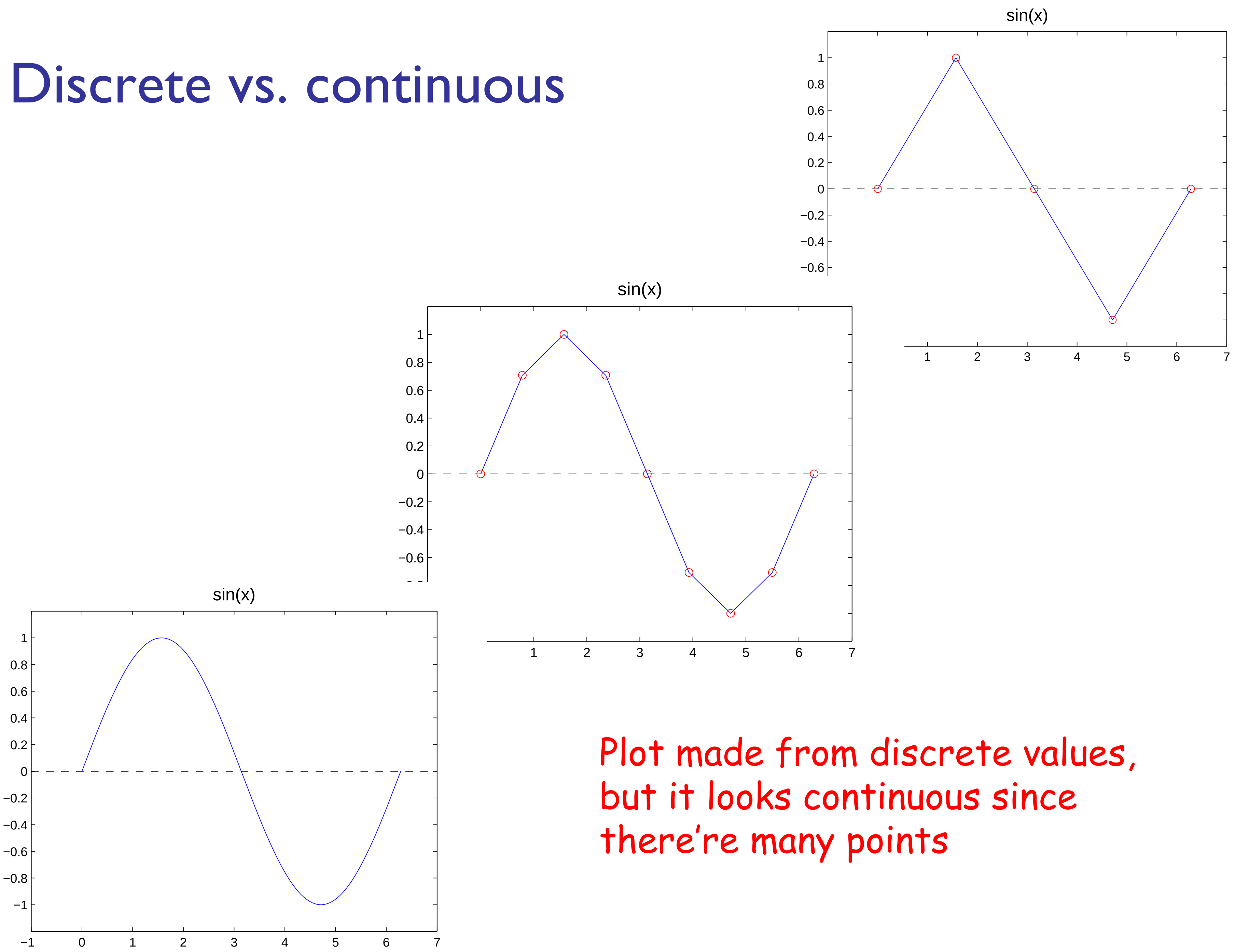
- Previous Lecture:
  - Review
  - Color as a 3-vector
  - Linear interpolation

- Today's Lecture:
  - Finite/inexact arithmetic
  - Plotting continuous functions using vectors and vectorized code
  - Introduction to user-defined functions

- Announcements:
  - Discussion this week in classrooms as listed on roster, not the lab
  - Prelim 1 on Thursday, Feb 24th at 7:30pm
    - Last names A-O in Statler Aud. main floor
    - Last names P-Z in Statler Aud. balcony

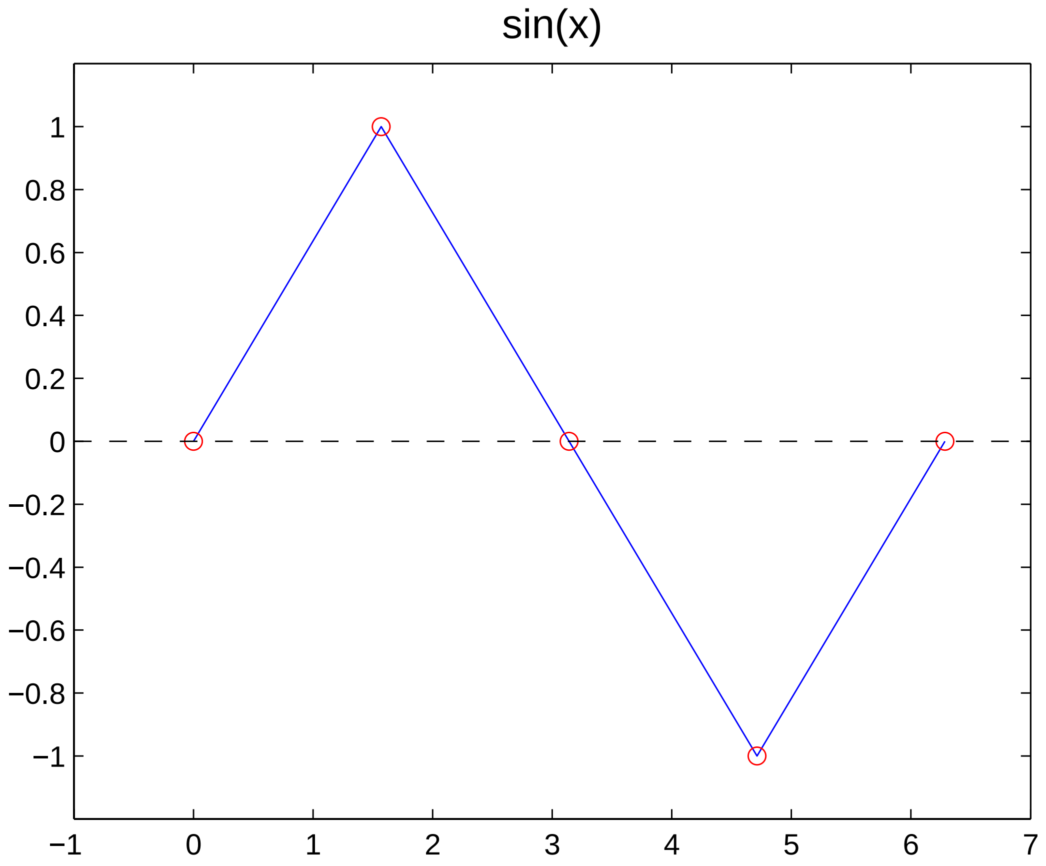
Discrete vs. continuous



Plot made from discrete values,
but it looks continuous since
there're many points

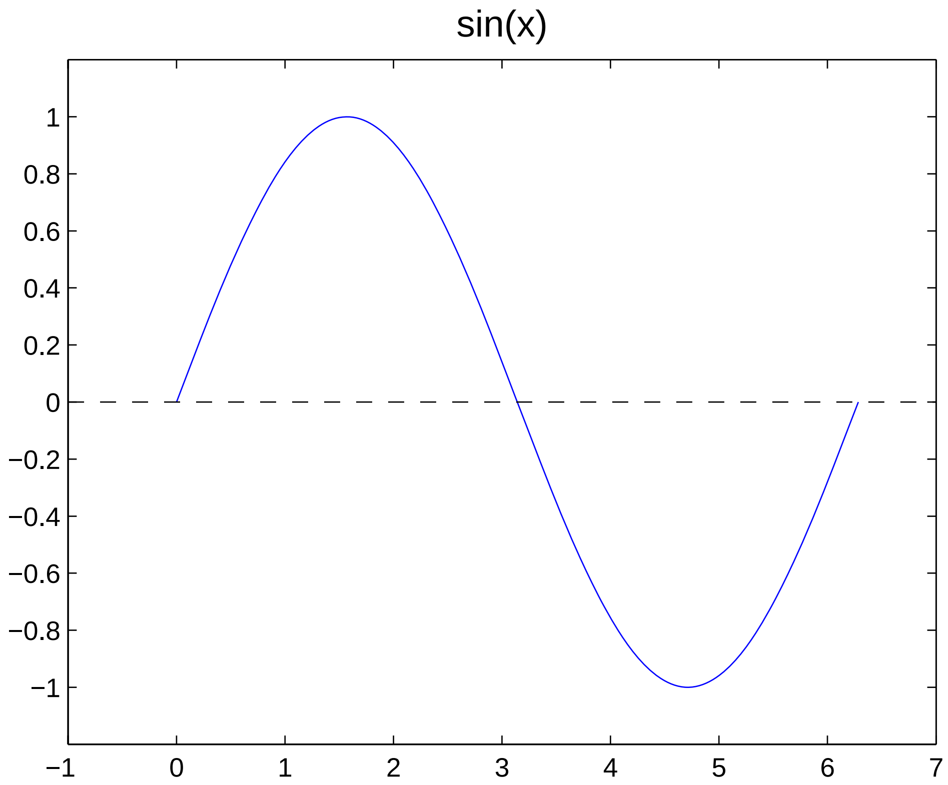
Plot a continuous function (from a table of values)

x	sin(x)
0.00	0.0
1.57	1.0
3.14	0.0
4.71	-1.0
6.28	0.0



Plot based on 5 points

Plot based on 200 discrete points, but it looks smooth

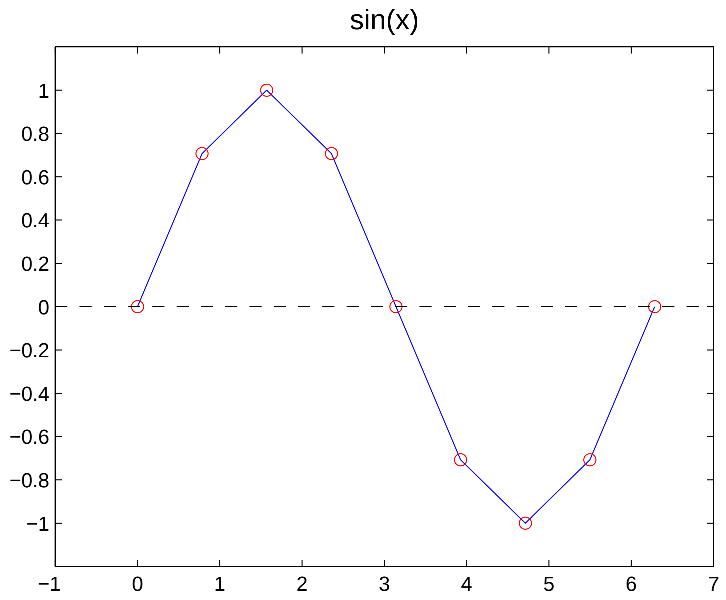


Generating tables and plots

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

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



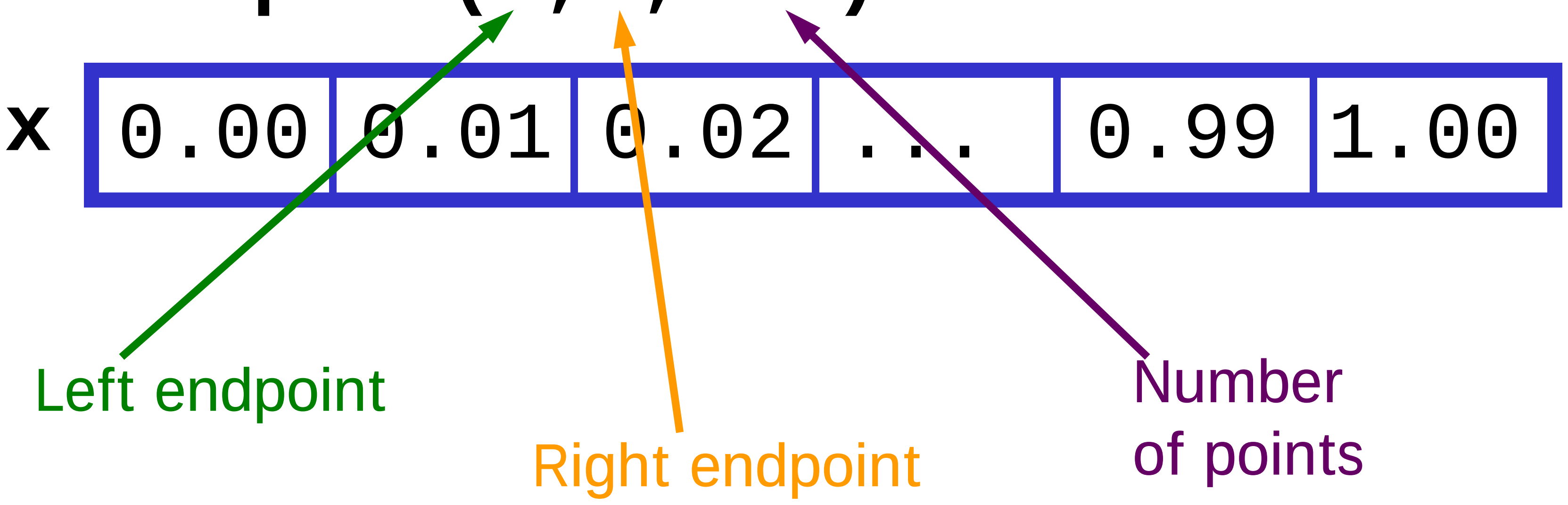
Note: x, y are shown in columns due to space limitation; they should be rows.

Built-in function **`linspace`**

```
x= linspace(1,3,5)
```

x	1.0	1.5	2.0	2.5	3.0

```
x= linspace(0,1,101)
```



x	0.00	0.01	0.02	...	0.99	1.00

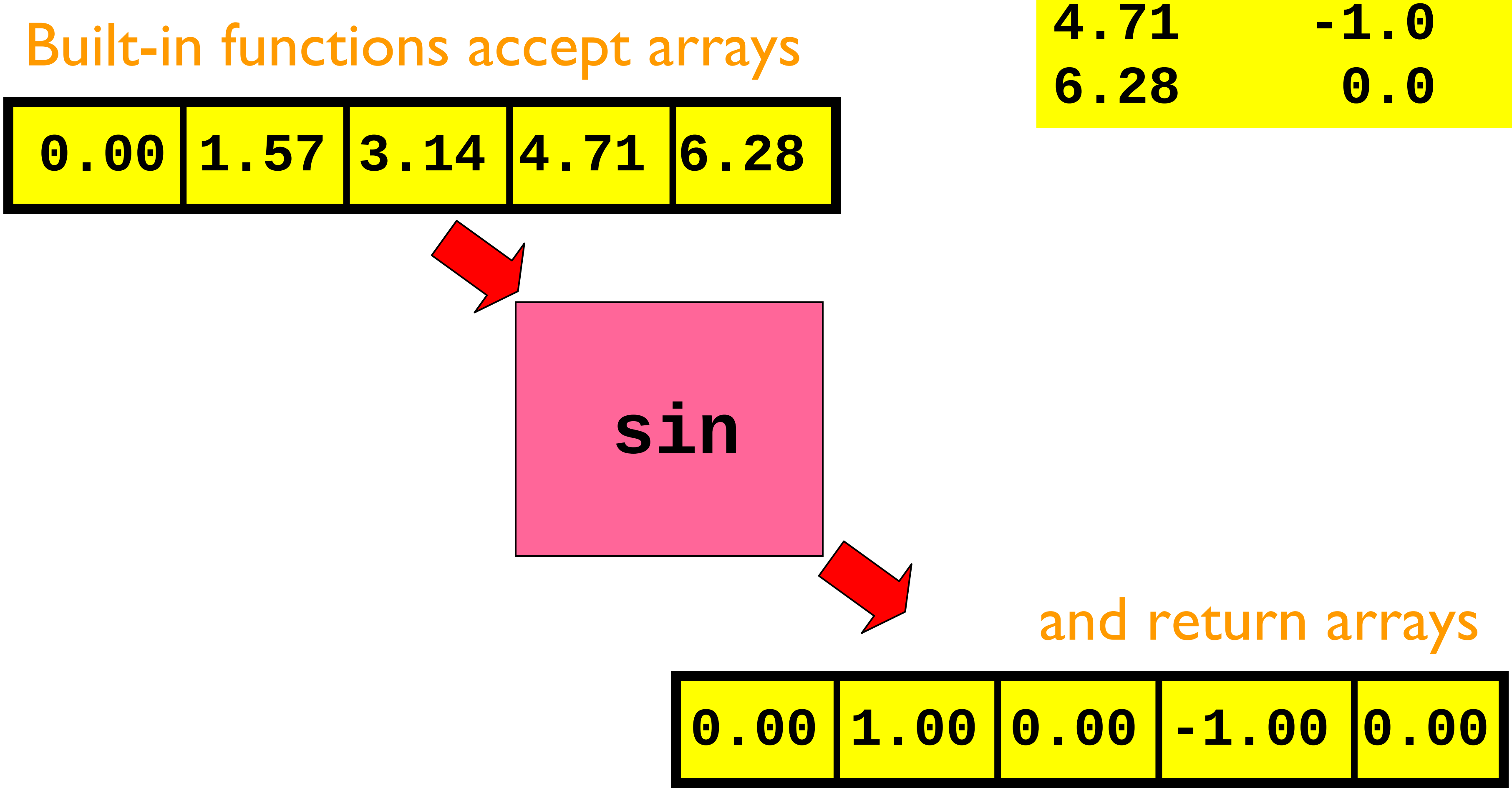
Left endpoint

Right endpoint

Number of points

How did we get all the sine values?

x	sin(x)
0.00	0.0
1.57	1.0
3.14	0.0
4.71	-1.0
6.28	0.0



Built-in functions accept arrays

0.00	1.57	3.14	4.71	6.28

and return arrays

0.00	1.00	0.00	-1.00	0.00

Examples of functions that can work with arrays

```
x= linspace(0,1,200);
y= exp(x);
plot(x,y)
```

```
x= linspace(1,10,200);
y= log(x);
plot(x,y)
```

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

Can we plot this?

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

for

-2 <= x <= 3

Can we plot this?

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

for
-2 <= x <= 3

Yes!

See `plotComparison.m`

Can we plot this?

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

for

-2 <= x <= 3

Yes!

```
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 and **y** are vectors

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

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.”

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

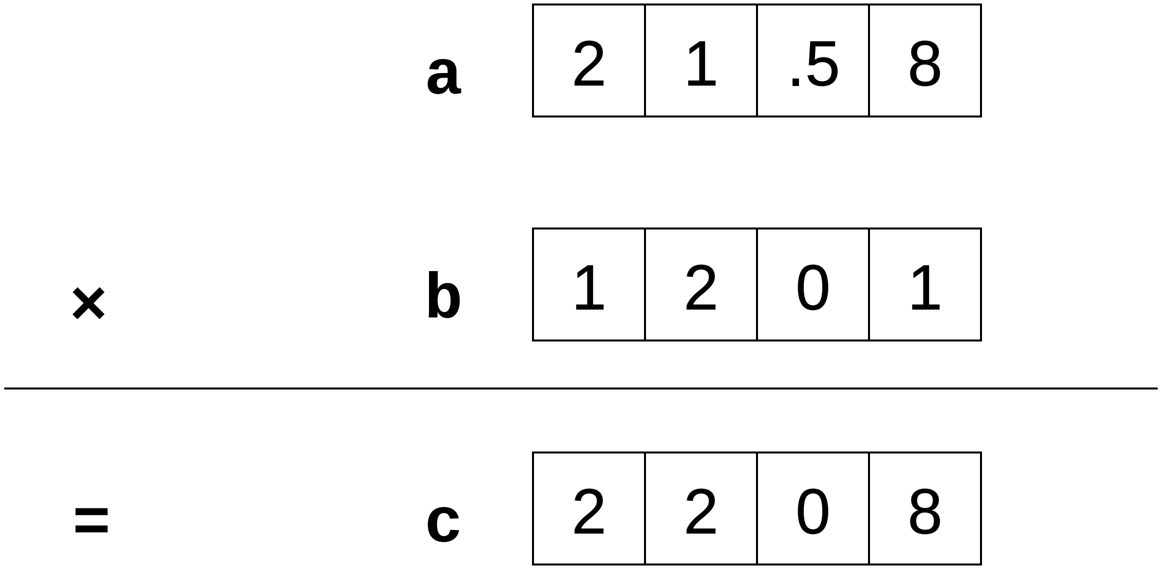
- Scalar operation: x + y

  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

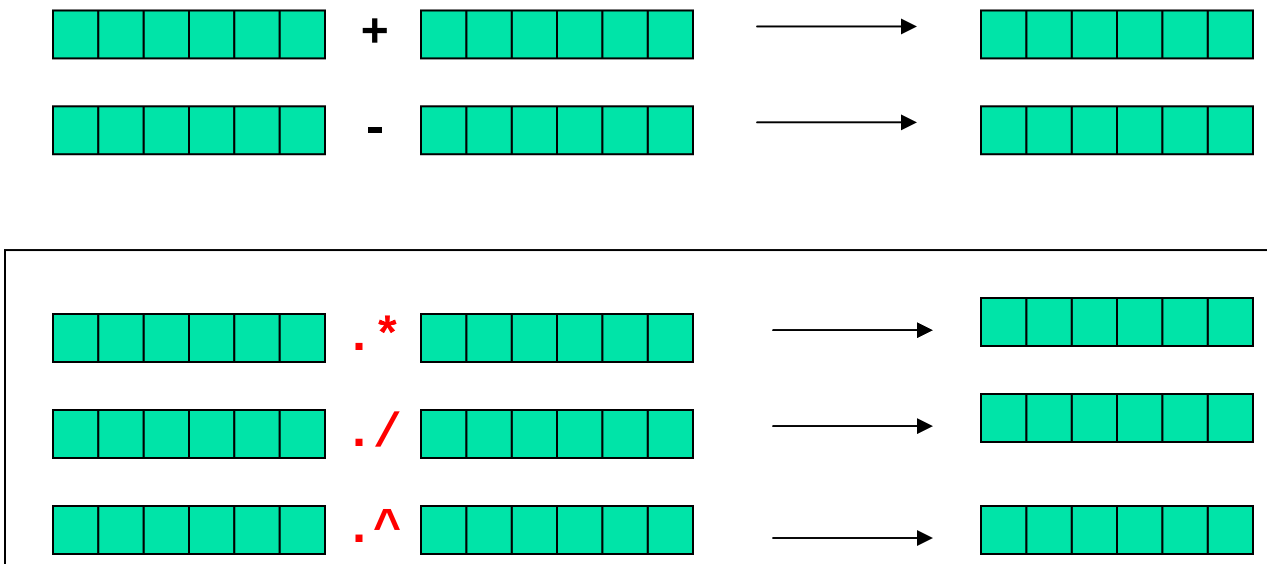


	a	2	1	.5	8
×	**b**	1	2	0	1
=	**c**	2	2	0	8

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

See full list of ops in §4.1

Vectorized code element-by-element arithmetic operations on arrays



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

Shift

	x	3			
+	y	2	1	.5	8
=	z	5	4	3.5	11

Matlab code: `z= x + y`

Reciprocate

	x	1			
/	**y**	2	1	.5	8
=	**z**	.5	1	2	.125

Matlab code: `z= x ./ y`

See full list of ops in §4.1

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

🟩🟩🟩🟩🟩🟩 + 🟩   🟩 + 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 - 🟩   🟩 - 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 * 🟩   🟩 * 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 / 🟩

🟩 ./ 🟩🟩🟩🟩🟩🟩

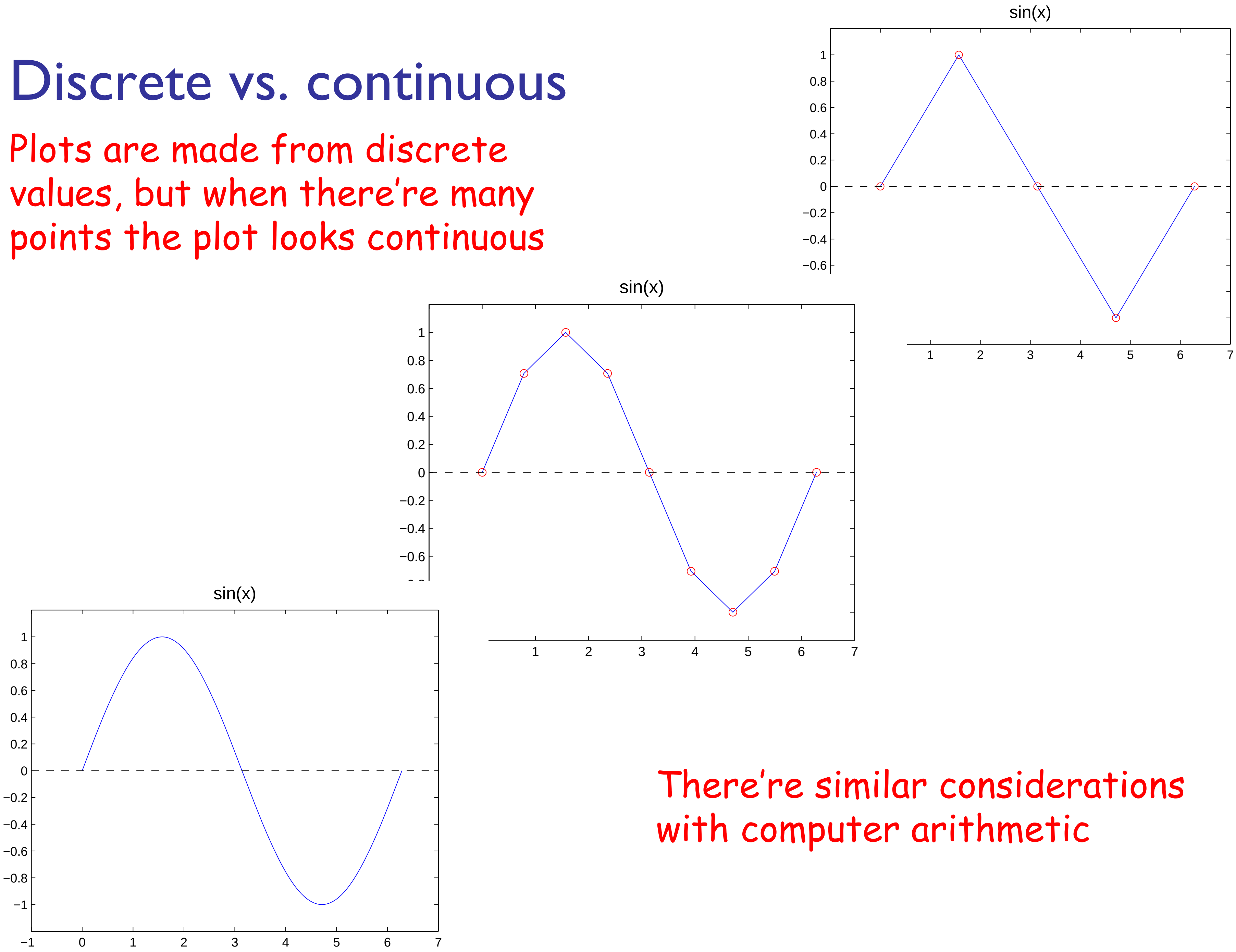
🟩🟩🟩🟩🟩🟩 .^ 🟩   🟩 .^ 🟩🟩🟩🟩🟩🟩

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

The dot in 🟩🟩.*🟩 , 🟩.*🟩🟩 , 🟩🟩./🟩 not necessary but OK

Discrete vs. continuous

Plots are made from discrete values, but when there're many points the plot looks continuous



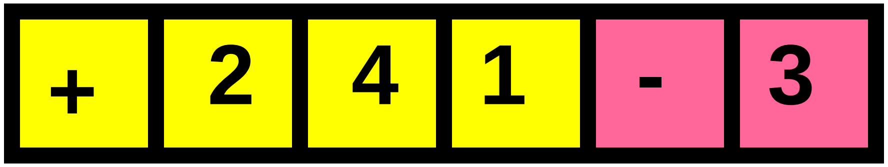
There're similar considerations with computer arithmetic

Does this script print anything?

```
k = 0;
while 1 + 1/2^k > 1
   k = k+1;
end
disp(k)
```

Computer Arithmetic—floating point arithmetic

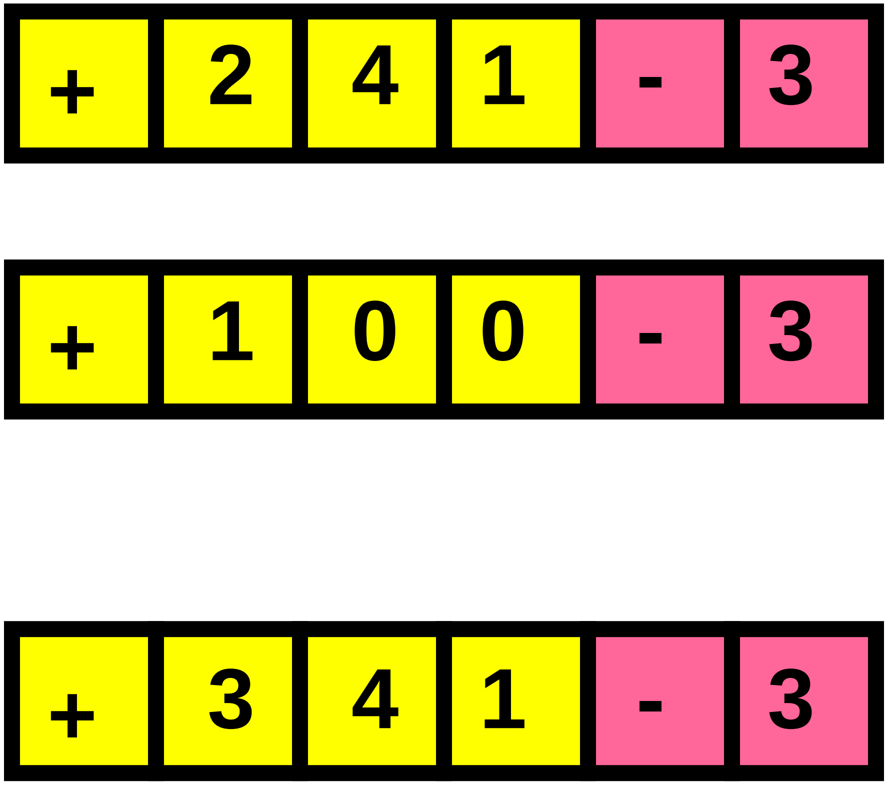
Suppose you have a calculator with
a window like this:



+	2	4	1	-	3

representing 2.41×10^{-3}

Floating point addition



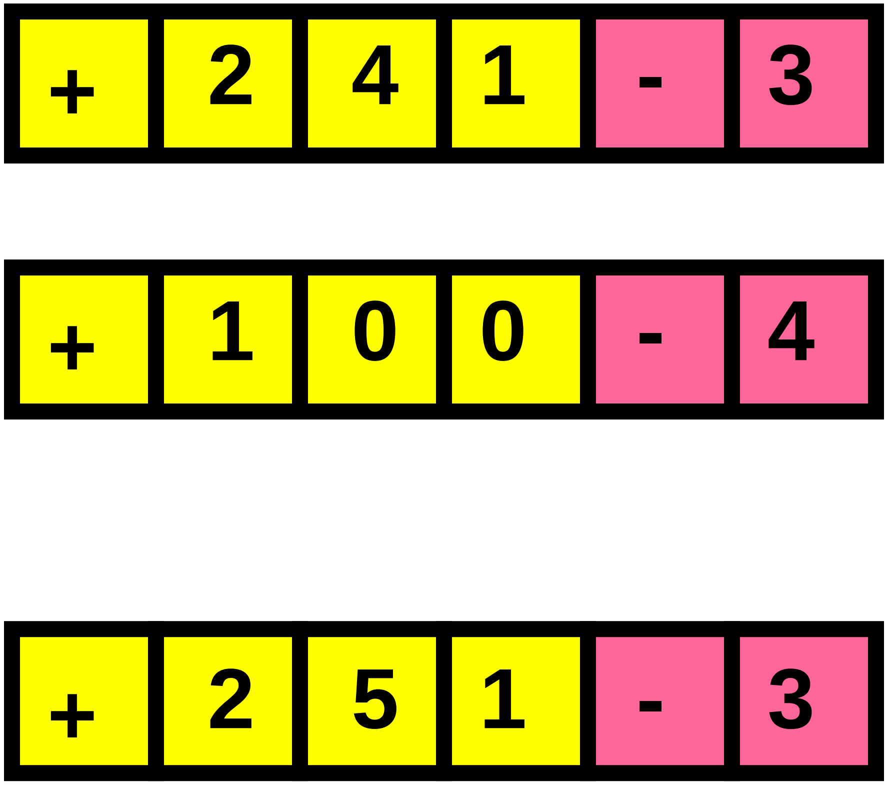
+	2	4	1	-	3

+	1	0	0	-	3

Result:

+	3	4	1	-	3

Floating point addition



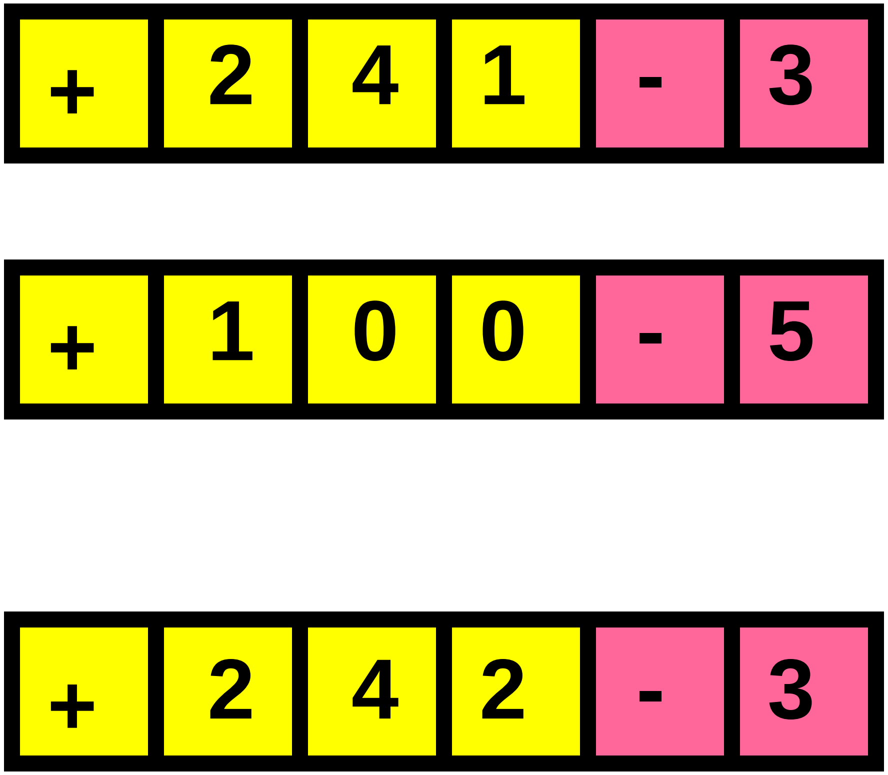
+	2	4	1	-	3

+	1	0	0	-	4

Result:

+	2	5	1	-	3

Floating point addition



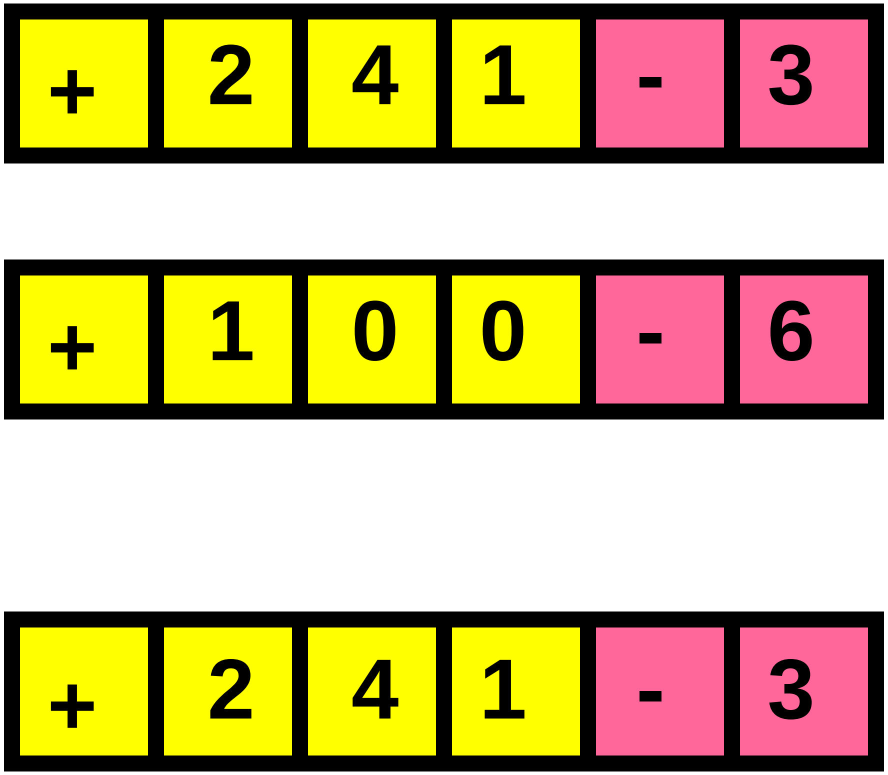
+	2	4	1	-	3

+	1	0	0	-	5

Result:

+	2	4	2	-	3

Floating point addition



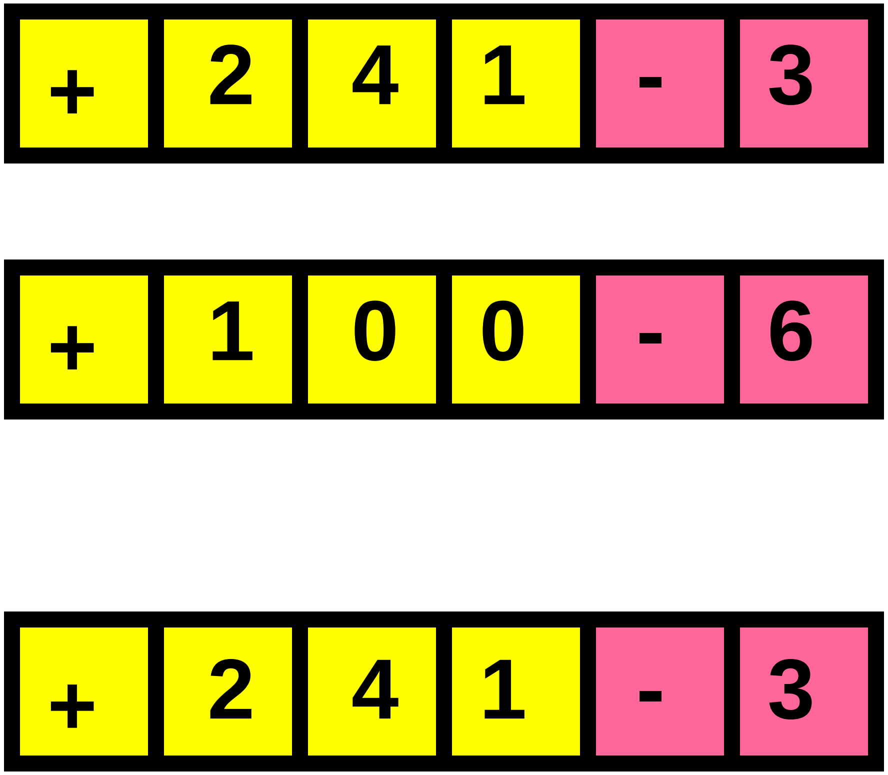
+	2	4	1	-	3

+	1	0	0	-	6

Result:

+	2	4	1	-	3

Floating point addition



+	2	4	1	-	3

+	1	0	0	-	6

Result:

+	2	4	1	-	3

Not enough room to represent .002411

The loop DOES terminate given the limitations of floating point arithmetic!

```
k = 0;
while 1 + 1/2^k > 1
   k = k+1;
end
disp(k)
```

1 + 1/2^53 is calculated to be just 1, so “53” is printed.

Patriot missile failure

www.namsa.nato.int/gallery/systems

In 1991, a Patriot Missile failed, resulting in 28 deaths and about 100 injured. The cause?

0.1

Inexact representation of time/number

- System clock represented time in tenths of a second: a clock tick every 1/10 of a second

- Time = number of clock ticks x 0.1

“exact” value

.00011001100110011001100110011…

.0001100110011001100110011

value in Patriot system

Error of .000000095 every clock tick

Resulting error

… after 100 hours

.000000095 x (100x60x60)

0.34 second

At a velocity of 1700 m/s, missed target by more than 500 meters!

Computer arithmetic is *inexact*

- There is error in computer arithmetic—floating point arithmetic—due to limitation in "hardware." Computer memory is finite.

- What is $1 + 10^{-16}$?
  - 1.0000000000000001 in real arithmetic
  - 1 in floating point arithmetic (IEEE)

- Read Sec 4.3

Built-in functions

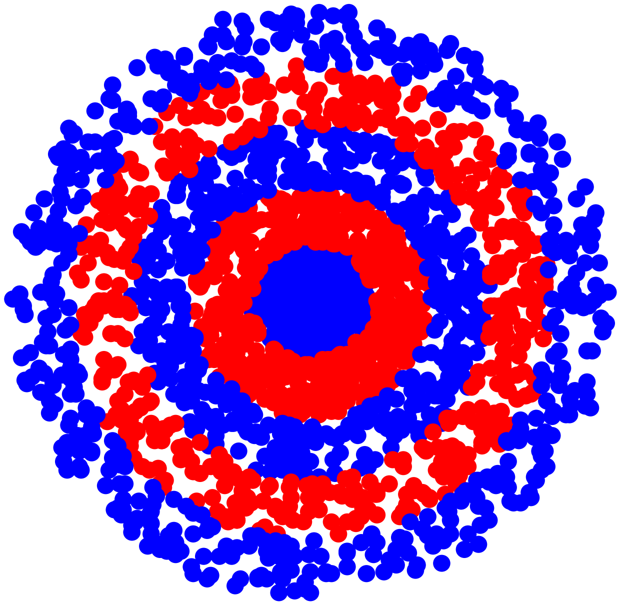
- We've used many Matlab built-in functions, e.g., **`rand`**, **`abs`**, **`floor`**, **`rem`**
- Example: **`abs(x-.5)`**
- Observations:
  - **`abs`** is set up to be able to work with any valid data
  - **`abs`** *doesn't prompt us for input; it expects that we provide data* that it'll then work on

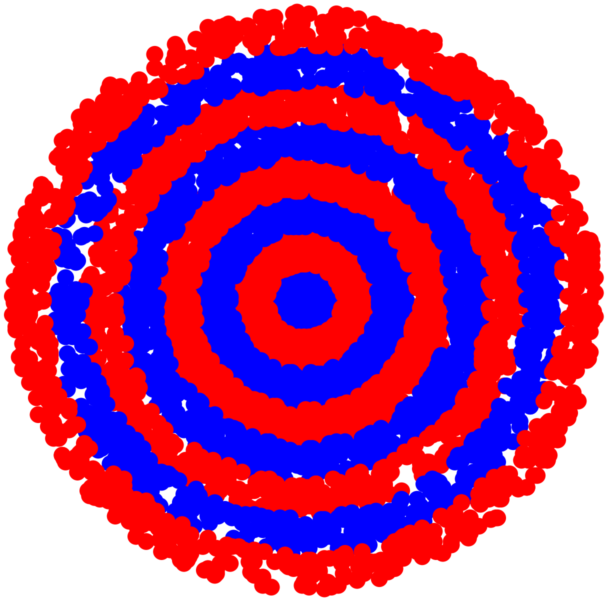
User-defined functions

- We can write our own functions to perform a specific task
  - Example: generate a random floating point number in a specified interval
  - Example: convert polar coordinates to x-y (Cartesian) coordinates

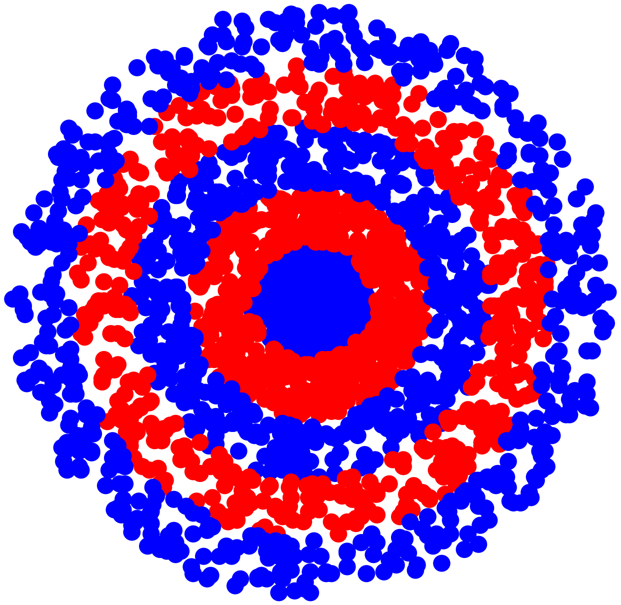
Draw a bulls eye figure with randomly placed dots

- Dots are randomly placed within concentric rings
- User decides how many rings, how many dots





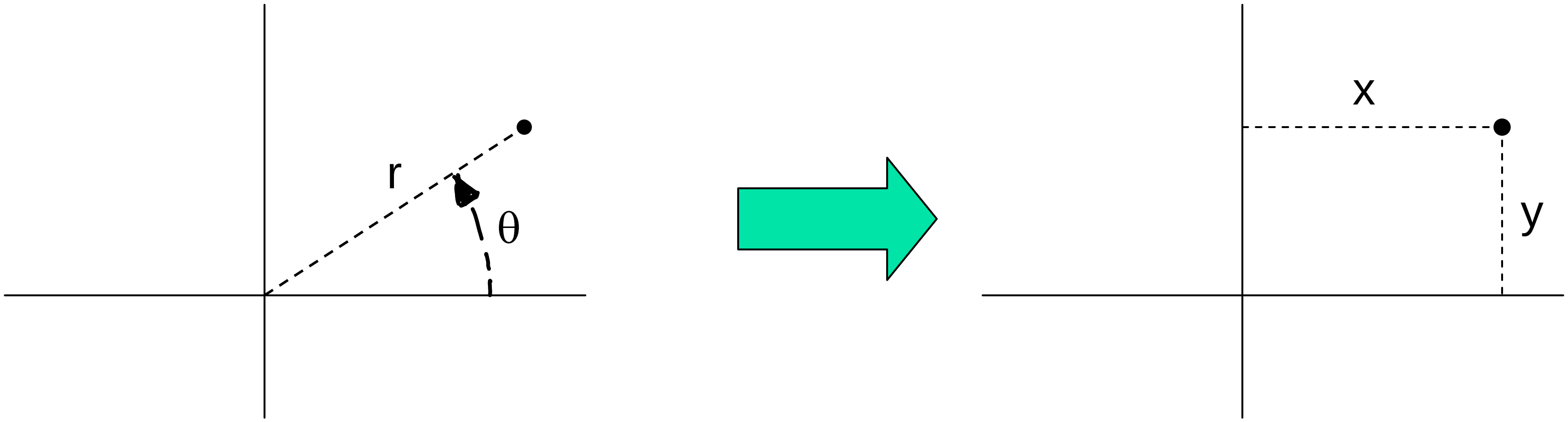
Draw a bulls eye figure with randomly placed dots



- What are the main tasks?
- Accommodate variable number of rings—loop

- For each ring
  - Need many dots
  - For each dot
    - Generate random position
    - Choose color
    - Draw it

Convert from polar to Cartesian coordinates



Polar coordinates

Cartesian coordinates

```
c= input('How many concentric rings? ');
d= input('How many dots? ');

% Put dots btwn circles with radii rRing and (rRing-1)
for rRing= 1:c
  % Draw d dots
  for count= 1:d

    % Generate random dot location (polar coord.)
    theta= _______
    r= _______

    % Convert from polar to Cartesian
    x= _______
    y= _______

    % Use plot to draw dot
  end
end
```

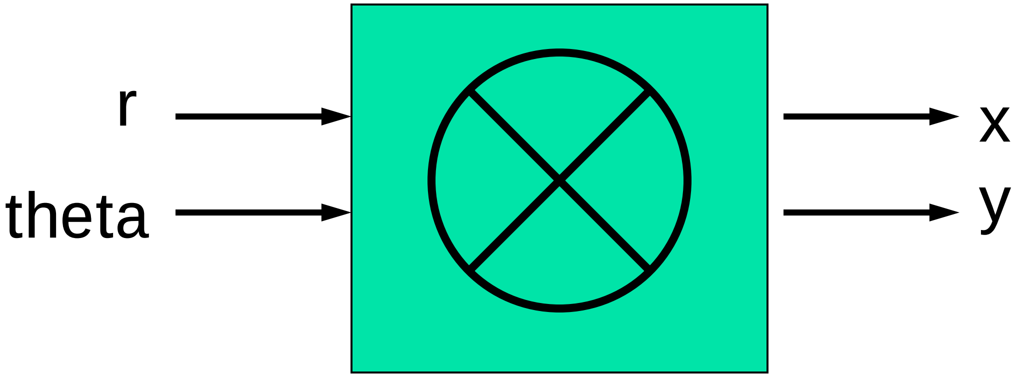
A common task! Create a function **polar2xy** to do this. **polar2xy** likely will be useful in other problems as well.

```
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 [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);
```

Think of **`polar2xy`** as a factory



```
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`**

```
r= input(‘Enter radius: ’);
theta= input(‘Enter angle in degrees: ’);

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

(Part of) a script file

```
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`**

```
r= input(‘Enter radius: ’);
theta= input(‘Enter angle in degrees: ’);

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

(Part of) a script file

```
function [x, y] = polar2xy(r,theta)
```

Output parameter list enclosed in [ ]

Function name (This file's name is **`polar2xy.m`**)

Input parameter list enclosed in ( )

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*’)
…
```

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*')
…
```

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*’)
…
```

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.

dotsInCircles.m

(functions with multiple input parameters)

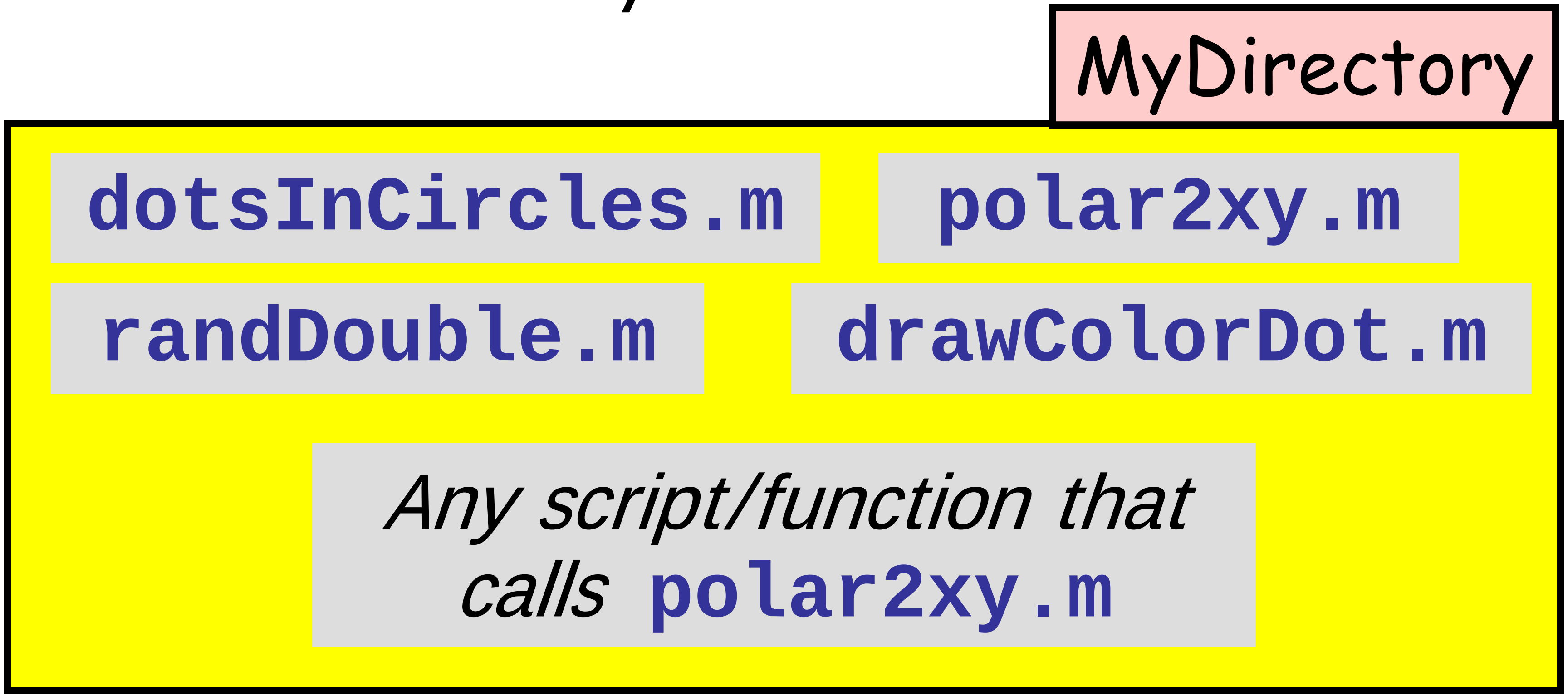
(functions with a single output parameter)

(functions with multiple output parameters)

(functions with no output parameter)

Accessing your functions

For now*, put your related functions and scripts in the same directory.



MyDirectory

`dotsInCircles.m`

`polar2xy.m`

`randDouble.m`

`drawColorDot.m`

Any script/function that calls `polar2xy.m`

*The **`path`** function gives greater flexibility