2

Matrices

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. ($\mathbb{R}^n \rightarrow \mathbb{R}$)

- The **scalar product** maps a scalar and a vector to a new vector ($\mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$), but can only scale the magnitude of the vector (or flip it if the scalar is negative).

- The **dot product** maps vectors to a scalar ($\mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$) 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 $y$ from vector $x$. To be as general as possible, we should let each element in $y$ be an arbitrary linear combination of the elements in $x$. This implies that

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

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 $x$ to $y$. For convenience, we collect the set of weights into an $n$ by $n$ numeric grid called a **matrix**.

If $A$ is a real-valued matrix with dimensions $m \times n$, we say $A \in \mathbb{R}^{m \times n}$ and $\dim(A) = m \times n$. 
\[
A = \begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\]

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 \(AB\) is a matrix \(C\) where each element \(c_{ij}\) in \(C\) is the dot product between the \(i\)th row in \(A\) and the \(j\)th column in \(B\):

\[
c_{ij} = A(i,:) \cdot B(:,j)
\]

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

\[
y = Ax
\]

### 2.1.1 Generalized Multiplication

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

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

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

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

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

For the system \(y = Ax\), if \(\dim(A) = m \times n\) and \(\dim(x) = n \times 1\), \(\dim(y) = m \times 1\), i.e. \(y\) is a column vector in \(\mathbb{R}^m\).
\[
\begin{align*}
\mathbf{AB} &= \begin{pmatrix} 1 \times 0 + 2 \times (-1) & 1 \times 1 + 2 \times 2 \\ 3 \times 0 + 4 \times (-1) & 3 \times 1 + 4 \times 2 \end{pmatrix} = \begin{pmatrix} -2 & 5 \\ -4 & 11 \end{pmatrix} \\
\mathbf{BA} &= \begin{pmatrix} 0 \times 1 + 1 \times 3 & 0 \times 2 + 1 \times 4 \\ -1 \times 1 + 2 \times 3 & -1 \times 2 + 2 \times 4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}
\end{align*}
\]

2.2 Identity Matrix

We need to find an element that serves as 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 1 will be a matrix, which we will call the identity matrix \( \mathbf{I} \). We require that

\[ \mathbf{I} \mathbf{x} = \mathbf{x} \mathbf{I} = \mathbf{x} \]

for all \( \mathbf{x} \). Assuming that \( \mathbf{x} \) is \( n \)-dimensional, \( \mathbf{I} \) must have \( n \) columns to be conformable. Also, the output of \( \mathbf{I} \mathbf{x} \) 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 \( \mathbf{I}(1,:) = (1 \, 0 \, \cdots \, 0) \). For the second row, \( \mathbf{I}(2,:) \cdot \mathbf{x} = x_2 \), so \( \mathbf{I}(2,:) = (0 \, 1 \, \cdots \, 0) \). In general, the \( i \)th row of \( \mathbf{I} \) has a 1 at position \( i \) and zeros everywhere else

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

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 \( \mathbf{A}^T \).

Other notations for the matrix transpose include \( \mathbf{A}^\intercal \) 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.

\[
\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}^T = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}^T = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}
\]

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 \( A \) into the identity matrix

\[
\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1' \\ y_2' \end{pmatrix}
\]

This leads us to our first strategy for solving linear systems of the form \( Ax = y \). We manipulate both sides of the equation (\( A \) and \( y \)) until \( A \) becomes the identity matrix. The vector \( x \) then equals the transformed vector \( y' \). Because we will be applying the same transformations to both \( A \) and \( y \), it is convenient to collect them both into an augmented matrix \( \begin{pmatrix} A & y \end{pmatrix} \). For \( 2 \times 2 \) 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 \( A \) into the identity matrix? There are three operations, called the elementary row operations, or EROs.
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

\[
\begin{align*}
4x_1 + 8x_2 - 12x_3 &= 44 \\
3x_1 + 6x_2 - 8x_3 &= 32 \\
-2x_1 - x_2 &= -7
\end{align*}
\]

This is a linear system of the form \(\mathbf{A}\mathbf{x} = \mathbf{y}\) 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.

\[
\frac{1}{2}R_1 \rightarrow \begin{pmatrix} 1 & 2 & -3 & 11 \\ 3 & 6 & -8 & 32 \\ -2 & -1 & 0 & -7 \end{pmatrix}
\]

\[
R_2 - 3R_3 \rightarrow \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 0 & 1 & -1 \\ -2 & -1 & 0 & -7 \end{pmatrix}
\]

\[
R_3 + 2R_1 \rightarrow \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 0 & 1 & -1 \\ 0 & 3 & -6 & 15 \end{pmatrix}
\]

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”.

\[
\frac{R_2}{R_3} \rightarrow \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 3 & -6 & 15 \\ 0 & 0 & 1 & -1 \end{pmatrix}
\]

\[
\frac{1}{3}R_2 \rightarrow \begin{pmatrix} 1 & 2 & -3 & 11 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix}
\]

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:

\[
x_3 = -1
\]
\[
x_2 + 2(-1) = 5 \Rightarrow x_2 = 3
\]
\[
x_1 + 2(3) - 3(-1) = 11 \Rightarrow x_1 = 2
\]

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

\[
\frac{R_1 - 2R_2}{R_3} \rightarrow \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix}
\]

\[
\frac{R_1 - R_3}{R_3} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & -2 & 5 \\ 0 & 0 & 1 & -1 \end{pmatrix}
\]

\[
\frac{R_2 + 2R_3}{R_3} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{pmatrix}
\]
The left three columns are the identity matrix, so the resulting system of equations has been simplified to

\[ \begin{align*}
  x_1 &= 2 \\
  x_2 &= 3 \\
  x_3 &= -1
\end{align*} \]

### 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 \( Ax = y \) when \( A \) is an \( n \times n \) matrix. We begin with the augmented matrix

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

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

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

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

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

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

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

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

\[
\begin{pmatrix}
  1 & a_{12}' & \cdots & a_{1n}' & y_1' \\
  0 & a_{22}' & \cdots & a_{2n}' & y_2' \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & a_{n2}' & \cdots & a_{nn}' & y_n'
\end{pmatrix}
\]
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 \(a_{32}, \ldots, a_{n2} = 0\). 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:

```plaintext
function Gaussian Elimination
  for j = 1 to n do
    \(a^{-1}_{jj} R_j\) \(\triangleright\) For every column
    for i = j + 1 to n do
      \(R_i - a_{ij} R_j\) \(\triangleright\) For every row below the diagonal
        \(R_i - a_{ij} R_j\) \(\triangleright\) Zero the below-diagonal element
    end for
  end for
end function
```

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 \((a^{-1}_{11} R_1)\) 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, \(a^{-1}_{11} a_{11}\), 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

\[
n \cdot a^{-1}_{11} R_1 + 2(n - 1)n = 2n^2 - n
\]

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 \(1 \times 1\) “submatrix” that includes only \(a_{nn}\). We add up the number of

Remember that

\[
\sum_{k=1}^{n} 1 = n
\]

\[
\sum_{k=1}^{n} k = \frac{n(n+1)}{2}
\]

\[
\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}
\]
operations for each of these $n$ submatrices

$$= \sum_{k=1}^{n} (2k^2 - k)$$

$$= 2 \sum_{k=1}^{n} k^2 - \sum_{k=1}^{n} k$$

$$= \frac{n(n+1)(2n+1)}{3} - \frac{n(n+1)}{2}$$

$$= \mathcal{O}(n^3)$$

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.

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