Previous class:
- User-defined function
- Nested loops

Now:
- Working with colors
- 1-dimensional array—vector
- Algorithm for finding the best item in a set

Generating tables and plots

\[
\begin{array}{cc}
 x & \sin(x) \\
 0.00 & 0.000 \\
 0.784 & 0.707 \\
 1.571 & 1.000 \\
 2.357 & 0.707 \\
 3.142 & 0.000 \\
 3.927 & -0.707 \\
 4.712 & -1.000 \\
 5.498 & -0.707 \\
 6.283 & 0.000
\end{array}
\]

Note: \( x, y \) are shown above as columns due to screen space; they are rows.

Built-in function \texttt{linspace}

\[
\begin{array}{c}
x= \text{linspace}(1,3,5) \\
x= \begin{bmatrix} 1.0 & 1.5 & 2.0 & 2.5 & 3.0 \end{bmatrix}
\end{array}
\]

\[
\begin{array}{c}
x= \text{linspace}(0,1,101) \\
x= \begin{bmatrix} 0.00 & 0.01 & 0.02 & \ldots & 0.99 & 1.00 \end{bmatrix}
\end{array}
\]

How did we get all the sine values?

Built-in functions accept arrays

\[
\begin{array}{c}
x= \text{linspace}(0,2*\pi,9) \\
x= \begin{bmatrix} 0.00 & 1.57 & 3.14 & 4.71 & 6.28 \end{bmatrix}
\end{array}
\]

\[
\begin{array}{c}
y= \sin(x) \\
\text{plot}(x,y)
\end{array}
\]

Vectorized addition

\[
\begin{array}{c}
x= \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
y= \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix}
\end{array}
\]

\[
\begin{array}{c}
z= x + y \\
= \begin{bmatrix} 3 & 3.5 & 9 \end{bmatrix}
\end{array}
\]

Matlab code: \texttt{z= x + y}

Vectorized subtraction

\[
\begin{array}{c}
x= \begin{bmatrix} 2 & 1.5 & 8 \end{bmatrix} \\
y= \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix}
\end{array}
\]

\[
\begin{array}{c}
z= x - y \\
= \begin{bmatrix} 1 & -1.5 & 7 \end{bmatrix}
\end{array}
\]

Matlab code: \texttt{z= x - y}
Vectorized multiplication

\[
\begin{array}{c}
a = 2 \ 1.5 \ 8 \\
x = b = [1 \ 2 \ 0 \ 1] \\
= c = 2 \ 2 \ 0 \ 8 \\
\end{array}
\]

Matlab code: `c = a .* b`

---

Vectorized element-by-element arithmetic operations on arrays

\[
\begin{align*}
&+ \quad - \\
&.* \quad ./ \\
\end{align*}
\]

A dot (.) is necessary in front of these math operators

---

Shift

\[
\begin{array}{c}
x = 3 \\
+ y = [2 \ 1 \ 5 \ 8] \\
= z = 5 \ 4 \ 3.5 \ 11 \\
\end{array}
\]

Matlab code: `z = x + y`

---

Reciprocate

\[
\begin{array}{c}
x = 1 \\
/ \quad - \\
\end{array}
\]

Matlab code: `z = x ./ y`

---

Color is a 3-vector, sometimes called the RGB values

- Any color is a mix of red, green, and blue
- Example:
  \[
  \text{colr} = [0 \ 0.6 \ 0]
  \]
- Each component is a real value in [0,1]
  - [0 0] is black
  - [1 1 1] is white
  - [0.2 0.2] is dark gray
  - [0.4 0.6 0.1] is a colorized hue
Mix two colors

Implement this function:

```matlab
function newc = mixEqual(c1,c2)
% Average colors c1 and c2.
% c1, c2, and newc are vectors.
% Display the three colors.
```

Let's show the "paint chips" from white to black

Name the script `white2black`

I-d array: vector

- An array is a named collection of like data organized into rows or columns
- A 1-d array is a row or a column, called a vector
- An index identifies the position of a value in a vector

```
score = [93 92 87 0 90 82]
1 2 3 4 5 6
```

Array index starts at 1

Let k be the index of vector x, then

- k must be a positive integer
- 1 <= k <= length(x)
- To access the kth element: x(k)

Accessing values in a vector

```
score = [93 99 87 80 85 82]
1 2 3 4 5 6
```

A few different ways to create a vector

(More later!)

```matlab
count = zeros(1,6)
x = linspace(10,30,5)
y = [3 7 2 1]
z = [3; 7; 2]
```
### Drawing a single line segment

\[ a = 0; \quad \% \text{x-coord of pt 1} \\
\] \[ b = 1; \quad \% \text{y-coord of pt 1} \\
\] \[ c = 5; \quad \% \text{x-coord of pt 2} \\
\] \[ d = 3; \quad \% \text{y-coord of pt 2} \\
\] \[ \text{plot}([a \ c], [b \ d], '-*') \]

### Drawing a polygon (multiple line segments)

% Draw a rectangle with the lower-left corner at (a,b), width w, height h.
\[ x = [a \ a+w \ a \ a]; \quad \% \text{x data} \]
\[ y = [b \ b \ b+h \ b+h \ b+h \ b+h \ b]; \quad \% \text{y data} \]
\[ \text{plot}(x, y) \]

### Coloring a polygon (fill)

% Draw a rectangle with the lower-left corner at (a,b), width w, height h, and fill it with a color named by c.
\[ x = [a \ a+w \ a \ a]; \quad \% \text{x data} \]
\[ y = [b \ b \ b+h \ b+h \ b]; \quad \% \text{y data} \]
\[ \text{fill}(x, y, c) \]

Built-in function `fill` actually does the "wrap-around" automatically.

### Example

- Write a program fragment that calculates the cumulative sums of a given vector \( v \).
- The cumulative sums should be stored in a vector of the same length as \( v \).

\[ v = 1, 3, 5, 0 \quad v \]
\[ 1, 4, 9, 9 \quad \text{cumulative sums of} \ v \]
Common loop pattern to process a vector

```matlab
% v is a given vector
for k = 1:length(v)
    % work with v(k)
end
```

A twinkling constellation

- Write a script that generates 9 random positions—the configuration of my constellation
- Simulate 10 rounds of twinkling
  - In each round, each star is equally likely to be lit or black
- Can you add some random adjustment to the color of the star?

Algorithm: Finding the best in a set

1. Init bestSoFar
2. Loop over set
   - if current is better than bestSoFar
     - bestSoFar = current
   end
end