Random numbers

- **Pseudorandom** numbers in programming
- Function `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:
  - `rand(1)` one random # in (0,1)
  - `6*rand(1)` one random # in (0,6)
  - `6*rand(1)+1` one random # in (1,7)

Simulate a fair 6-sided die

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

A. `round(rand*6)`
B. `ceil(rand*6)`
C. Both expressions above
Keep tally on repeated rolls of a fair die

Repeat the following:

\[
\begin{align*}
% \text{ roll the die} \\
% \text{ increment correct "bin"}
\end{align*}
\]

```matlab
function count = rollDie(rolls)
    FACES= 6;               % # faces on die
    count= zeros(1,FACES);  % bins to store counts
    % Count outcomes of rolling a FAIR die
    for k= 1:rolls
        % Roll the die
        % Increment the appropriate bin
    end
    % Show histogram of outcome
end
```

% Simulate the rolling of 2 fair dice
totalOutcome = ???

A. \(\text{ceil}(\text{rand} \times 12)\)
B. \(\text{ceil}(\text{rand} \times 11) + 1\)
C. \(\text{floor}(\text{rand} \times 11) + 2\)
D. 2 of the above
E. None of the above

2-dimensional random walk

Start in the middle tile, (0,0).

For each step, randomly choose between N,E,S,W and then walk one tile. Each tile is 1×1.

Walk until you reach the boundary.

Another representation for the random step

- Observe that each update has the form
  \[
  x_c = x_c + \Delta x \\
  y_c = y_c + \Delta y
  \]

  no matter which direction is taken.
- So let’s get rid of the if statement!
- Need to create two “change vectors” \(\Delta x\) and \(\Delta y\)

See `RandomWalk2D_v2.m`
Example: polygon smoothing

Can store the x-y coordinates in vectors x and y

First operation: centralize

Move a polygon so that the centroid of its vertices is at the origin

Before

After

function [xNew,yNew] = Centralize(x,y)
% Translate polygon defined by vectors x,y such that the centroid is on the origin. New polygon defined by vectors xNew,yNew.

n = length(x);
xBar = sum(x)/n;   yBar = sum(y)/n;
xNew = zeros(n,1); yNew = zeros(n,1);
for k = 1:n
    xNew(k) = x(k)-xBar;
    yNew(k) = y(k)-yBar;
end

sum returns the sum of all values in the vector

Second operation: normalize

Shrink (enlarge) the polygon so that the vertex furthest from the (0,0) is on the unit circle

Before

After

function [xNew,yNew] = Normalize(x,y)
% Resize polygon defined by vectors x,y such that distance of the vertex furthest from origin is 1

Third operation: smooth

Obtain a new polygon by connecting the midpoints of the edges
function [xNew,yNew] = Smooth(x,y)
% Smooth polygon defined by vectors x,y
% by connecting the midpoints of
% adjacent edges
n = length(x);
xNew = zeros(n,1);
yNew = zeros(n,1);
for i=1:n
    xNew(i) = (x(i) + x(i+1))/2;
yNew(i) = (y(i) + y(i+1))/2;
end

Polygon Smoothing

% Given x, y
for i=1:n
    xNew(i) = (x(i) + x(i+1))/2;
yNew(i) = (y(i) + y(i+1))/2;
end

Does above fragment compute the new n-gon?
A: Yes  
B: No

Show a simulation of polygon smoothing
Create a polygon with randomly located vertices.
Repeat:
    Centralize
    Normalize
    Smooth

Loop patterns for working with a vector
% Given a vector v
for k = 1:length(v)
    % Work with v(k)
    % E.g., disp(v(k))
end
% Given a vector v
k = 1;
while k <= length(v)
    % Work with v(k)
    % E.g., disp(v(k))
    k = k+1;
end