Previous Lecture:
- Finite vs. infinite; discrete vs. continuous
- `plot`, `fill`
- Intro to 1-d array and vectorized code

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

Announcements:
- Project 3 due Friday 3/1 at 11pm
- Discussion this week in Upson B7 lab
- Prelim 1 on Mar 7th at 7:30pm. Email Randy Hess (rbh27) now if you have an exam conflict (specify conflicting course and instructor contact info)
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>200</td>
</tr>
<tr>
<td>2</td>
<td>180</td>
</tr>
<tr>
<td>3</td>
<td>230</td>
</tr>
<tr>
<td>4</td>
<td>190</td>
</tr>
<tr>
<td>5</td>
<td>170</td>
</tr>
<tr>
<td>6</td>
<td>210</td>
</tr>
</tbody>
</table>

Roll 2 fair dice
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: \( x(k) \)
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 \( \mathbf{v} \).

\[
1, 3, 5, 0 \quad \mathbf{v}
\]

\[
1, 4, 9, 9 \quad \text{cumulative sums of } \mathbf{v}
\]
\[ \text{csum}(3) = v(1) + v(2) + v(3) \]
\[ \text{csum}(4) = v(1) + v(2) + v(3) + v(4) \]

\[ \text{csum}(k) = \text{csum}(k-1) + v(k) \]

\[ \text{csum}(1) = v(1); \]
for \( k = 2 : \text{length}(v) \)
\[ \text{csum}(k) = \text{csum}(k-1) + v(k); \]
end
Example

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

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

- 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`
\[ \text{coef} \quad 1 \quad 2 \quad 3 \quad 4 \]

\[ c_1 x^0 + c_2 x^1 + c_3 x^2 + c_4 x^3 \]
function val = evalPoly (coef, x)

% val is polynomial evaluated at x
% coef is a vector where coef(1) is for term x^0

% xpow = 1;
val = coef(1)
for k = 2: length (coef )
val = val + coef(k) * x^(k-1);
% xpow = xpow * x;
% val = val + coef(k) * xpow;
end
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:
  
  \[
  \begin{align*}
  \text{rand}(1) & \quad \text{one random \# in \((0, 1)\)} \\
  6\times\text{rand}(1) & \quad \text{one random \# in \((0, 6)\)} \\
  6\times\text{rand}(1)+1 & \quad \text{one random \# in \((1, 7)\)}
  \end{align*}
  \]
Uniform probability distribution in (0, 1)

\texttt{rand}

Normal distribution with zero mean and unit standard deviation \texttt{randn}

Distribution of \texttt{randn(1000000,1)}
Simulate a fair 6-sided die

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

A. $\text{round}(\text{rand} \times 6)$
B. $\text{ceil}(\text{rand} \times 6)$
C. *Both expressions above*
(rand(1) * 6)
round(rand(1) * 6)
round(rand(1) * 6)
\[ \text{round(rand(1) * 6)} \]

\[ (\text{rand(1) * 6}) \]
\text{round}(\text{rand}(1) \times 6)

\text{ceil}(\text{rand}(1) \times 6)
\text{round}(\text{rand}(1) \times 6)

\text{ceil}(\text{rand}(1) \times 6)
Possible outcomes from rolling a fair 6-sided die
Simulation

1  2  3  4  5  6
Simulation

1  2  3  4  5  6
Simulation

1  2  3  4  5  6

3
Simulation
Simulation

1  2  3  4  5  6
Simulation
Simulation
Simulation
### Simulation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simulation

1
2
3
4
5
6
Simulation
Simulation
Simulation
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 = 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
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
    face = ceil(rand*FACES);
    % Increment the appropriate bin
end

% Show histogram of outcome
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
    face = ceil(rand*FACES);
    % Increment the appropriate bin
    count(face) = count(face) + 1;
end

% Show histogram of outcome
% Count outcomes of rolling a FAIR die

count = zeros(1,6);
for k = 1:100
    face = ceil(rand*6);
    if face == 1
        count(1) = count(1) + 1;
    elseif face == 2
        count(2) = count(2) + 1;
    :;
    elseif face == 5
        count(5) = count(5) + 1;
    else
        count(6) = count(6) + 1;
    end
end
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
    face = ceil(rand*FACES);
    % Increment the appropriate bin
    count(face) = count(face) + 1;
end

% Show histogram of outcome
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 $1 \times 1$.

Walk until you reach the boundary.
function [x, y] = RandomWalk2D(N)
% 2D random walk in 2N-1 by 2N-1 grid.
% Walk randomly from (0,0) to an edge.
% Vectors x,y represent the path.
function [x, y] = RandomWalk2D(N)

k=0; xc=0; yc=0;

while not at an edge
    % Choose random dir, update xc, yc

    % Record new location in x, y

end
function [x, y] = RandomWalk2D(N)

k=0; xc=0; yc=0;

while abs(xc)<N && abs(yc)<N
    % Choose random dir, update xc, yc
    % Record new location in x, y

end
function [x, y] = RandomWalk2D(N)

k=0; xc=0; yc=0;

while abs(xc)<N && abs(yc)<N
  \% Choose random dir, update xc,yc

  % Record new location in x, y
  k=k+1; x(k)=xc; y(k)=yc;

end
% Standing at (xc, yc)
% Randomly select a step
    r = rand(1);
    if r < .25
        yc = yc + 1;  % north
    elseif r < .5
        xc = xc + 1;  % east
    elseif r < .75
        yc = yc - 1;  % south
    else
        xc = xc - 1;  % west
end
RandomWalk2D.m