Today’s topics

- Review of topics for Test 1
- Characters and strings

Announcements/Reminders:

- Assignment 1b due Tuesday 11:59pm
- Test 1 in class on Wednesday
- Assignment 1a re-submission due Friday 11:59pm
The **if** construct

```plaintext
if  boolean expression1
    statements to execute if  expression1 is true
elseif  boolean expression2
    statements to execute if  expression1 is false
    but  expression2 is true
:
else
    statements to execute if all previous conditions
    are false
end
```

Can have any number of elseif branches but at most one else branch.
Generating random numbers

- \texttt{rand}(m,n) gives an m-by-n matrix of random values, each in interval (0,1)

- Generate a random number in the range (a,b)

- Generate a random integer in the range [a,b]
Generating random numbers

- \texttt{rand(m,n)} gives an m-by-n matrix of random values, each in interval (0,1)

- Generate a random number in the range (a,b)
  \[ \texttt{rand} \times (b-a) + a \]

- Generate a random integer in the range [a,b]
  \[ \text{floor} \left( \texttt{rand} \times (b-a+1) + a \right) \]
  \[ \text{ceil} \left( \texttt{rand} \times (b-a+1) + a-1 \right) \]
Built-in functions for creating/manipulating arrays

- **Creation**
  - zeros, ones, rand
  - linspace
- **Manipulation**
  - length
  - size
Built-in functions for creating/manipulating arrays

- **Creation**
  - zeros, ones, rand
  - linspace

- **Manipulation**
  - length
  - size

- \[ \text{zeros}(3, 2) \]
- \[ \text{linspace}(\frac{3}{3}, 10, 8) \]
- \[ \text{size}(M) \]
- \[ a = \begin{bmatrix} 4 & 2 & 3 \end{bmatrix} \text{ones}(2, 3) \]
Common loop patterns

Do something \( n \) times

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

Do something an indefinite number of times

\[
\% \text{Initialize loop variables} \\
\text{while ( not stopping signal )} \\
\% \text{ Do something} \\
\% \text{ Update loop variables} \\
\text{end}
\]
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 a bound

for k = 5:2:1
    disp(k)
end
The set of values for k is the empty set: the loop body won’t execute
for k = 4:6
    disp(k)
k= 9;
disp(k)
end

Not a condition (boolean expression) that checks whether k<=6.

It is an expression that specifies values:

4 5 6
Example

- Write a function `evalPoly` to evaluate an $n^{\text{th}}$ order polynomial of $x$:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- Input parameter `coef` has length $n+1$, contains the coefficients of the polynomial
- `coef(1)` is the coefficient for the term $x^0$
- Input parameter `x`
- Return the value of the polynomial evaluated at `x`
- No Matlab predefined function other than `length`
\( \text{coeff} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \) \( c_1 x^0 + c_2 x^1 + c_3 x^2 + c_4 x^3 \)
\begin{align*}
\text{function} & \quad \text{val} = \text{evalPoly} \left( \text{coef}, x \right) \\
\% \text{ val is polynomial evaluated at } x \\
\% \text{ coef is a vector where coef}(1) \text{ is for term } x^0 \\
\% \quad \text{xpow} = 1; \\
\text{val} & = \text{coef}(1) \\
\text{for } & \quad k = 2: \text{length(coef)} \\
\text{val} & = \text{val} + \text{coef}(k) \times x^{(k-1)}; \\
\% \quad \text{xpow} = \text{xpow} \times x; \\
\% \quad \text{val} = \text{val} + \text{coef}(k) \times \text{xpow}; \\
\text{end}
\end{align*}
Simulation problem:

- Ann and Bob take turns flipping an unfair coin—twice as likely to be heads than tails
- In one round, each player flips once
- Ann gets 1 point if she gets heads; Bob gets 2 points if he gets tails
- Game ends after the round in which at least one player gets 10 points. Display the final scores.
Simulation problem: \( \text{Stop} : pA > -10 \text{ or } pB > -10 \)

- Ann and Bob take turns flipping an unfair coin—twice as likely to be heads than tails
- In one round, each player flips once
- Ann gets 1 point if she gets heads; Bob gets 2 points if he gets tails
- Game ends after the round in which at least one player gets 10 points. Display the final scores.

\[
\begin{align*}
pA &= 0; \quad pB = 0; \\
\text{while } & \ pA < 10 \text{ and } pB < 10 \\
\% & \text{ 1 round: A flips; B flips; scoring} \\
r & = \text{rand}, \\
\text{if } & \ r < \frac{2}{3} \\
\quad pA & = pA + 1 \\
\end{align*}
\]

\[
\begin{align*}
r & = \text{rand}, \\
\text{if } & \ r = \text{rand} \\
\quad pB & = pB + 2 \\
\end{align*}
\]

Display \( \text{play} \)
Write a function triSums to return the column sums of the largest lower left triangular part of matrix M (same number of elements on each side of the triangle; including the main diagonal if matrix is square)
Write a function triSums to return the column sums of the largest lower left triangular part of matrix M (same number of elements on each side of the triangle; including the main diagonal if matrix is square)

```matlab
function colsum = triSums(M)
    [nr, nc] = size(M);
    minD = min(nr, nc);
    colsums = zeros(1, minD);
    for c = 1:nc-minD
        for r = 0:nr
            colsum(c) = colsum(c) + 
        end
    end
end
```
Function header is the “contract” for how the function will be used (called)

You have this function:

```matlab
function [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x,y).  Theta in degrees.
...```

Code to call the above function:

```matlab
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1 = 1;  t1 = 30;
[x1, y1] = polar2xy(r1, t1);
polar2xy(r1, t1);
plot(x1, y1, 'b*')
...```