- Relational operators and logical variables
- Conditional statements
  - if-elseif-else
  - while
- Discussion of examples & HW problems [ran out of time; will add some HW discussion at end of this file.]
- Quiz #1 (last 30 minutes)

Note: powerpoint slides included are available on Blackboard under the appropriate file names.
## Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Less than.</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to.</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than.</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to.</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal to.</td>
</tr>
<tr>
<td><code>~=</code></td>
<td>Not equal to.</td>
</tr>
</tbody>
</table>

Values returned using relational operations are either `1` (true) or `0` (false).

Operator: 

Meaning:

**Relational Operators (Review)**
For example, suppose that $x = [6, 3, 9]$ and $y = [14, 2, 9]$. The following MATLAB session shows some examples.

```
>> z = (x < y)
z =
1   0   0

>> z = (x ~= y)
z =
1   1   0

>> z = (x > 8)
z =
0   0   1
```

The following MATLAB session shows some examples.

```
>> x = [14, 2, 9] and y = [6, 3, 9]

For example, suppose that $x = \{14, 2, 9\}$ and $y = \{6, 3, 9\}$ and $z = 1, 0, 0$.
```
The relational operators can be used for array addressing. For example, with \( x = [6, 3, 9] \) and \( y = [14, 2, 9] \), typing \( z = x(x < y) \) finds all the elements in \( x \) that are less than the corresponding elements in \( y \). The result is \( z = 6 \).

\[
\begin{align*}
\log_{10} \text{values of } x & = [1, 0, 0] \\
(\log_{10} x) x & = z
\end{align*}
\]

For example, with \( x = [6, 3, 9] \) and \( y = [14, 2, 9] \), typing \( z = x(y > x) \).

The relational operators can be used for array addressing.
The arithmetic operators +, -, *, /, and \ have precedence over the relational operators. Thus the statement
\[ z = 5 > 2 + 7 \]
evaluates to \( z = 8 \).

We can use parentheses to change the order of precedence; for example, \( z = (5 < 2 + 7) \) is equivalent to \( z = (5 < 2) + 7 \) which evaluates to \( z = 0 \).

\[ (5 < 2 + 7) \]

and returns the result \( z = 0 \).
The logical Class

When the relational operators are used, such as

\[ x = (5 > 2) \]

they create a logical variable, in this case, \( x \).

Prior to MATLAB 6.5, logical was an attribute of any numeric data.

Logical variables may have only the values 1 (true) and 0 (false).

Prior to MATLAB 6.5, logical was an attribute of any numeric data type. Now logical is a first-class data type and a MATLAB class, and so logical is now equivalent to other first-class types such as character and cell arrays.

When the relational operators are used, such as
Just because an array contains only 0s and 1s, however, it is not necessarily a logical array. For example, in the following session, \( k \) and \( w \) appear the same, but \( k \) is a logical array and \( w \) is a numeric array, and thus an error message is issued.

\[
\begin{align*}
[k] & = [1, 0, 0, 0, 1] \\
&w = [1, 0, 0, 0, 1, 0, 0, 0, 1, 0] \\
&x = \left\lfloor \frac{\text{abs}(x)}{2} \right\rfloor < 1 \\
&\text{abs}(x) = [2, 0, 1, 2] \\
&x = [2, 0, 1, 2] \\
&x = [1, 0, 0, 0, 1]
\end{align*}
\]

To correct this error, first type:

\[
\begin{align*}
&>> w = \text{logical}([1, 0, 0, 0, 1]) \\
&>> z = x(w)
\end{align*}
\]
Accessing Arrays Using Logical Arrays

When a logical array is used to address another array, it extracts from that array the elements in the locations where the logical array has 1s.

So typing `A(B)`, where `B` is a logical array of the same size as `A`, returns the values of `A` at the indices where `B` is 1.

Array `B` has 1s.

From that array the elements in the locations where the logical array has 1s.

When a logical array is used to address another array, it extracts

Accessing Arrays Using Logical Arrays...
Accessing Arrays Using Logical Arrays

Specifying array subscripts with logical arrays extracts the elements that correspond to the true (1) elements in the logical array.

```matlab
A = [5, 6, 7; 8, 9, 10; 11, 12, 13];
B = logical(eye(3));
C = A(B);
```

Given

```matlab
A = [5, 6, 7; 8, 9, 10; 11, 12, 13];
B = logical(eye(3));
C = A(B);
```

A by typing 

```matlab
C = A(B) to obtain
C = [5; 9; 13].
```

we can extract the diagonal elements of

```matlab
A
```

by typing

```matlab
C = A(B)
```

and obtain

```matlab
C = [5, 9, 13].
```

(continued)
Logical Operators and Functions
Operator Name Definition

~A returns an array the same dimension as A; the new array has ones where A is zero and zeros where A is nonzero.

A & B returns an array the same dimension as A and B; the new array has ones where both A and B have nonzero elements and zeros where either A or B is zero.

A | B returns an array the same dimension as A and B; the new array has ones where at least one element in A or B is nonzero and zeros where A and B are both zero.

>> A = [1 2 3]; B = [-2 0 7];
>> C = A & B
  C =
    1  0  1

>> C = A | B
  C =
    1  1  1

(continued …)
Operator Name Definition

Short-Circuit AND Operator for scalar logical expressions.

\[ A \&\& B \]

returns true if both \( A \) and \( B \) evaluate to true, and false if they do not.

\[ \text{Short-Circuit OR Operator for scalar logical expressions.}\]

\[ A \| B \]

returns true if either \( A \) or \( B \) or both evaluate to true, and false if they do not.

\[ \text{Exclusive OR} \]

\[ \text{ xor } \]

is the logical symmetric difference of \( S \) and \( T \). The result is 1 where either \( S \) or \( T \), but not both, is nonzero. The result is 0 where \( S \) and \( T \) are both zero or nonzero.

Elements \( S \) and \( T \). The result is 1 where either \( S \) or \( T \) is nonzero.
The above table can be generated from MATLAB directly:

```matlab
>> x = [1 1 0 0]';
>> y = [1 0 1 0]';
>> Truth_Table = [x, y, ~x, x|y, x&y, xor(x,y)]
```

```
Truth_Table =
1     1     0     1     1     0
1     0     0     1     0     1
1     0     1     0     0     1
0     1     1     0     1     0
```

The truth table is:

<table>
<thead>
<tr>
<th>~x</th>
<th>x</th>
<th>y</th>
<th>x&amp;y</th>
<th>x</th>
<th>y</th>
<th>xor(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>true</td>
</tr>
</tbody>
</table>

**Truth Table**

<table>
<thead>
<tr>
<th>~x, y</th>
<th>x&amp;y</th>
<th>x</th>
<th>y</th>
<th>xor</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>
2. Consider the following piecewise function:

\[ f(t) = \begin{cases} 
6t^2 - 10t & 0 \leq t \leq 10 \\
600 - 10t & 10 < t \leq 20 \\
(t - 40)^2 & t \geq 20 \\
0 & \text{otherwise}
\end{cases} \]

Complete the following script file to compute \( f \) as a function of \( t \) using:

(a) Logical functions (if, else, etc)
(b) Relational operators (>, <, etc).

Solution:

(a)
\[
t = 5:50; \ N = \text{length}(t);
\]
\[
\text{for } i = 1: N
\]
\[
\text{if } t(i) < 0; \ f(i) = 0;
\]
\[
\text{elseif } t(i) > 0 \& t(i) \leq 10; \ f(i) = 6 \times t(i)^2 - 10 \times t(i);
\]
\[
\text{elseif } t(i) > 10 \& t(i) \leq 20; \ f(i) = 600 - 10 \times t(i);
\]
\[
\text{else } t(i) > 20; \ f(i) = (t(i) - 40)^2;
\]
\[
\text{end}
\]

(b)
\[
t = 5:50; \ N = \text{length}(t);
\]
\[
\text{for } i = 1: N
\]
\[
f(i) = ((t(i) > 0) \& (t(i) \leq 10)) \times (6 \times t(i)^2 - 10 \times t(i)) + ... \\
(((t(i) > 10) \& (t(i) \leq 20)) \times (600 - 10 \times t(i)) + (t(i) > 20) \times (t(i) - 40)^2;
\]
\[
\text{end}
\]

This second approach seems to be less efficient, but more compact in numerical coding.
Example using a function
(from Monday’s lecture)

1. Consider the following piecewise function in which \( v(t) \) is given in cm/s and \( t \) in seconds.

\[
v(t) = \begin{cases} 
4t^2 - 10t & 0 \leq t \leq 10 \\
400 - 10t & 10 \leq t \leq 20 \\
(t - 5)^2 - 25 & t \geq 20 \\
0 & \text{otherwise}
\end{cases}
\]

Given the following m-file to generate a plot of \( v(t) \) versus \( t \) for \( t = -10 \) to 40, develop a m-file function for \texttt{funv}.

\[
t = -10:0.1:40; \ % \text{define an array variable } t \\
v = \texttt{funv}(t); \ % \text{define array variable } v \text{ using function } \texttt{funv}
\]

\[
\text{plot}(t,v)
\]

**Solution:**

function vel = funv(t)
% This function defines the piecewise continuous function
% Note that the input t may be an array of numbers
% 
% n = length(t);
for k = 1:n
if t(k) <= 10 & t(k) >= 0
    vel(k) = 4*t(k)^2 - 10*t(k);
elseif t(k) <= 20 & t(k) >= 10
    vel(k) = 400 - 10*t(k);
elseif t(k) >= 20
    vel(k) = (t(k) - 5)^2 - 25;
else
    vel(k) = 0;
end
end

\[\text{this is necessary because of the nature of the relational operators} \]

1. (10 points) The singularity function of order $n$ is

$$
(x-a)^n = \begin{cases} 
(x-a)^n & \text{when } x > a \\ 
0 & \text{when } x \leq a 
\end{cases}
$$

A student wrote the following MATLAB function code to define the singularity function.

```matlab
function z = fsgl(a,n,x)
% This evaluates the singularity function <x-a> to power of n.
% x is a numeric array; a and n are both scalars.
if x <= a
    z = 0;
elseif x > a
    z = (x-a).^n;
else disp('Error in the input arguments')
end
```

a) Consider $x = [1 \ 2 \ 3]$. Write down the results when each of the followings is executed. Briefly explain your results.

```matlab
>> fsgl(0,3,x)
```

```
ans = [1 \ 8 \ 27]  
because (x > a) is a true statement
```

```matlab
>> fsgl(2,3,x)
```

```
Error message will be displayed
\[ x = [1 \ 2 \ 3] \]
\[ x < 2 \rightarrow \text{False} \]
\[ x > 2 \rightarrow \text{False} \]
```
b) Modify the above code of `fsgl` so that it correctly defines the singularity function (note that the input parameter `x` is an array).

```matlab
function z = fsgl(a,n,x)
% This evaluates the singularity function <x-a> to power of n.
% x is a numeric array; a and n are both scalars.
% N = length(x);
for i = 1:N
    if x(i) <= a
        z(i) = 0;
    elseif x(i) > a
        z(i) = (x(i)-a)^n;
    else disp('Error in the input arguments');
    end
end
```

If \( x = [1 \ 2 \ 3] \), \( a=2 \), \( n=3 \)
the expected result is \( z = [0 \ 0 \ 1] \)
we need to check element by element
add in the code \( N=\text{length}(x) \);
for \( i=1: N \)
The while loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance. A simple example of a while loop is:

```matlab
x = 5;
while x < 25
    disp(x)
    x = 2*x - 1;
end
```

The results displayed by the `disp` statement are 5, 9, and 17.

The while loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance. A simple example of a while loop is:
The typical structure of a while loop follows.

```matlab
while logical expression
    statements
end
```

For the `while` loop to function properly, the following two conditions must occur:

1. The loop variable must have a value before the `while` statement is executed.
2. The loop variable must be changed somehow by the `end` statements.

The typical structure of a `while` loop follows.
Flowchart of the while loop
A simple example of a while loop is

```matlab
x = 5; k = 0;
while x < 25
    k = k + 1;
y(k) = 3*x;
x = 2*x - 1;
end
```

The loop variable `x` is initially assigned the value 5, and it keeps this value until the statement `x = 2*x - 1` is encountered the first time. Its value then changes to 9. Before each pass through the loop, `x` is checked to see if its value is less than 25. If so, the pass is made; if not, the loop is skipped.
Another example of a while loop

Write a script file to determine how many terms are required for the sum of the series $5k^2 - 2k$, $k = 1, 2, 3, \ldots$ to exceed 10,000. What is

$\text{sum}_{\text{total}} = 0; K = 0$!

while total $\geq 1e+4$

set index to be 0

$\text{sum}$ is initialized to be zero.

$\sum_{n=1}^{\infty} \text{so that } 5 > 10,000$

$k = 1$

$5k^2 - 2k$

end

disp('The number of terms is:');
disp(k)
disp('The sum is:');
disp(total)

The number of terms is: 18
The sum is: 10,203
1. (40 points) Write a MATLAB function file by using a while loop to determine the largest value of \( N \) such that the series

\[
\sum_{k=1}^{N} 2k^2 + k
\]

does not exceed 10,000. In the function file, there should be an error message display if the number of terms \( N < 1 \).

function N = series(maxsum)
% This function file determines the largest number of terms N
% in a series such that the series sum is less than "maxsum".
% For this problem, maxsum = 10,000.

k = 1;
sum = 0;

while sum <= maxsum
    sum = sum + 2*k^2 + k;
    if sum > maxsum
        sum = sum - 2*k^2 + k; % optional: obtain largest sum <= maxsum
        break
    end
    k = k + 1;
end

N = k - 1;

if N < 1
    error('maxsum is too small and N < 1')
end

Test it in MATLAB command window:

```matlab
>> N = series(10000)
N =
    23
```

**Check:** The sum for 23 terms is 8924 and the sum for 24 terms is 10,100. Hence, the program gives the correct answer of 23 terms.
Problem on m6.20

\[ RV = \left| \frac{V_0}{V_i} \right| = \frac{\omega R C}{\sqrt{(1-\omega^2 L C)^2 + (\omega R C)^2}} \]

where \( \omega \) = frequency of input signal

Write a user-defined function: \( RV = \text{bandpass}(R, C, L, \omega) \)

\( R, C, L \) = system parameters (constants)
\( \omega \) = a data array of signal frequencies

Then write a script file to call the function "bandpass".

```
function RV = bandpass(R, C, L, omega)
    RV = omega*R*C/sqrt((1-omega.^2*L*C).^2 + omega.^2*C^2);
end
```

If too long, practice using "..." as a continuation in Matlab.

Script file:

```matlab
% add your comments
% Input system parameters:
% R = resistance in ohms (\Omega)
% C = capacitance in Farads (F)
% L = inductance in Henry (H)

R = input('Enter the value of R in ohms = ')  
C = ----
L = ----
```
Generates a data array for omega using logspace
\[ \text{omega} = \text{logspace} \left(-2, 7, 100\right); \]
\[ \text{RV} = \text{bandpass} \left( R, \text{c, L, omega} \right) \]
calls the function "bandpass"

The set of data points is linear in log-scale for \(10^{-2} \leq \omega \leq 10^{7}\).

```matlab
semilogx(omega, RV)
xlabel('Frequency \omega (rad/s)')
ylabel('Voltage Ratio')
grid
```

In command window
```matlab
>> m6-20
```

Enter the value of \(R\) in ohms =

---

---

(type in the value after this prompt)
Problem m 7.14 (to be discussed in next lecture)

Taylor’s series for cosine function

\[ \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} \]

Write a user-defined function \( y = \text{cosTaylor}(x) \), where \( x \) is an input data array in degrees. Stop adding terms to the series when the relative error \( E \leq 10^{-6} \). Use Matlab built-in function “factorial” (or user-defined function in Problem 13 as a subfunction).

```matlab
function y = cosTaylor(x)

% add in your own comments here....

% Etol = Error tolerance
% nmax = maximum value of n (number of terms in series)

% nmax is optional but it prevents the program to % run into an infinite loop should there be a mistake % in checking the relative error E.

global Etol nmax  % define these variables as global in scope. Enter these values in Command Window.

% or simply type in values here:
Etol = 1e-6; nmax = ...;
```
% convert X to radians
X = X * pi/180;

yold = 1 ← initialise the series sum to be equal to 1

for n = 1: nmax
    y = yold + (-1)^n * x.^((2*n))/factorial(2*n);
    E = abs(y - yold)/abs(yold);
    if E <= Etol, break, end
    yold = y;
end

if n == nmax
    fprintf('Maximum number of terms used in the series with error still > Etol')
end

to tell the user that more terms are needed to achieve the required accuracy.

One can also use the "while" loop to accomplish the above objectives. Think about the "while" loop.
Next lecture: discuss pseudo-code approach.