathbf{u}\| \cos \theta$, so the expression $\|\mathbf{u}\| \cos \theta$ can be written in terms of the dot product $(\mathbf{v} \cdot \mathbf{u}) / \|\mathbf{u}\|$. We can rewrite our formula for the projection using only dot products:

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = (\|\mathbf{v}\| \cos \theta) \hat{\mathbf{u}} = \frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|} \frac{\mathbf{u}}{\|\mathbf{u}\|} = \frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|^2} \mathbf{u} = \frac{\mathbf{v} \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}} \mathbf{u}$$

We can use the projection to make any two vectors orthogonal, as demonstrated by the following theorem.

Theorem. *Given any vectors \mathbf{v} and \mathbf{u} , the vector*

$$\mathbf{v} - \text{proj}_{\mathbf{u}}(\mathbf{v})$$

is orthogonal to \mathbf{u} .

Proof. If the vector $\mathbf{v} - \text{proj}_{\mathbf{u}}(\mathbf{v})$ is orthogonal to \mathbf{u} , the dot product between these vectors must be zero.

$$\begin{aligned} (\mathbf{v} - \text{proj}_{\mathbf{u}}(\mathbf{v})) \cdot \mathbf{u} &= \left(\mathbf{v} - \frac{\mathbf{v} \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}} \mathbf{u} \right) \cdot \mathbf{u} \\ &= \mathbf{v} \cdot \mathbf{u} - \frac{\mathbf{v} \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}} \mathbf{u} \cdot \mathbf{u} \\ &= \mathbf{v} \cdot \mathbf{u} - \mathbf{v} \cdot \mathbf{u} \\ &= 0 \end{aligned}$$

□

Subtracting the projection of \mathbf{v} onto \mathbf{u} from the vector \mathbf{v} “corrects” \mathbf{v} by removing its overlap with \mathbf{u} . The resulting vector is a vector closest to \mathbf{v} that is still orthogonal to \mathbf{u} .

7.5.4 Creating orthonormal basis vectors

We can make any two vectors orthogonal by adjusting one based on its projection onto the other. We can apply these corrections iteratively to make any set of linearly independent vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ into an orthonormal basis set $\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \dots, \hat{\mathbf{u}}_n$. First, we set

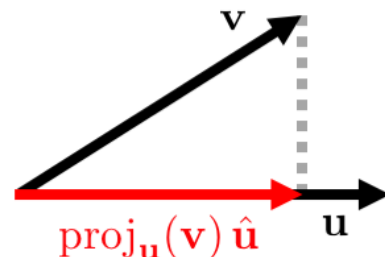


Figure 7.1: The projection is a vector “shadow” of one vector onto another.

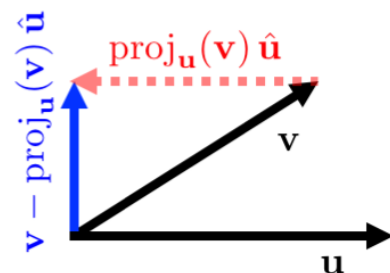


Figure 7.2: Subtracting the projection of \mathbf{v} onto \mathbf{u} from \mathbf{v} makes the vectors orthogonal.

We must begin with linearly independent vectors. Otherwise, orthogonalization will turn one of the vectors into the zero vector, which is not allowed in a basis.

$$\mathbf{u}_1 = \mathbf{v}_1$$

We leave this first vector unchanged. All other vectors will be made orthogonal to it (and each other). Next, we create \mathbf{u}_2 by making \mathbf{v}_2 orthogonal to \mathbf{u}_1 :

$$\mathbf{u}_2 = \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2)$$

Now we have two orthogonal vectors, \mathbf{u}_1 and \mathbf{u}_2 . We continue by creating \mathbf{u}_3 from \mathbf{v}_3 , but this time we must make \mathbf{v}_3 orthogonal to both \mathbf{u}_1 and \mathbf{u}_2 :

$$\mathbf{u}_3 = \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_3)$$

We continue this process for all n vectors, making each vector \mathbf{v}_i orthogonal to all the newly created orthogonal vectors $\mathbf{u}_1, \dots, \mathbf{u}_{i-1}$.

This approach is called the Gram-Schmidt algorithm. Given a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$, we create a set of orthogonal vectors

Said more succinctly,

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{v}_1 \\ \mathbf{u}_2 &= \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2) \\ \mathbf{u}_3 &= \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \text{proj}_{\mathbf{u}_2}(\mathbf{v}_3) \\ &\vdots \\ \mathbf{u}_i &= \mathbf{v}_i - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_i) - \dots - \text{proj}_{\mathbf{u}_{i-1}}(\mathbf{v}_i) \\ &\vdots \\ \mathbf{u}_n &= \mathbf{v}_n - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_n) - \dots - \text{proj}_{\mathbf{u}_{n-1}}(\mathbf{v}_n) \end{aligned}$$

$$\mathbf{u}_i = \mathbf{v}_i - \sum_{k=1}^{i-1} \text{proj}_{\mathbf{u}_k}(\mathbf{v}_i)$$

The Gram-Schmidt algorithm produces an orthogonal set of vectors. To make the set orthonormal, we must subsequently normalize each vector.

8

Eigenvalues and Eigenvectors

Consider the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}$$

Multiplying \mathbf{A} by the vector $\mathbf{x}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ gives an interesting result.

$$\mathbf{A}\mathbf{x}_1 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ -5 \end{pmatrix} = -5 \begin{pmatrix} -1 \\ 1 \end{pmatrix} = -5\mathbf{x}_1$$

Similarly, with $\mathbf{x}_2 = \begin{pmatrix} -7 \\ 1 \end{pmatrix}$:

$$\mathbf{A}\mathbf{x}_2 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix} \begin{pmatrix} -7 \\ 1 \end{pmatrix} = \begin{pmatrix} -7 \\ 1 \end{pmatrix} = \mathbf{x}_2$$

In both cases, multiplication by \mathbf{A} returned a scalar multiple of the vector (-5 for \mathbf{x}_1 and 1 for \mathbf{x}_2). This is not a property of solely the matrix \mathbf{A} , since the vector $\mathbf{x}_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is not transformed by a single scalar.

$$\mathbf{A}\mathbf{x}_3 = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ -5 \end{pmatrix} \neq \lambda\mathbf{x}_3$$

Similarly, the results we are seeing are not properties of the vectors \mathbf{x}_1 and \mathbf{x}_2 , since they do not become scalar multiples of themselves when multiplied by other matrices.

$$\mathbf{B} = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix}$$

$$\mathbf{B}\mathbf{x}_1 = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} \neq \lambda\mathbf{x}_1$$

$$\mathbf{B}\mathbf{x}_2 = \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} -7 \\ 1 \end{pmatrix} = \begin{pmatrix} -13 \\ 21 \end{pmatrix} \neq \lambda\mathbf{x}_2$$

The phenomena we're observing is a result of the paring between the matrix \mathbf{A} and the vectors \mathbf{x}_1 and \mathbf{x}_2 . In general, we see that multiplying a vector by a matrix returns a scalar multiple of the vector, or

$$\mathbf{Ax} = \lambda\mathbf{x}$$

Any vector \mathbf{x} that obeys the above relationship is called an *eigenvector* of the matrix \mathbf{A} . The scalar λ is called the *eigenvalue* associated with the eigenvector \mathbf{x} . The vector \mathbf{x} is an eigenvector of the matrix \mathbf{A} ; it is not generally an eigenvector of other matrices.

In the example above, the matrix $\mathbf{A} = \begin{pmatrix} 2 & 7 \\ -1 & -6 \end{pmatrix}$ has two eigenvectors, $\mathbf{v}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ with eigenvalue $\lambda_1 = -5$, and $\mathbf{v}_2 = \begin{pmatrix} -7 \\ 1 \end{pmatrix}$ with eigenvalue $\lambda_2 = 1$.

Eigenvectors were originally called *characteristic* vectors, as they describe the character of the matrix. German mathematicians dropped this nomenclature in favor of the German prefix "eigen-", which mean "own". An eigenvector can be viewed as one of a matrix's "own" vectors since it is not rotated when transformed by multiplication.

8.1 Properties of Eigenvectors and Eigenvalues

Only square matrices have eigenvectors and eigenvalues. An n by n matrix of real numbers can have up to n distinct eigenvectors. Each eigenvector is associated with an eigenvalue, although the eigenvalues can be duplicated. Said another way, two eigenvectors \mathbf{v}_1 and \mathbf{v}_2 of a matrix will never be the same, but the corresponding eigenvalues λ_1 and λ_2 can be identical.

To understand why the matrix must be square, remember that a non-square matrix with m rows and n columns transforms an n -dimensional vectors into an m -dimensional vector. Clearly, the m -dimensional output cannot be the n -dimensional input multiplied by a scalar!

Although the number of eigenvectors may vary, all eigenvectors for a matrix are linearly independent. Thus, if an n by n matrix has n eigenvectors, these vectors can be used as a basis (called an *eigenbasis*). If an eigenbasis exists for a matrix, decomposing vectors over this basis simplifies the process of matrix multiplication. To illustrate, imagine we decompose the vector \mathbf{x} over a set of eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$. Decomposing \mathbf{x} means we can find coefficients a_1, \dots, a_n such that

$$\mathbf{x} = a_1\mathbf{v}_1 + \dots + a_n\mathbf{v}_n$$

Now let's compute the product \mathbf{Ax} . We multiply both sides of the decomposition by \mathbf{A} .

$$\mathbf{Ax} = \mathbf{A}(a_1\mathbf{v}_1 + \dots + a_n\mathbf{v}_n)$$

We distribute the matrix \mathbf{A} into the sum on the right hand side and note that the constants a_i can be moved in front of the matrix multiplication.

$$\mathbf{Ax} = a_1\mathbf{Av}_1 + \dots + a_n\mathbf{Av}_n$$

Remember that $\mathbf{v}_1, \dots, \mathbf{v}_n$ are eigenvectors of \mathbf{A} , so $\mathbf{Av}_i = \lambda_i\mathbf{v}_i$. We can simplify the previous expression to

$$\mathbf{Ax} = a_1\lambda_1\mathbf{v}_1 + \dots + a_n\lambda_n\mathbf{v}_n$$

An n by n matrix with n eigenvectors and n distinct eigenvalues is called a *perfect matrix*. As the name implies, perfect matrices are great to find, but somewhat uncommon.

We don't need to perform the multiplication at all! Instead, we can scale each eigenvector by the eigenvalue. Multiplying again by the matrix \mathbf{A} multiplies each eigenvector by its eigenvalue.

$$\mathbf{A}^2 \mathbf{x} = a_1 \lambda_1^2 \mathbf{v}_1 + \cdots + a_n \lambda_n^2 \mathbf{v}_n$$

$$\mathbf{A}^k \mathbf{x} = a_1 \lambda_1^k \mathbf{v}_1 + \cdots + a_n \lambda_n^k \mathbf{v}_n$$

We use the notation \mathbf{A}^2 to denote $\mathbf{A}\mathbf{A}$, \mathbf{A}^3 for $\mathbf{A}\mathbf{A}\mathbf{A}$, and \mathbf{A}^k for the product of k matrices \mathbf{A} .

8.2 Computing Eigenvectors and Eigenvalues

We can use the relationship between matrix multiplication and eigenvalues to find eigenvectors for any matrix. Our computational approach is based on the following theorem.

Theorem. *Given any (random) vector \mathbf{b} , repeated multiplication by the matrix \mathbf{A} will converge to the eigenvector of \mathbf{A} with the largest magnitude eigenvalue – provided the largest eigenvalue is unique. Said another way,*

$$\lim_{k \rightarrow \infty} \mathbf{A}^k \mathbf{b} = \mathbf{v}_{\max}$$

Proof. We know that the product $\mathbf{A}\mathbf{x}$ can be expressed as a linear combination of the eigenvectors and eigenvalues of \mathbf{A} , i.e. $\mathbf{A}\mathbf{x} = a_1 \lambda_1 \mathbf{v}_1 + \cdots + a_n \lambda_n \mathbf{v}_n$. Thus

$$\lim_{k \rightarrow \infty} \mathbf{A}^k \mathbf{b} = \lim_{k \rightarrow \infty} (a_1 \lambda_1^k \mathbf{v}_1 + \cdots + a_n \lambda_n^k \mathbf{v}_n)$$

As k increases, the values λ_i^k grow very large. However, the λ_i do not grow at the same rate. The largest eigenvalue will grow the fastest. At very large values of k , the term associated with the largest eigenvalue will dominate the entire sum, so the result will point in only the direction of the associated eigenvector. Note that convergence to a single eigenvector requires that the largest eigenvalue be distinct. If two eigenvectors have the same (largest) eigenvalue, both terms in the above sum would “blow up” at the same rate. Repeated multiplications by \mathbf{A} would then converge to the sum of the two associated eigenvectors. \square

The above theorem allows us to find the eigenvector paired with the largest eigenvalue. While the direction of the eigenvector doesn't change, its magnitude grows as the number of multiplication of \mathbf{A} increases. If convergence is slow, we might need to work with numbers before finding the eigenvector. To avoid numerical difficulties, we renormalize the vector after every multiplication by \mathbf{A} . This algorithm is called the Power Iteration method, which proceeds as follows:

1. Choose a random vector \mathbf{b}_0 . For fastest convergence, it helps to choose a vector close to \mathbf{v}_{\max} if possible. Normalize this vector to product $\hat{\mathbf{b}}_0 = \mathbf{b}_0 / \|\mathbf{b}_0\|$.

2. Compute vector $\mathbf{b}_1 = \mathbf{A}\hat{\mathbf{b}}_0$. Normalize this vector to give $\hat{\mathbf{b}}_1 = \mathbf{b}_1 / \|\mathbf{b}_1\|$.
3. Repeat step 2 to product $\hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3, \dots, \hat{\mathbf{b}}_k$. Stop when all entries of $\hat{\mathbf{b}}_k$ do not change from the entries in $\hat{\mathbf{b}}_{k-1}$. The vector $\hat{\mathbf{b}}_k$ is an eigenvector of \mathbf{A} .

Now that we have the eigenvector \mathbf{v}_{\max} , how do we find the associated eigenvalue λ_{\max} ? We know that \mathbf{v}_{\max} is an eigenvector of \mathbf{A} , to $\mathbf{A}\mathbf{v}_{\max} = \lambda_{\max}\mathbf{v}_{\max}$. The i th element in $\mathbf{A}\mathbf{v}_{\max}$ should be equal to λ_{\max} times the i th element in \mathbf{v}_{\max} . However, since we only found a numerical approximation to the \mathbf{v}_{\max} , the estimate for λ_{\max} from each element in \mathbf{v}_{\max} might differ slightly. To “smooth out” these variations, compute the eigenvalue using the Rayleigh quotient:

$$\lambda_{\max} = \frac{\mathbf{v}_{\max} \cdot \mathbf{A}\mathbf{v}_{\max}}{\mathbf{v}_{\max} \cdot \mathbf{v}_{\max}}$$

The dot product in the Rayleigh quotient averages out all of the small discrepancies between our estimate \mathbf{v}_{\max} and the true largest eigenvector. The Rayleigh quotient provides a numerically stable estimate of the largest eigenvalue.

Now that we’ve found the first eigenvector, how do we find the others? If we start the Power Iteration method over again using the matrix $(\mathbf{A} - \lambda_{\max}\mathbf{I})$ instead of \mathbf{A} , the algorithm will converge to the eigenvector associated with the second largest eigenvalue. We can subtract this eigenvalue from \mathbf{A} and repeat to find the third eigenvector, and so on. Proving Power Iteration is able to find subsequent eigenvectors is beyond the scope of this course. However, as we’ll see later, finding only the first eigenvector is sufficient for addressing a number of interesting problems.

8.2.1 Eigenvalues and Eigenvectors in MATLAB

The MATLAB function `eig` computes eigenvalues and eigenvectors. The statement `[V,L] = eig(A)` involving an n by n matrix \mathbf{A} returns two n by n matrices:

- Each column of the matrix \mathbf{V} is an eigenvector \mathbf{A} .
- The matrix \mathbf{L} is a diagonal matrix. The i th entry on the diagonal is the eigenvalue associated with the i th column in \mathbf{V} .

Remember that any vector that points in the same direction as an eigenvector of a matrix is also an eigenvector of that matrix. If the eigenvectors returned by computational systems like MATLAB are not what you expect, remember that they may be normalized or scaled – but still point along the same direction.

The eigenvector associated with the largest magnitude eigenvalue is called the *leading eigenvector*.

To see why the Raleigh quotient works, consider an eigenvector \mathbf{v} for matrix \mathbf{A} with associated eigenvalue λ . Then

$$\frac{\mathbf{v} \cdot \mathbf{A}\mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} = \frac{\mathbf{v} \cdot (\lambda\mathbf{v})}{\mathbf{v} \cdot \mathbf{v}} = \lambda \frac{\mathbf{v} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} = \lambda$$

8.3 Applications

Eigenvalue and eigenvectors can be used to solve a number of interesting engineering and data science problems.

8.3.1 Solving Systems of ODEs

Consider the linear system of ODEs

$$\begin{aligned}\frac{dx_1}{dt} &= x_1 + 2x_2 \\ \frac{dx_2}{dt} &= 3x_1 + 2x_2\end{aligned}$$

with initial conditions $x_1(0) = 0$ and $x_2(0) = -4$. We can write this system using vectors and matrices as

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}, \quad \mathbf{x}(0) = \mathbf{x}_0$$

where for the example above

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}, \quad \mathbf{x}_0 = \begin{pmatrix} 0 \\ -4 \end{pmatrix}$$

If we know the eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ and eigenvalues $\lambda_1, \dots, \lambda_n$ for the matrix \mathbf{A} , we can compute the solution as

$$\mathbf{x}(t) = c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} + \dots + c_n \mathbf{v}_n e^{\lambda_n t}$$

The scalars c_1, \dots, c_n are the constants of integration. To find these values, notice what happens to our solution at time $t = 0$:

$$\mathbf{x}(0) = \mathbf{x}_0 = c_1 \mathbf{v}_1 + c_2 \mathbf{v}_2 + \dots + c_n \mathbf{v}_n$$

At $t = 0$, the right hand side is a decomposition of the initial conditions \mathbf{x}_0 . If we collect the eigenvectors as columns of a matrix $\mathbf{V} = (\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_n)$, we can find the constants c_1, \dots, c_n by solving the linear system

$$\mathbf{V} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \mathbf{x}_0$$

Returning to our original example, the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix}$$

has eigenvalue/eigenvector pairs

$$\lambda_1 = -1, \quad \mathbf{v}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \text{and} \quad \lambda_2 = 4, \quad \mathbf{v}_2 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

This solution requires the matrix \mathbf{A} be perfect and therefore have a complete set of eigenvectors.

The function $f(t) = e^{\lambda t}$ is an *eigenfunction* of the derivative operator, i.e.

$$\frac{d}{dt} f(t) = \lambda e^{\lambda t} = \lambda f(t)$$

. The solution of a system of linear ODEs is the product of the eigenvectors of \mathbf{A} and the eigenfunctions of $\frac{d\mathbf{x}}{dt}$.

The integration constants c_1 and c_2 are defined by the system $\mathbf{V}\mathbf{c} = \mathbf{x}_0$, which for this example is

$$\begin{pmatrix} -1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ -4 \end{pmatrix}$$

Solving the above equations reveals $c_1 = -4/5$ and $c_2 = -8/5$. The final solution to this systems of ODEs is

$$\mathbf{x}(t) = -\frac{8}{5} \begin{pmatrix} -1 \\ 1 \end{pmatrix} e^{-t} - \frac{4}{5} \begin{pmatrix} 2 \\ 3 \end{pmatrix} e^{4t}$$

8.3.2 Stability of Linear ODEs

The eigenvalues of \mathbf{A} are sufficient to tell if the system $\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$ is stable. For a system of linear ODEs to be stable, all eigenvalues of \mathbf{A} must be nonpositive. If the eigenvalues are all negative, each term $e^{\lambda_i t}$ goes to zero at long times, so all variables in the system to go zero. If any of the eigenvalues are zero, the system is still stable (provided all other eigenvalues are negative), but the system will go to a constant value $c_i \mathbf{v}_i$, where \mathbf{v}_i is the eigenvector associated with the zero eigenvalue.

8.3.3 Positive Definite Matrices

A symmetric matrix \mathbf{A} is *positive definite* (p.d.) if $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all nonzero vectors \mathbf{x} . If a matrix \mathbf{A} satisfies the slightly relaxed requirement that $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ for all nonzero \mathbf{x} , we say that \mathbf{A} is *positive semi-definite* (p.s.d.).

Knowing that a matrix is positive (semi-)definite is useful for quadratic programming problems like the Support Vector Machine classifier. The quadratic function $f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x}$ if and only if the matrix \mathbf{Q} is positive semi-definite. For optimization problems like quadratic programs, the convexity of the objective function has enormous implications. Convex quadratic programs must only have global optima, making them easy to solve using numerical algorithms.

Determining if a matrix is positive (semi-)definite can be difficult unless we use eigenvalues. Any matrix with only positive eigenvalues is positive definite, and any matrix with only nonnegative eigenvalues is positive semi-definite. For example, consider the 2×2 identity matrix

$$\mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The product $\mathbf{x}^T \mathbf{I} \mathbf{x}$ is

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1^2 + x_2^2$$

Remember that a matrix \mathbf{A} is symmetric if $\mathbf{A} = \mathbf{A}^T$.

If \mathbf{Q} is positive definite (rather than just positive semi-definite) then $\mathbf{x}^T \mathbf{Q} \mathbf{x}$ is strictly convex.

Since $x_1^2 + x_2^2$ is greater than zero for all nonzero inputs x_1 and x_2 , the matrix \mathbf{I} is positive definite and all its eigenvalues should be positive. Indeed, the eigenvalues for the identity matrix are $\lambda_1 = \lambda_2 = 1$.

As another example, consider the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$$

The product $\mathbf{x}^T \mathbf{A} \mathbf{x} = x_1^2 - 4x_1x_2 + x_2^2$, which is not always positive. When $x_1 = x_2 = 1$, we see that $x_1^2 - 4x_1x_2 + x_2^2 = -2$. We know that \mathbf{A} is not positive definite (or even positive semi-definite), so \mathbf{A} should have at least one negative eigenvalue. As expected, the eigenvalues for \mathbf{A} are $\lambda_1 = 3$ and $\lambda_2 = -1$.