Previous Lecture:
- Iteration using `for`

Today’s Lecture:
- Iteration using `while`
- Review loops, conditionals using graphics

Announcements:
- Did you watch MatTV episode “Troubleshooting Loops”? Available on course website
- Project 2 due Monday, 2/29
- We do not use `break` in this course
- Read Insight Section 3.2 before your discussion section next week
- Come to office/consulting hours to get help! “Limited tutoring” scheduled periodically—check CMS. See also “Additional Help” link on course website.
Syntax of the `for` loop

```
for <var> = <start value>:<incr>:<end bound>
  statements to be executed repeatedly
end
```

Loop header specifies all the values that the index variable will take on, one for each pass of the loop.

E.g, `k = 3:1:7` means `k` will take on the values 3, 4, 5, 6, 7, one at a time.
Pattern for doing something $n$ times

\[
\begin{align*}
n &= \_\_\_ \\
\text{for } &k = 1:1:n \\
\text{\% code to do} \\
\text{\% that something} \\
\text{end}
\end{align*}
\]
for loop examples

for k = 2:0.5:3
    disp(k)
end

k takes on the values 2,2.5,3
Non-integer increment is OK

for k = 1:4
    disp(k)
end

k takes on the values 1,2,3,4
Default increment is 1

for k = 0:-2:-6
    disp(k)
end

k takes on the values 0,-2,-4,-6
“Increment” may be negative

for k = 0:-2:-7
    disp(k)
end

k takes on the values 0,-2,-4,-6
Colon expression specifies bounds

for k = 5:2:1
    disp(k)
end
for loop examples

for k = 2:0.5:3
    disp(k)
k takes on the values 2, 2.5, 3
Non-integer increment is OK
end

for k = 1:4
    disp(k)
k takes on the values 1, 2, 3, 4
Default increment is 1
end

for k = 0:-2:-6
    disp(k)
“Increment” may be negative
end

for k = 0:-2:-7
    disp(k)
Colon expression specifies bounds
end

for k = 5:2:1
    disp(k)
The set of values for k is the empty set: the loop body won’t execute
end
% What will be printed?
for k = 10:-1:14
    fprintf('%d ', k)
end
fprintf('!')

A: error
   (incorrect bounds)
B: 10 (then error)
C: 10 !
D: 14 !
E: !
Example: $n$-gon $\rightarrow$ circle

Inscribed hexagon
$(n/2) \sin(2\pi/n)$

Circumscribed hexagon
$n \tan(\pi/n)$

As $n$ approaches infinity, the inscribed and circumscribed areas approach the area of a circle.

When will $|\text{OuterA} - \text{InnerA}| \leq .000001$?
Find $n$ such that $\text{outerA}$ and $\text{innerA}$ converge

First, itemize the tasks:
- define how close is close enough
- select an initial $n$
- calculate $\text{innerA}$, $\text{outerA}$ for current $n$
- $\text{diff} = \text{outerA} - \text{innerA}$
- close enough?
- if not, increase $n$, repeat above tasks
Find \( n \) such that \( \text{outerA} \) and \( \text{innerA} \) converge

Now organize the tasks \( \rightarrow \) algorithm:

\( n \) gets initial value

Repeat until difference is small:

increase \( n \)

calculate \( \text{innerA} \), \( \text{outerA} \) for current \( n \)

\( \text{diff} = \text{outerA} - \text{innerA} \)
Find \( n \) such that \( \text{outerA} \) and \( \text{innerA} \) converge

Now organize the tasks \( \rightarrow \) algorithm:

\( n \) gets initial value
\( \text{innerA}, \text{outerA} \) get initial values

Repeat until difference is small:
- increase \( n \)
- calculate \( \text{innerA}, \text{outerA} \) for current \( n \)
- \( \text{diff} = \text{outerA} - \text{innerA} \)
Find $n$ such that $outerA$ and $innerA$ converge

$n$ gets initial value
calculate $innerA$, $outerA$ for current $n$
while <difference is not small enough>
  increase $n$
calculate $innerA$, $outerA$ for current $n$
  $diff = outerA - innerA$
end

areaCircle.m
Guard against infinite loop

Use a loop guard that guarantees termination of the loop. Or just limit the number of iterations.

```latex
while (B_n - A_n > \text{delta} \land n < nMax)
```
Another use of the while-loop: user interaction

- Example: Allow a user to repeatedly calculate the inscribed and circumscribed areas of n-gons on a unit circle.
- Need to define a “stopping signal”
Common loop patterns

**Do something \( n \) times**

\[
\textbf{for} \ k= 1:1:n \\
\quad \% \text{ Do something } \\
\textbf{end}
\]

**Do something an indefinite number of times**

\[
\% \text{Initialize loop variables} \\
\textbf{while} \ ( \text{not stopping signal} \ ) \\
\quad \% \text{ Do something } \\
\quad \% \text{ Update loop variables} \\
\textbf{end}
\]
Important Features of Iteration

- A task can be accomplished if some steps are repeated; these steps form the loop body
- Need a starting point
- Need to know when to stop
- Need to keep track of (and measure) progress
In Matlab, which claim is true? (without `break`)

A: for-loop can do anything while-loop can do

B: while-loop can do anything for-loop can do

C: for- and while-loops can do the same things
Common loop patterns

Do something \( n \) times

\[
\text{for } k = 1:1:n \\
\quad \text{\% Do something} \\
\text{end}
\]

Do something an indefinite number of times

\[
\text{\% Initialize loop variables} \\
\text{\% Do something} \\
\text{\% Update loop variables} \\
\text{end} \\
\text{while ( not stopping signal )}
\]
Pattern to do something \text{n} times

\begin{verbatim}
for k = 1:1:n
    % Do something
end
\end{verbatim}

\begin{verbatim}
% Initialize loop variables
k = 1;
while ( k <= n )
    % Do something
    % Update loop variables
    k = k + 1;
end
\end{verbatim}
for-loop or while-loop: that is the question

- **for-loop**: loop body repeats a *fixed* (predetermined) number of times.

- **while-loop**: loop body repeats an *indefinite* number of times under the control of the “loop guard.”