Announcements

- Assignment 1 due Sep 13
  - Submit what you have by the deadline to avoid penalizing next week's resubmission

Su	M	Tu	W	Th	F	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Agenda

- Applications of vectors and probability
- How to plot data beyond points
  - Bar charts, lines
- How to populate vectors efficiently
- How to store 2D data
  - Matrices

Example: cumulative sum

- Write a program fragment that calculates the cumulative sums of a given vector v.
- The cumulative sums should be stored in a vector of the same length as v.

`1, 3, 5, 0` v

`1, 4, 9, 9` cumulative sums of v

```
csum(1) = v(1)
csum(2) = v(1) + v(2)          = csum(1) + v2
csum(3) = v(1) + v(2) + v(3)   = csum(2) + v3
csum(k)                        = ???
```

Rolling dice

- Problem: watch for loaded dice being used at Casino Night
- Solution: write a program to visualize how even the odds are

- Questions
    - How should the data be recorded?
    - How many rolls will it take before the data should look fair?
    - How do I know my program will work during the big event?
- Approach: simulation!

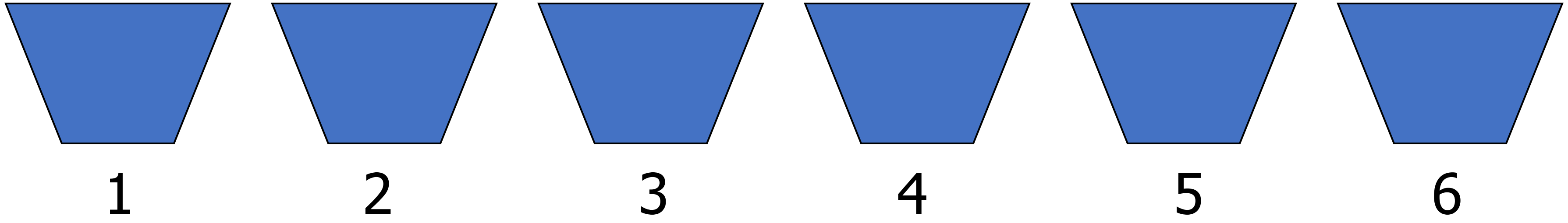
Program design: step 1

```
% Collect data
Repeat:
    Roll die
    Increment corresponding "bin"

% Visualize results
Draw bar for each bin with height $\propto$ bin count
```

How to keep track of results

Possible outcomes from rolling a fair 6-sided die

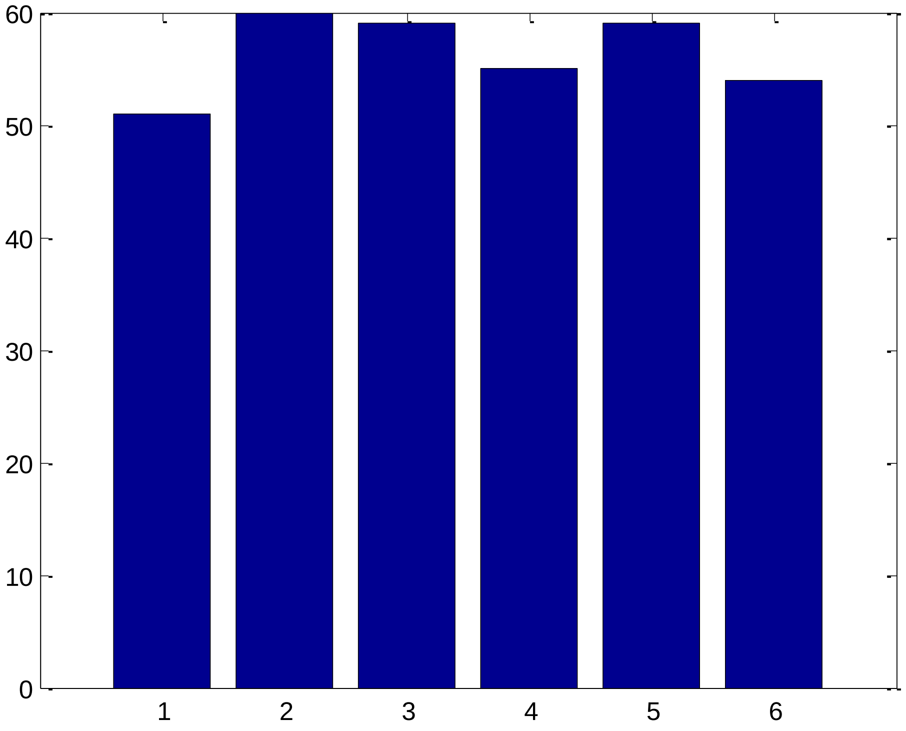


Simulation result

Data in bins

```
bar(1:6, counts)
```

Bin numbers



counts						
	51	60	59	55	59	54
	1	2	3	4	5	6

```
function counts = rollDie(rolls)

FACES= 6;               % #faces on die
counts= 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
bar(1:FACES, counts)
```

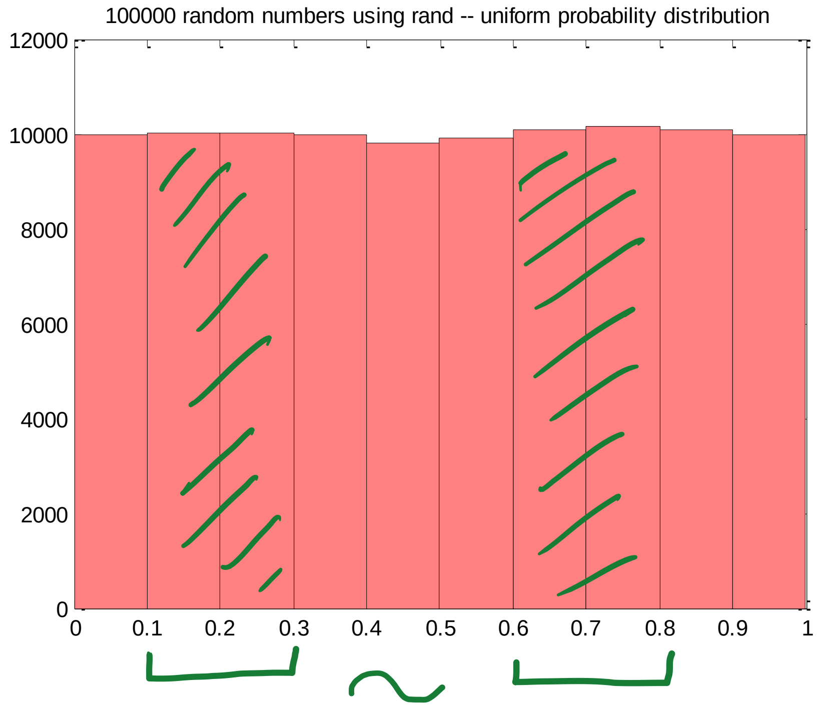
Uniform probability

Fair dice

- Equally likely to be 1 as to be 6
  - or 2, or 3, or 4, or 5

`rand()`

- Equally likely to be in (0,½) as to be in (½,1)
  - Equally likely to be in any two intervals of the same width (down to ~1e-15)
  - In particular, equally likely to be in (0,1/6) as in (5/6,1)

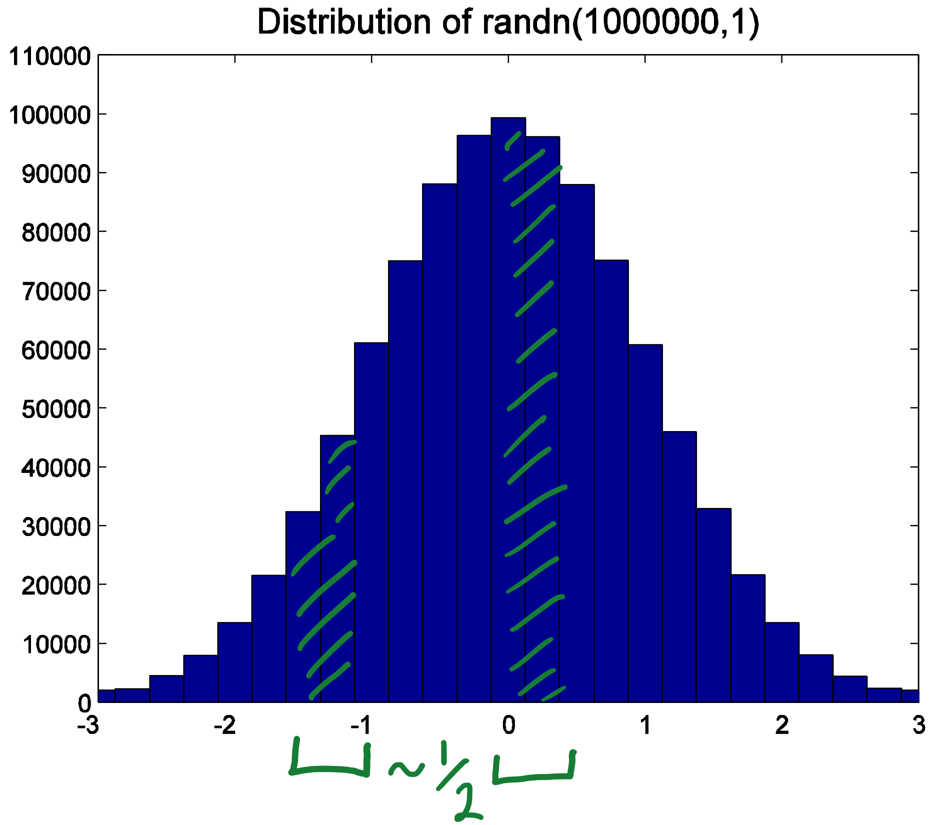


Uniform probability distribution in (0,1)

`rand()`

“Normal” distribution with zero mean and unit standard deviation

`randn()`



Mapping ranges to outcomes

Option 1: If-else

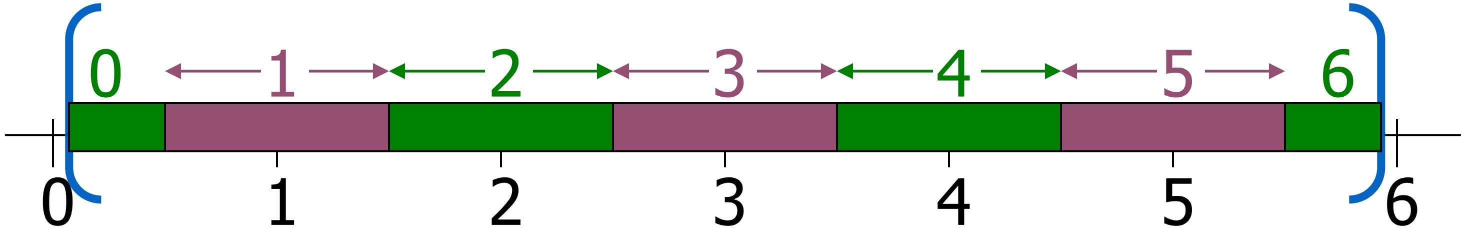
- Tedious to write
- What if number of outcomes (sides on die) changes?

Option 2: Scale and round

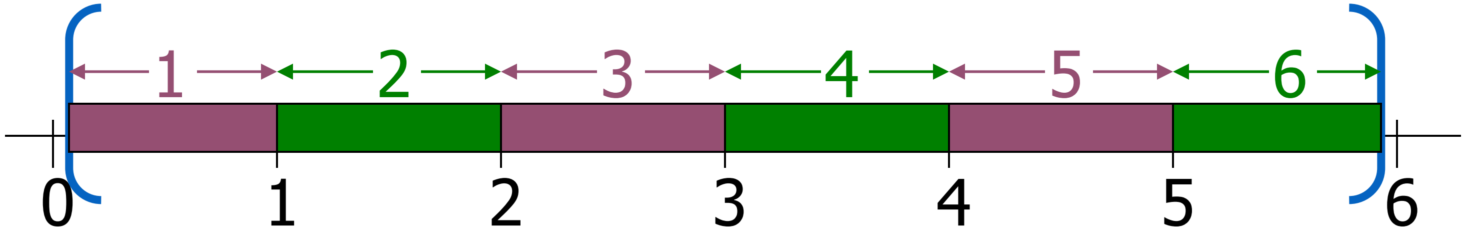
- Multiply so each outcome's range has width 1
- Round to integer
- (shift if necessary)

```
ceil(6*rand())
```

(prefer `floor()+1` for languages other than MATLAB) round(rand()*6)



ceil(rand()*6)



```
function counts = rollDie(rolls)

FACES= 6;                      % #faces on die
counts= 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
bar(1:FACES, counts)
```

Choosing bins based on outcome

Option 1: if-else

- Tedious to write
- What if number of outcomes (sides on die) changes?

Option 2: Direct indexing

- If indices are integers from 1 to N, and outcomes are integers from 1 to N, use outcome *as* index

```
function counts = rollDie(rolls)

FACES= 6;               % #faces on die
counts= 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
    counts(face)= counts(face) + 1;
end

% Show histogram of outcome
bar(1:FACES, counts)
```

More plotting

Figure management

- `title('Title of figure')`
- `xlabel('Label for x-axis')  % also ylabel`

- `figure       % open a new figure window`
- `close all    % close all figure windows`
- `shg          % show current figure window`
- `hold on      % plot on top of current figure contents`
- `hold off     % subsequent plots replace figure contents (default)`

- `axis off     % hide axes; to show (default), use on`
- `axis equal   % x, y tics are same size`

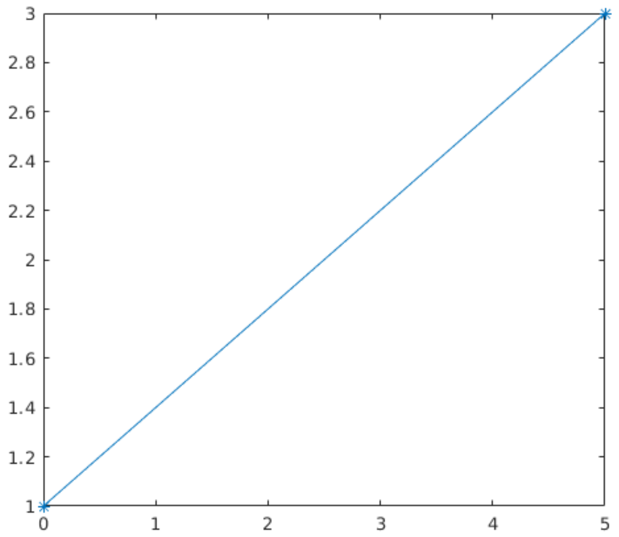
Start with drawing a single line segment

```
a= 0;  % x-coord of pt 1
b= 1;  % y-coord of pt 1
c= 5;  % x-coord of pt 2
d= 3;  % y-coord of pt 2
plot([a c], [b d], '-*')
```

x-values
(a vector)

y-values
(a vector)

Line/marker
format



Colors: r, g, b, m
Line types: -, :
Symbols: ., o, *

Default: auto-colored line

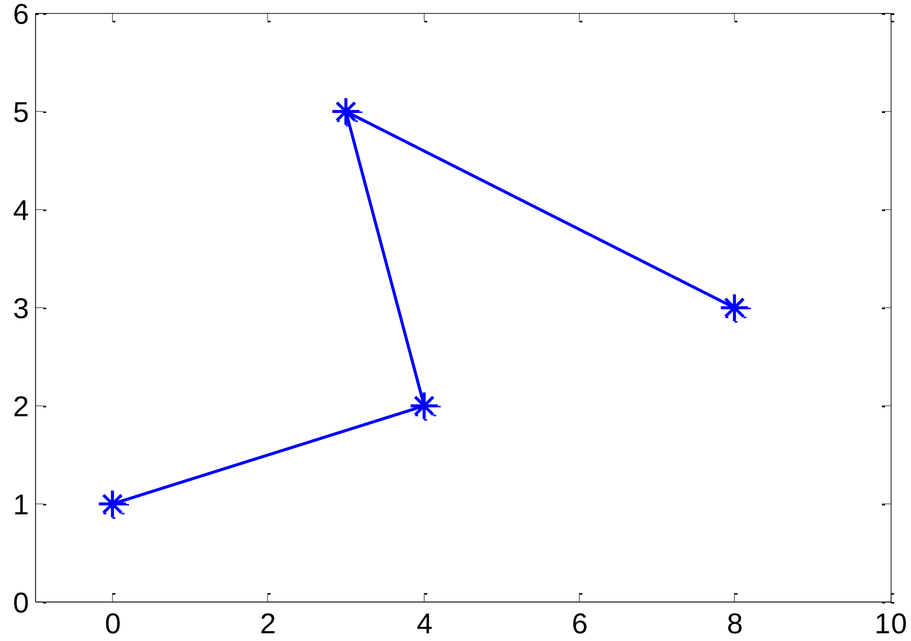
Making an x-y plot

```
a= [0 4 3 8];  % x-coords
b= [1 2 5 3];  % y-coords
plot(a, b, '-*')
```

x-values
(a vector)

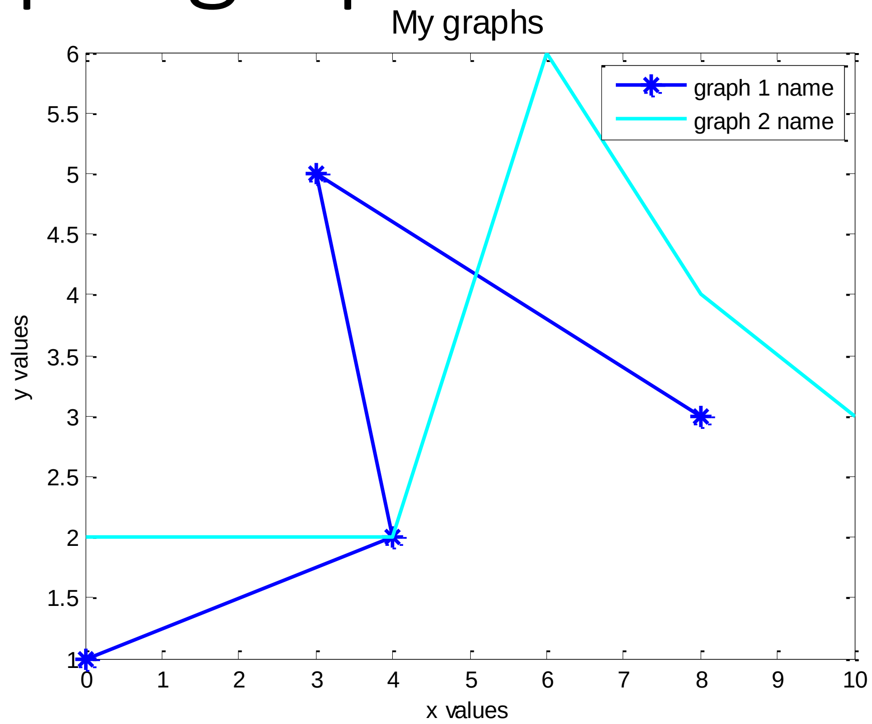
y-values
(a vector)

Line/marker
format



Making an x-y plot with multiple graphs (lines)

```
a= [0 4 5 8];
b= [1 2 5 3];
f= [0 4 6 8 10];
g= [2 2 6 4  3];
plot(a,b,'-*',f,g,'c')
legend('graph 1 name', 'graph 2 name')
xlabel('x values')
ylabel('y values')
title('My graphs', 'Fontsize',14)
```



See also showMultigraph, plotComparison2.m

Initialize vectors/matrices if dimensions are known

…instead of “building” the array one component at a time

```
% Initialize y
x=linspace(a,b,n);
y=zeros(1,n);
for k=1:n
   y(k)=myF(x(k));
end
```

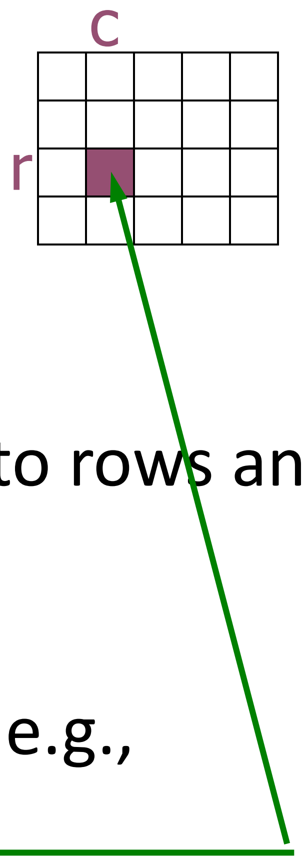
```
% Build y on the fly
x=linspace(a,b,n);

for k=1:n
   y(k)=myF(x(k));
end
```

Much faster for large n!

2D arrays

2-d array: **matrix**



- An array is a named collection of like data organized into rows and columns
- A 2-d array is a table, called a *matrix*
- Two *indices* identify the position of a value in a matrix, e.g.,

  `mat(r,c)`

  refers to component in row r, column c of matrix mat
- Array index starts at 1
- Rectangular: all rows have the same #of columns

Creating a matrix

- Built-in functions: `ones()`, `zeros()`, `rand()`
  - E.g., `zeros(2,3)` gives a 2-by-3 matrix of 0s
- “Build” a matrix using square brackets, `[ ]`, but the dimension must match up:
  - `[x  y]` puts `y` to the right of `x`
  - `[x; y]` puts `y` below `x`
  - `[4 0 3; 5 1 9]` creates the matrix
  - `[4 0 3; ones(1,3)]` gives
  - `[4 0 3; ones(3,1)]` doesn’t work

4	0	3
5	1	9

4	0	3
1	1	1

Working with a matrix: `size()` and individual components

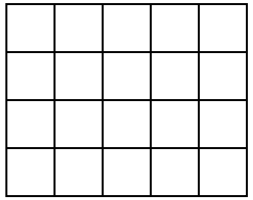
Given a matrix M,

2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

```
[nr, nc]= size(M)  % nr is #of rows,
                   % nc is #of columns
nr= size(M, 1)  % # of rows
nc= size(M, 2)  % # of columns

M(2,4)= 1;
disp(M(3,1))
M(1,nc)= 4;
```

Traverse a matrix using nested loops



```
function printMatrix(M)
% Print the values in matrix M
```

printMatrix.m

Pattern for traversing a matrix ("row-major")

```
[nr, nc] = size(M);
for r = 1:nr
   % At row r
   for c = 1:nc
      % At column c (in row r)
      % Do something with M(r,c) ...
   end
   % Optional end-of-row action
end
```