Previous Lecture:
- Executing a user-defined function
- Function scope
- Subfunction

Today’s Lecture:
- 1-d array—vector
- Probability and random numbers
- Simulation using random numbers, vectors

Announcement:
- Project 3 due Monday 10/3 at 11pm
Outcomes from 1200 rolls of a fair die

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Roll 2 fair dice
What is the output?

```
x = 1;
x = f(x+1);
y = x+1;
disp(y)
```

```
function y = f(x)
x = x+1;
y = x+1;
```

A: 1  B: 2  C: 3  D: 4  E: 5
Execute the statement

\[ y = \text{foo}(x); \]

- Matlab looks for a function called \text{foo} (m-file called \text{foo.m})
- Argument (value of \text{x}) is copied into function \text{foo}'s local parameter
  - called “pass-by-value,” one of several argument passing schemes used by programming languages
- Function code executes within its own workspace
- At the end, the function’s output argument (value) is sent from the function to the place that calls the function. E.g., the value is assigned to \text{y}.
- Function’s workspace is deleted
  - If \text{foo} is called again, it starts with a new, empty workspace
1-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

\[ \begin{array}{ccc} 
1 & 2 & 3 \\
0.8 & 0.2 & 1 \\
\end{array} \]
Here are a few different ways to create a vector

```matlab
count = zeros(1, 6)
count = [0 0 0 0 0 0]
```

```
a = linspace(10, 30, 5)
a = [10 15 20 25 30]
b = 7:-2:0
b = [7 5 3 1]
c = [3 7 2 1]
c = [3 7 2 1]
d = [3; 7; 2]
d = [3; 7; 2]
e = d'
e = [3 7 2]
```

Similar functions: `ones`, `rand`
Start with drawing a single line segment

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

\[ a = [0, 4, 3, 8]; \]  \% x-coords

\[ b = [1, 2, 5, 3]; \]  \% y-coords

\[ \text{plot}(a, b, '-*') \]

- **x-values** (a vector)
- **y-values** (a vector)
- **Line/marker format**

![Graph example](image-url)
Making an x-y plot with multiple graphs (lines)

```matlab
a = [0 4 5 8];
b = [1 2 5 3];
f = [0 4 6 8 10];
g = [2 2 6 4 3];
plot(a,b,'-*',f,g,'c')
legend('graph 1 name', 'graph 2 name')
xlabel('x values')
ylabel('y values')
title('My graphs', 'Fontsize',14)
```
Array index starts at 1

Let $k$ be the index of vector $x$, then

- $k$ must be a positive integer
- $1 \leq k \leq \text{length}(x)$
- To access the $k^{th}$ element: $x(k)$
Accessing values in a vector

<table>
<thead>
<tr>
<th>score</th>
<th>93</th>
<th>92</th>
<th>87</th>
<th>0</th>
<th>90</th>
<th>82</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Given the vector **score** …
Accessing values in a vector

Given the vector `score` ...

\[ \text{score}(4) = 80; \]
\[ \text{score}(5) = (\text{score}(4) + \text{score}(5)) / 2; \]
\[ k = 1; \]
\[ \text{score}(k+1) = 99; \]

See `plotComparison2.m`
Example

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

1, 3, 5, 0 \hspace{1cm} \mathbf{v}

1, 4, 9, 9 cumulative sums of $\mathbf{v}$
\[ V \]
\[ cSum \]
\[ 1 \ 2 \ 3 \]

\[ cSum(k) = cSum(k-1) + V(k) \]

\[ cSum(3) = V(1) + V(2) + V(3) \]
\[ cSum(4) = V(1) + V(2) + V(3) + V(4) \]

\[ cSum(3) \]

\[ cSum(1) = V(1); \]
\[ \text{for} \quad k = 2 : \text{length}(V) \]
\[ \quad cSum(k) = cSum(k-1) + V(k); \]
\[ \text{end} \]
Random numbers

- Pseudorandom numbers in programming

- Function \texttt{rand(...)} generates random real numbers in the interval (0,1). All numbers in the interval (0,1) are equally likely to occur—uniform probability distribution.

- Examples:
  - \texttt{rand} \quad one random \# in (0,1)
  - 6*\texttt{rand} \quad one random \# in (0,6)
  - 6*\texttt{rand}+1 \quad one random \# in (1,7)
Uniform probability distribution in (0, 1)

Normal distribution with zero mean and unit standard deviation

\texttt{randn}
Simulate a fair 6-sided die

Which expression(s) below will give a random integer in [1..6] with equal likelihood?

A \[ \text{round}(\text{rand}*6) \]
B \[ \text{ceil}(\text{rand}*6) \]
C Both expressions above
(rand*6)
round(rand*6)
round(rand*6)
\texttt{round(rand*6)}
\texttt{round(rand*6)}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{round.png}
\end{figure}

\texttt{ceil(rand*6)}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ceil.png}
\end{figure}
\[\text{round}(\text{rand}*6)\]

\[\text{ceil}(\text{rand}*6)\]
Possible outcomes from rolling a fair 6-sided die

1  2  3  4  5  6
Simulation
Simulation
Simulation result

Data in bins

$\text{bar}(1:6, \text{ count})$

Bin numbers

<table>
<thead>
<tr>
<th>Bin</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>54</td>
</tr>
</tbody>
</table>
Keep tally on repeated rolls of a fair die

*Repeat the following:*

% roll the die

% increment correct “bin”
function count = rollDie(rolls)

FACES = 6;

% #faces on die
% Count outcomes of rolling a FAIR die
for k = 1:rolls
    % Roll the die
    % Increment the appropriate bin
end

% Show histogram of outcome