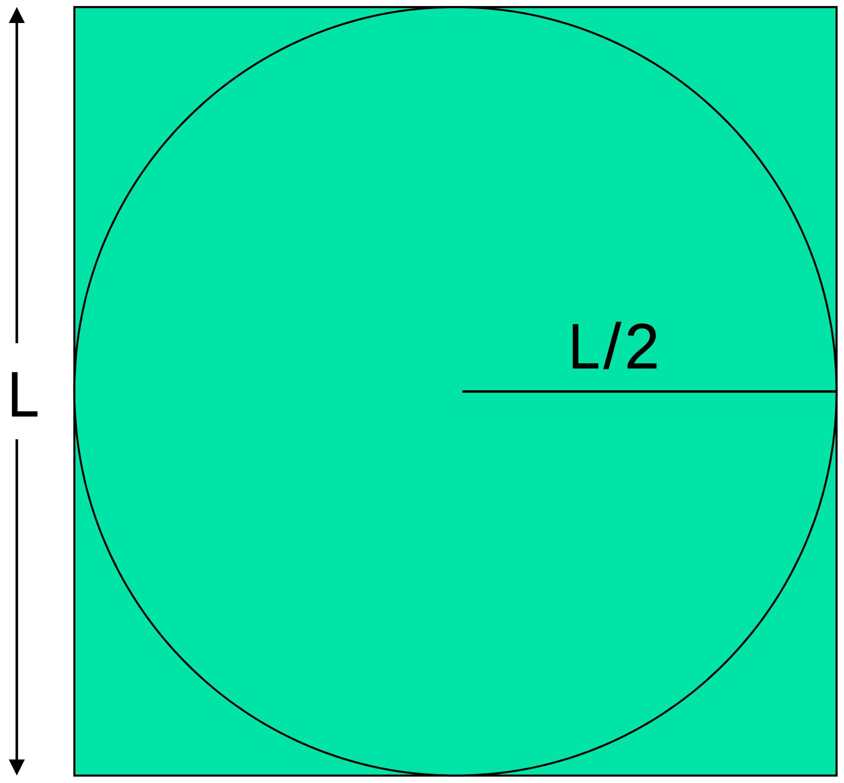
- Today's topics
  - Loops
  - Conditionals
  - More on user-defined function
  - 1-d array
- Announcement/Reminder:
  - Assignment 1a is due Feb 5th at 11:59pm

Monte Carlo Approximation of π



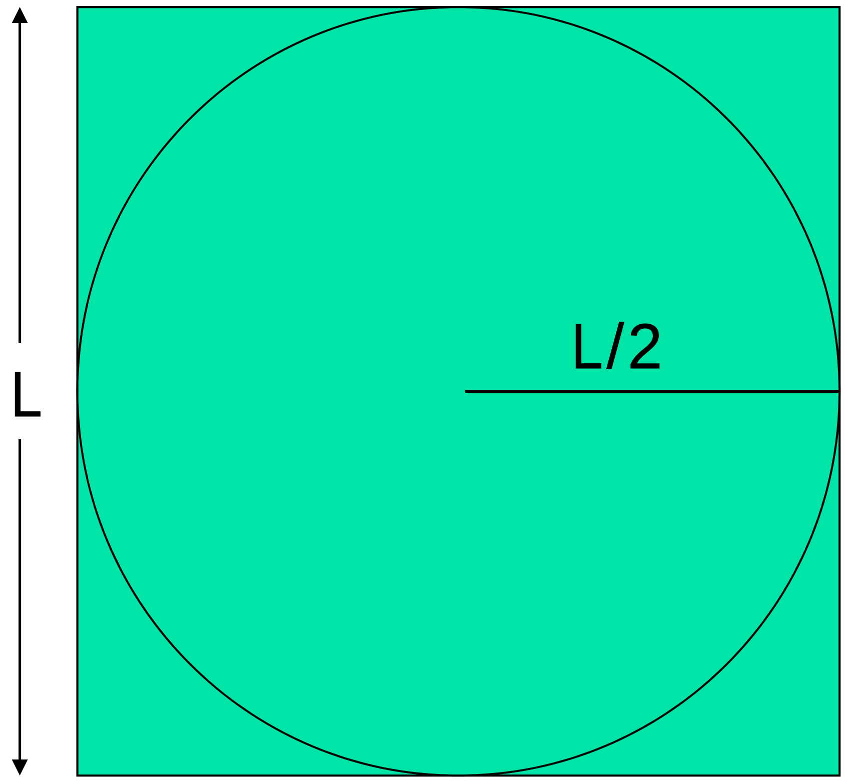
Throw N darts

Sq. area = $\boldsymbol{N} = \boldsymbol{L} \times \boldsymbol{L}$

Circle area = $\boldsymbol{N_{in}}$

$= \pi \boldsymbol{L^2 / 4}$

Monte Carlo Approximation of π



Throw N darts

Sq. area = $\boldsymbol{N} = \boldsymbol{L} \times \boldsymbol{L}$

Circle area = $\boldsymbol{N_{in}}$

$= \pi \boldsymbol{L^2 / 4}$

$\pi = \boldsymbol{4\ N_{in} / N}$

Monte Carlo Approximation of π

```
For each of N trials
  Throw a dart
  If it lands in circle
    add  1  to total # of hits

Pi  is  4*hits/N
```

Monte Carlo π with N darts on L-by-L board

```
N= ____;
for k = 1:N







end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ____;
for k = 1:N
   % Throw kth dart



   % Count it if it is in the circle




end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ___;  L= ___;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L – L/2;
   y = rand(1)*L – L/2;
   % Count it if it is in the circle



end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ___;  L= ___;  hits= 0;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L – L/2;
   y = rand(1)*L – L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
end
myPi = 4*hits/N;
```

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.

`for` loop examples

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

k takes on the values ____________

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

k takes on the values ____________

Default increment is 1

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

k takes on the values ____________

“Increment” may be negative

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

k takes on the values ____________

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
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

The **if** construct

```
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

Monte Carlo π with N darts on L-by-L board

```
N= ___;  L= ___;  hits= 0;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L – L/2;
   y = rand(1)*L – L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
end
myPi = 4*hits/N;
```

Using a while-loop

```
N= ___;  L= ___;  hits= 0;  k= 1;
while k <= N
   % Throw kth dart
   x = rand(1)*L – L/2;
   y = rand(1)*L – L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
   k = k+1;
end
myPi = 4*hits/N;
```

Common loop patterns

Do something *n* times

```
for k= 1:1:n
    % Do something

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

General form of a user-defined function

```
function [out1, out2, …]= functionName (in1, in2, …)
% 1-line comment to describe the function
% Additional description of function

    Executable code that at some point assigns
    values to output parameters out1, out2, …
```

- *in1*, *in2*, … are defined when the function begins execution. Variables *in1*, *in2*, … are called function *parameters* and they hold the function *arguments* used when the function is invoked (called).
- *out1*, *out2*, … are not defined until the executable code in the function assigns values to them.

```
function myPi = mcPiFun(N)
% myPi is Monte Carlo estimate of pi by
% throwing N darts

~~N= ___;~~  L= ___;  hits= 0;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L – L/2;
   y = rand(1)*L – L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
end
myPi = 4*hits/N;
```

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

rads= theta*pi/180;  % radian
x= r*cos(rads);
y= r*sin(rads);
```

A function file **polar2xy.m**

Function header is the "contract" for how the function will be used (called)

You have this function:

```
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:

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

Given this function:

```
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).

   . . .
```

How many proper calls to convertLength are shown below?

```
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1   B: 2   C: 3   D: 4   E: 5 or 0

Comments in functions

- Block of comments after the function header is printed whenever a user types

  ```
  help <functionName>
  ```

  at the Command Window
- 1st line of this comment block is searched whenever a user types

  ```
  lookfor <someWord>
  ```

  at the Command Window
- Every function should have a comment block after the function header that says what the function does concisely

Accessing a function

- A function is accessible if it is in the current directory or if it is on the search path
- Easy: put all related m-files in the same directory
- Better: the `path` function gives greater flexibility

Arrays

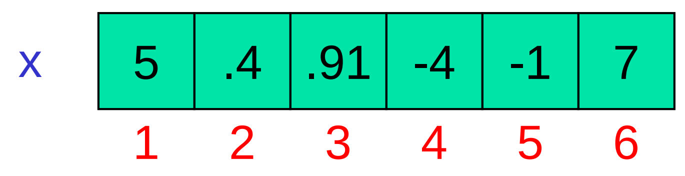
The basic variable in Matlab is a matrix:

- Scalar → 1 x 1 matrix

- 1-d array of length 4 →
  1 x 4 matrix   or   4 x 1 matrix

- 2-d array → a matrix, naturally

Array index starts at 1



Let k be the index of vector x, then

- k must be a positive integer
- 1<= k <= length(x)
- To access the k^{th} element: x(k)

Here are a few different ways to create a vector

```
count= zeros(1,6)
```

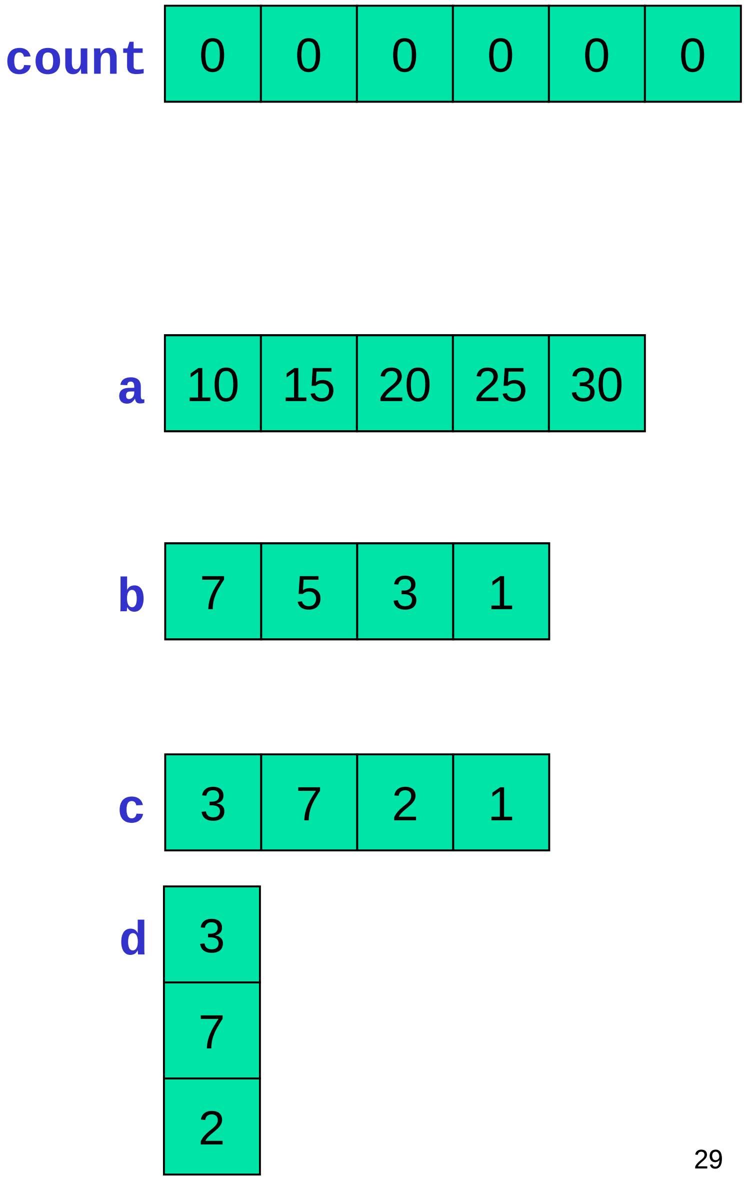
Similar functions: `ones`, `rand`

```
a= linspace(10,30,5)
```

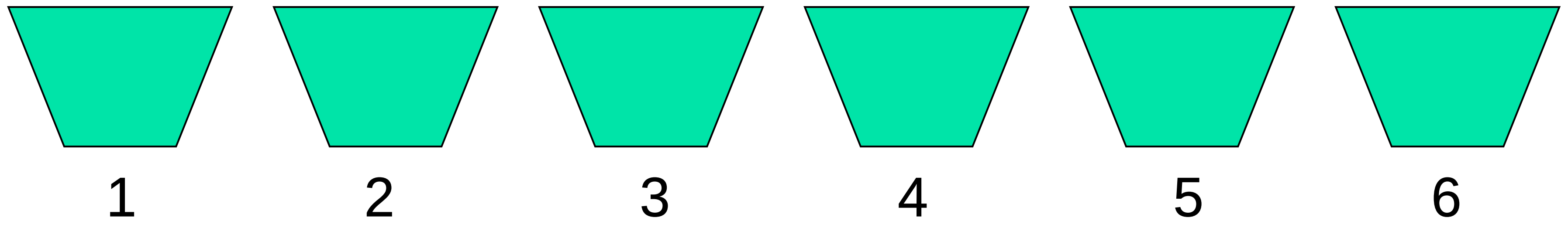
```
b= 7:-2:0
```

```
c= [3 7 2 1]
```

```
d= [3; 7; 2]
```



Possible outcomes from rolling a fair 6-sided die



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
```

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!