Modeling the World with Arrays
• We have so far (mostly) looked at what we can do with single numbers (and functions that return single numbers).

• Things can get much more interesting once we allow not just one, but many numbers together.

• It is natural to view an array of numbers as one object with its own rules. The simplest such set of rules is that of a vector.

• A 2D array of numbers can also be looked at as a matrix.

• So it’s natural to use the tools of computational linear algebra.
• ‘Vector’ and ‘matrix’ are just representations that come to life in many (many!) applications. The purpose of this section is to explore some of those applications.
What’s a vector?

An array that defines *addition* and *scalar multiplication* with reasonable rules such as

\[
\begin{align*}
    u + (v + w) &= (u + v) + w \\
    v + w &= w + v \\
    \alpha(u + v) &= \alpha u + \alpha v
\end{align*}
\]

These axioms generally follow from properties of “+” and “·” operators.
What would the concept of a vector look like in a programming language (e.g. Java)?

In a sense, ‘vector’ is an abstract interface, like this:

(Along with guarantees that add and multiply interact appropriately.)
Vectors in the ‘Real World’

**Demo:** Images as Vectors (click to visit)

**Demo:** Sounds as Vectors (click to visit)

**Demo:** Shapes as Vectors (click to visit)
What does a matrix do?

It represents a linear function between two vector spaces \( f : U \rightarrow V \) in terms of bases \( \mathbf{u}_1, \ldots, \mathbf{u}_n \) of \( U \) and \( \mathbf{v}_1, \ldots, \mathbf{v}_m \) of \( V \). Let

\[
\mathbf{u} = \alpha_1 \mathbf{u}_1 + \cdots + \alpha_n \mathbf{u}_n
\]

and

\[
\mathbf{v} = \beta_1 \mathbf{v}_1 + \cdots + \beta_m \mathbf{v}_m.
\]
Then $f$ can *always* be represented as a matrix that obtains the $\beta$s from the $\alpha$s:

\[
\begin{pmatrix}
  a_{11} & \cdots & a_{1n} \\
  \vdots & \ddots & \vdots \\
  a_{m1} & \cdots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
  \alpha_1 \\
  \vdots \\
  \alpha_n
\end{pmatrix}
= 
\begin{pmatrix}
  \beta_1 \\
  \vdots \\
  \beta_m
\end{pmatrix}.
\]
Example: The ‘Frequency Shift’ Matrix

Assume both \( u \) and \( v \) are linear combination of sounds of different frequencies:

\[
\begin{align*}
\mathbf{u} &= \alpha_1 u_{110 \text{ Hz}} + \alpha_2 u_{220 \text{ Hz}} + \cdots + \alpha_4 u_{880 \text{ Hz}} \\
\text{(analogously for \( \mathbf{v} \), but with \( \beta \)s).}
\end{align*}
\]

What matrix realizes a ‘frequency doubling’ of a signal represented this way?

\[
\begin{pmatrix}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
\alpha_1 \\
\alpha_2 \\
\alpha_3 \\
\alpha_4
\end{pmatrix}
= 
\begin{pmatrix}
\beta_1 \\
\beta_2 \\
\beta_3 \\
\beta_4
\end{pmatrix}
\]
What are some examples of matrices in applications?

**Demo:** Matrices for geometry transformation (click to visit)

**Demo:** Matrices for image blurring (click to visit)

**In-class activity:** Computational Linear Algebra
How could this (directed) graph be written as a matrix?

\[ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \]
What is the general rule for turning a graph into a matrix?

If there is an edge from node $i$ to node $j$, then $A_{ji} = 1$.

(otherwise zero)

What does the matrix for an undirected graph look like?

Symmetric.
How could we turn a weighted graph (i.e. one where the edges have weights—maybe ‘pipe widths’) into a matrix?

Allow values other than zero and one for the entries of the matrix.
If we multiply a graph matrix by the \( i \)th unit vector, what happens?
We get a vector that indicates (with a 1) all the nodes that are reachable from node $i$. 
Demo: Matrices for graph traversal (click to visit)
Consider the following graph of states:

Suppose this is an accurate model of the behavior of the average student. :) Can this be described using a matrix?
Important assumption: Only the most recent state matters to determine probability of next state. This is called the Markov property, and the model is called a Markov chain.

Write transition probabilities into matrix as before:
(Order: surf, study, eat–‘from’ state along columns)

\[
A = \begin{pmatrix}
.8 & .6 & .8 \\
.2 & .3 & 0 \\
0 & .1 & .2 \\
\end{pmatrix}
\]

Observe: Columns add up to 1, to give sensible probability distribution of next states. Given probabilities of states
$p = (p_{\text{surf}}, p_{\text{study}}, p_{\text{eat}})$, $Ap$ gives us the probabilities after one unit of time has passed.
Some types of matrices (including graph matrices) contain many zeros. Storing all those zero entries is wasteful. How can we store them so that we avoid storing tons of zeros?

- Python dictionaries (easy, but not efficient)
- Using arrays...?
How can we store a sparse matrix using just arrays? For example:

\[
\begin{pmatrix}
0 & 2 & 0 & 3 \\
1 & 4 & & \\
& & 5 & \\
6 & 7 & &
\end{pmatrix}
\]

**Idea:** ‘Compressed Sparse Row’ (‘CSR’) format

- Write all non-zero *values* from top-left to bottom-right
- Write down what *column* each value was in
- Write down the index where each *row started*
Storing Sparse Matrices Using Arrays

RowStarts = \( \begin{pmatrix} 0 & 2 & 4 & 5 & 7 \end{pmatrix} \) (zero-based)

Columns = \( \begin{pmatrix} 1 & 3 & 0 & 1 & 2 & 0 & 3 \end{pmatrix} \) (zero-based)

Values = \( \begin{pmatrix} 2 & 3 & 1 & 4 & 5 & 6 & 7 \end{pmatrix} \)
Demo: Sparse Matrices in CSR Format (click to visit)