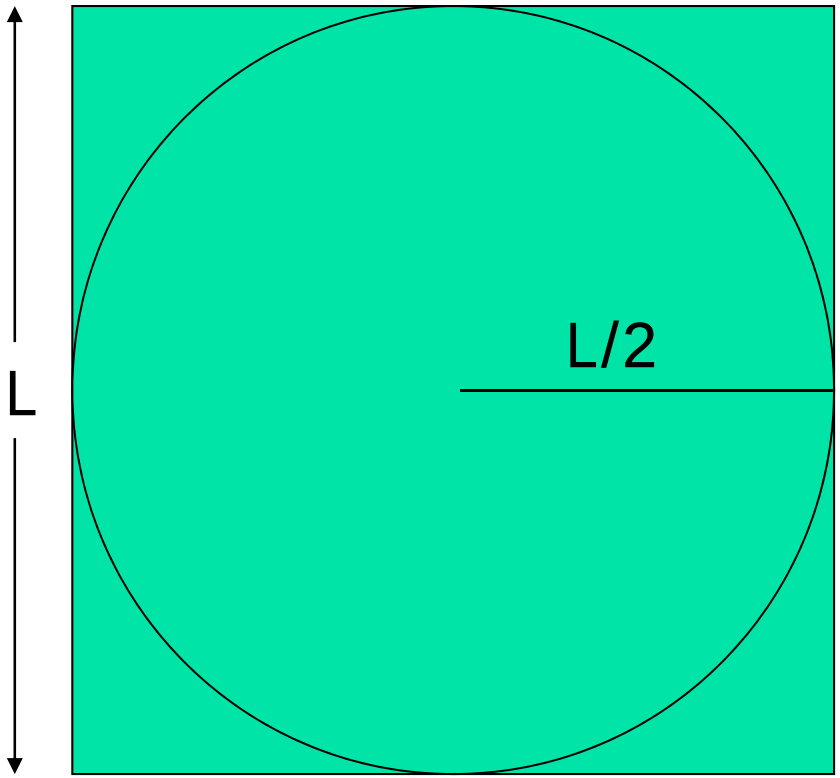
- Today's Lecture:
  - Vectorized computation
  - Introduction to graphics

- Announcements:
  - Assignment 1a: due Tues at 11:59pm, at which time submission on CMS will close. Will re-open for re-submission later.
  - Assignment 1b: due Tues Feb12 at 11:59pm

Monte Carlo Approximation of π



Throw N darts

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

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

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

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

1. Make a plot
2. Use vectors to store all values
3. Use vectorized arithmetic

Vectorized addition

	x	2	1	.5	8
+	y	1	2	0	1
=	z	3	3	.5	9

Matlab code: `z= x + y`

Vectorized subtraction

	x	2	1	.5	8
-	y	1	2	0	1
=	z	1	-1	.5	7

Matlab code: `z= x - y`

Vectorized code

—a Matlab-specific feature

See Sec 4.1 for list of vectorized arithmetic operations

- Code that performs element-by-element arithmetic/relational/logical operations on array operands in one step

- Scalar operation: x + y

  where x, y are scalar variables

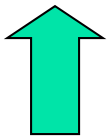
- Vectorized code: x + y

  where x and/or y are vectors. If x and y are both vectors, they must be of the same shape and length

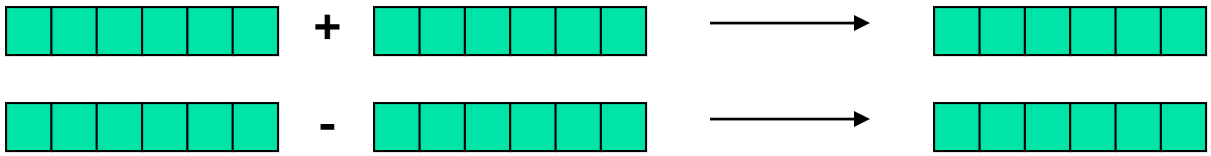
Vectorized multiplication

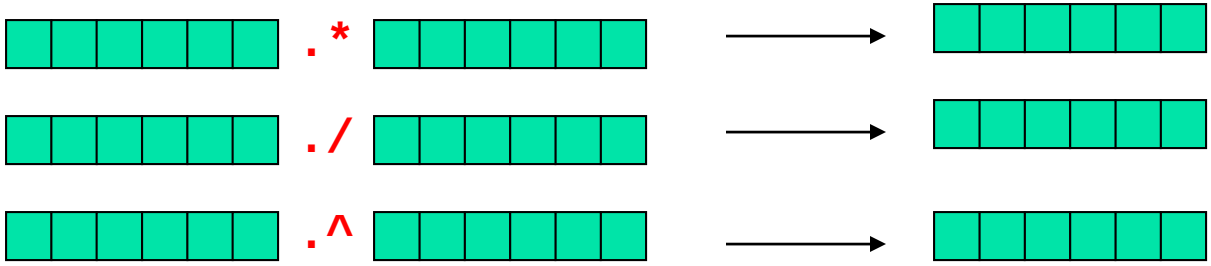
	a	2	1	.5	8
×	**b**	1	2	0	1
=	**c**	2	2	0	8

Matlab code: **c= a .* b**



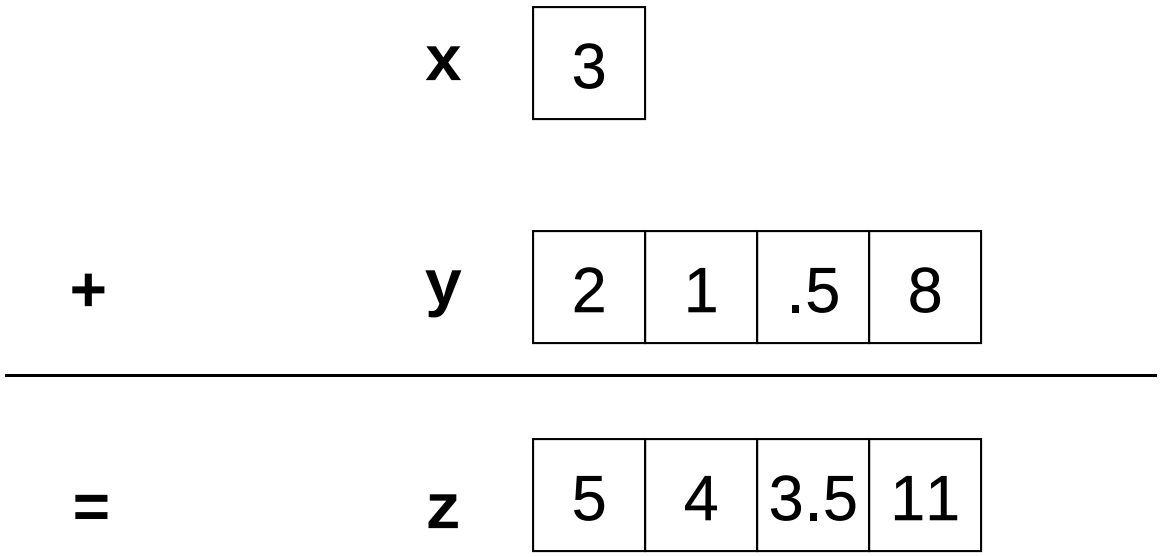
Vectorized
element-by-element arithmetic operations on arrays





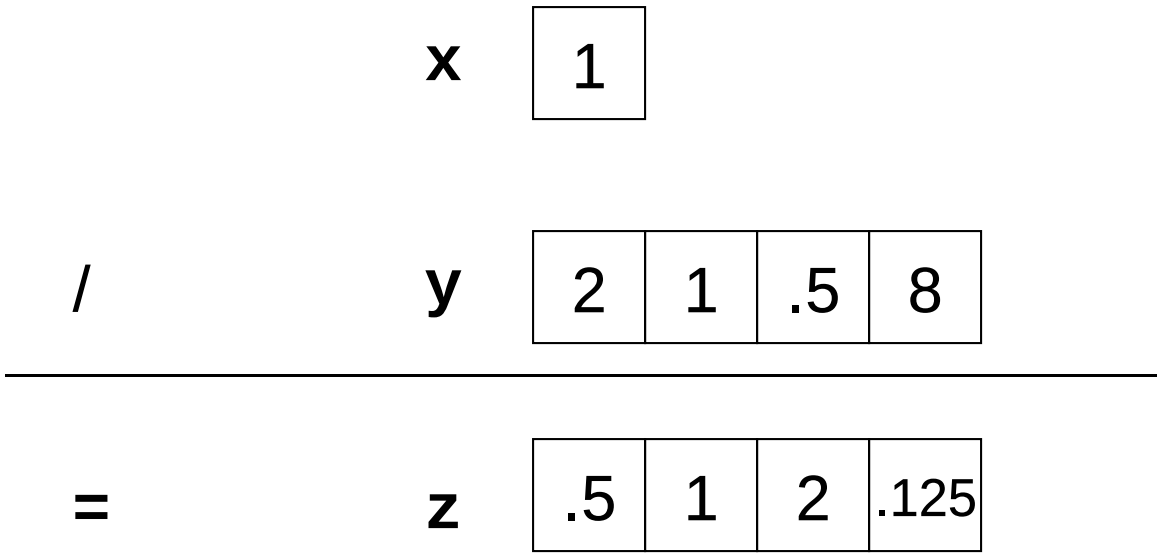
A dot (.) is necessary in front of these math operators

Shift



Matlab code: `z= x + y`

Reciprocate

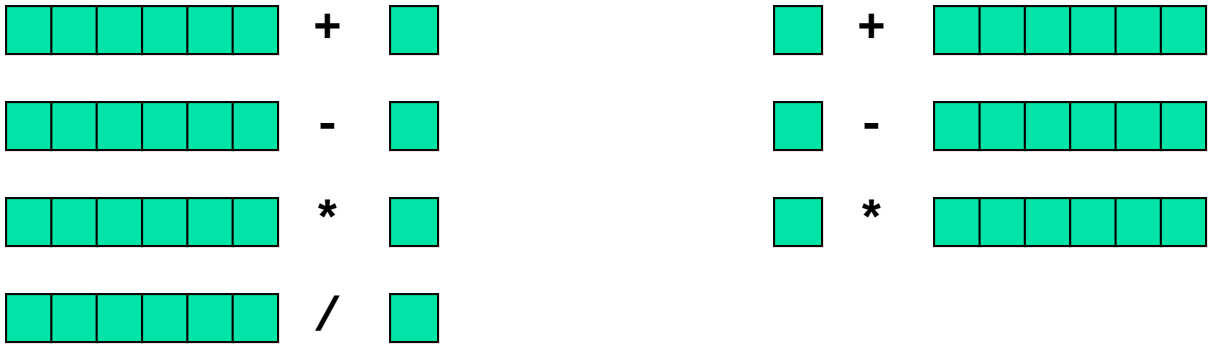


Matlab code: `z= x ./ y`

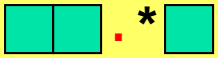
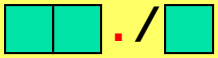
See full list of ops in §4.1

Vectorized

element-by-element arithmetic operations between an array and a scalar



A dot (.) is necessary in front of these math operators

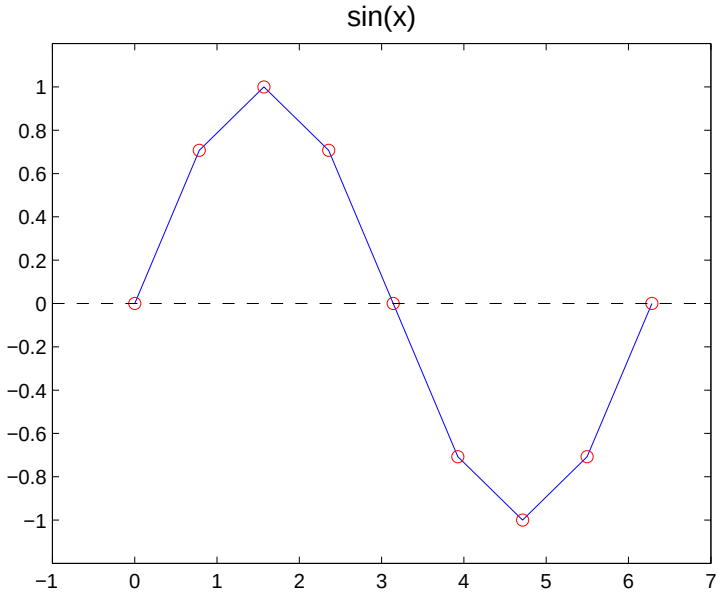
The dot in  ,  ,  not necessary but OK

Generating tables and plots

x	sin(x)
0.000	0.000
0.784	0.707
1.571	1.000
2.357	0.707
3.142	0.000
3.927	-0.707
4.712	-1.000
5.498	-0.707
6.283	0.000

`x`, `y` are vectors. A vector is a 1-dimensional list of values

```
x= linspace(0,2*pi,9);
y= sin(x);
plot(x,y)
```



Note: x, y are shown in columns due to space limitation; they should be rows.

Does this assign to y the values
sin(0°), sin(1°), sin(2°), …, sin(90°)?

```
x = linspace(0,pi/2,90);

y = sin(x);
```

A: yes B: no

Plot this!

See **plotComparison.m**

$$f(x) = \frac{\sin(5x)\exp(-x/2)}{1+x^2}$$

for
-2 <= x <= 3

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
plot(x,y)
```

Element-by-element arithmetic operations on arrays

Element-by-element arithmetic operations on arrays…
Also called "vectorized code"

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
```

`x` and `y` are vectors

Contrast with scalar operations that we've used previously…

```
a = 2.1;
b = sin(5*a);
```

`a` and `b` are scalars

The operators are (mostly) the same; the operands may be scalars or vectors.

When an operand is a vector, you have "vectorized code."

Some format commands to use with **plot**

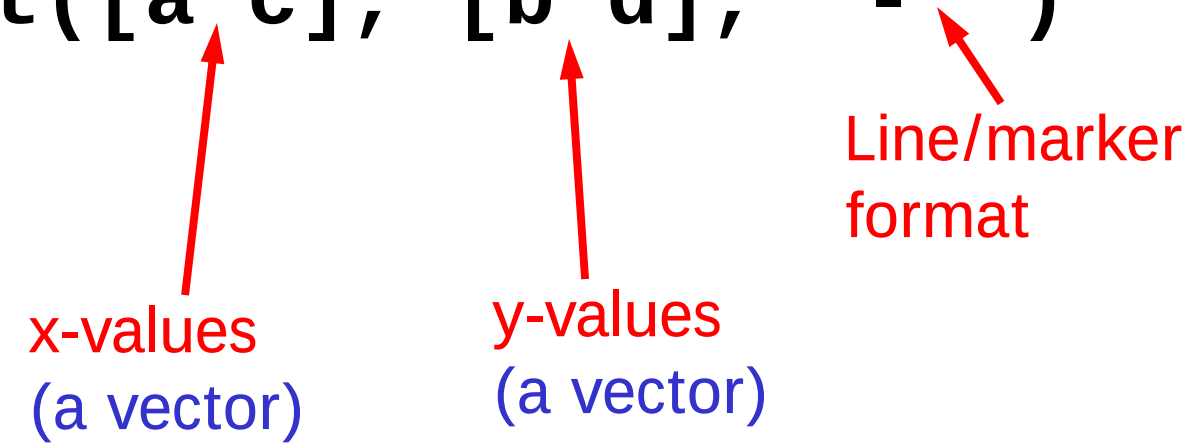
```
xlabel('text for labeling x-axis')
ylable('text for labeling y-axis')
title('text for plot title at top center')

hold on  % hold subsequent plot commands to current axes
hold off  % subsequent plot command refreshes axes--
          % erase previous items

close all   % close all graphics windows
axis equal  % same scaling for x, y axes
axis off    % hide axes
axis on     % show axes
```

default

default

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], '-*')
```



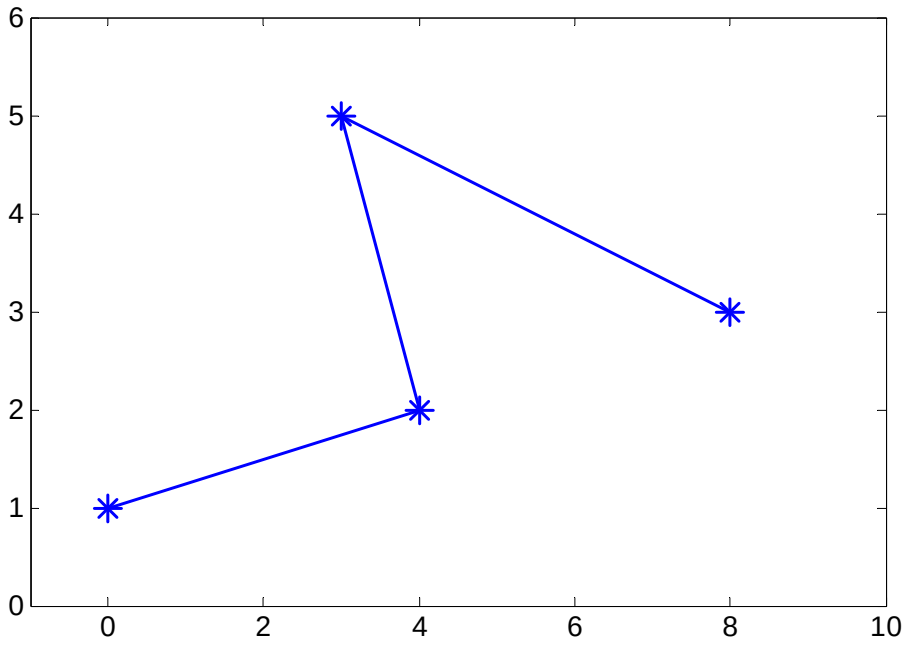
Making an x-y plot

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

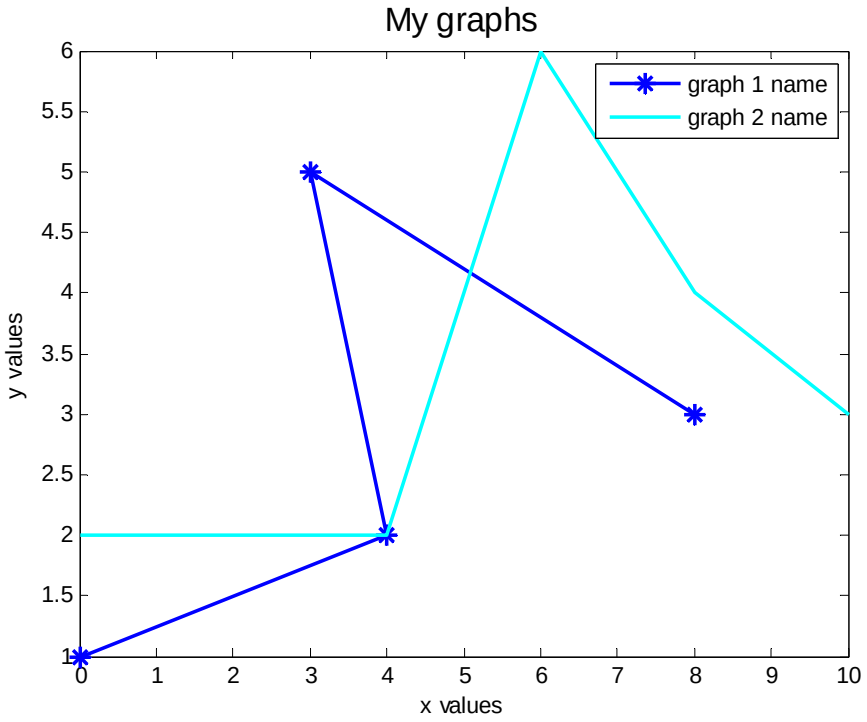
x-values
(a vector)

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 plotComparison.m

Drawing a polygon (multiple line segments)

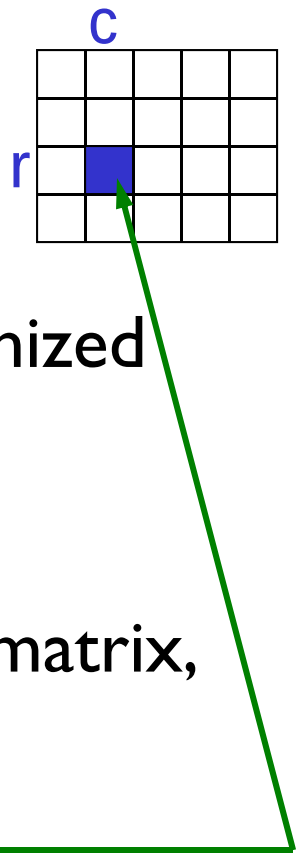
```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h.
x= [                              ];  % x data
y= [                              ];  % y data
plot(x, y)
```

Fill in the missing vector values!

Drawing a polygon (multiple line segments)

```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h.
x= [a  a+w  a+w  a    a ];  % x data
y= [b  b    b+h  b+h  b ];  % y data
plot(x, y)
```

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(1)
  - 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
5	1	9

  - [4 0 3; ones(1,3)] gives

4	0	3
1	1	1

  - [4 0 3; ones(3,1)] doesn't work