

- Previous Lecture:
  - Vectors
  - Color computation
  - Linear interpolation
- Today's Lecture:
  - Vectorized operations
  - 2-d array—matrix
- Announcements:
  - Discussion this week in classrooms as listed in Student Center
  - **Prelim I** on 10/16 (Thursday) at 7:30pm

## Initialize arrays 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!

## 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)
```

## Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left  
% corner at (a,b), width w, height h,  
% and fill it with a color named by c.  
x= [a  a+w  a+w  a  a]; % x data  
y= [b  b  b+h  b+h  b]; % y data  
fill(x, y, c)
```

A built-in function



## Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h,
% and fill it with a color named by c.
x= [a  a+w  a+w  a  a]; % x data
y= [b  b    b+h  b+h b]; % y data
fill(x, y, c)
```

Built-in function `fill` actually does the "wrap-around" automatically.

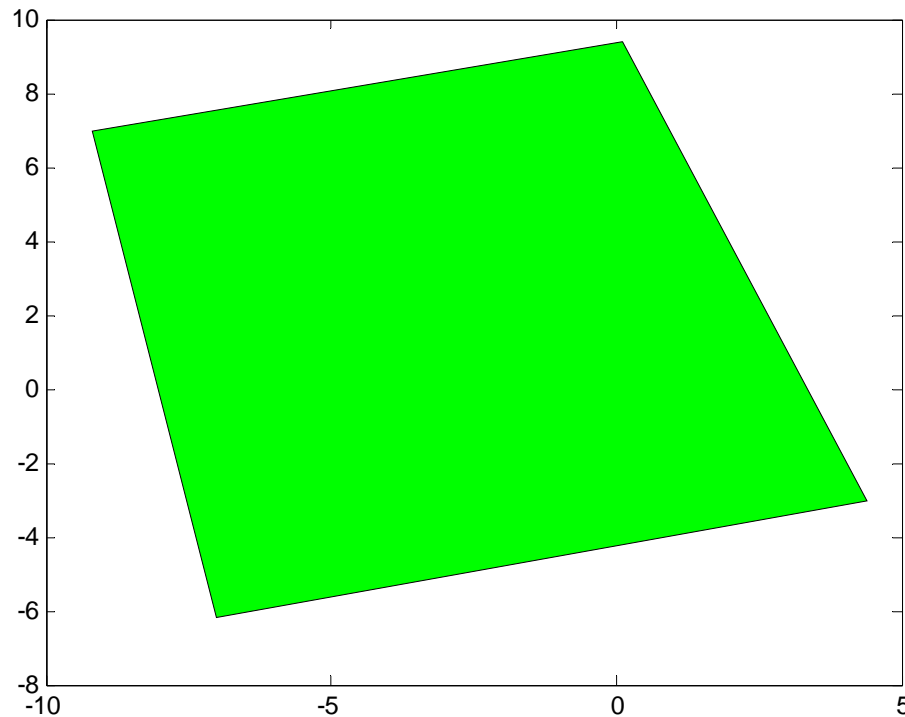
## Coloring a polygon (fill)

```
x= [0.1 -9.2 -7 4.4];
```

```
y= [9.4 7 -6.2 -3];
```

```
fill(x,y,'g')
```

Can be a vector  
(RGB values)



## 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 addition

$$\begin{array}{r} \mathbf{x} \quad \boxed{2 \quad 1 \quad .5 \quad 8} \\ + \quad \mathbf{y} \quad \boxed{1 \quad 2 \quad 0 \quad 1} \\ \hline = \quad \mathbf{z} \quad \boxed{3 \quad 3 \quad .5 \quad 9} \end{array}$$

Matlab code: `z = x + y`

## Vectorized subtraction

$$\begin{array}{r} \mathbf{x} \quad \boxed{2 \quad 1 \quad .5 \quad 8} \\ - \quad \mathbf{y} \quad \boxed{1 \quad 2 \quad 0 \quad 1} \\ \hline = \quad \mathbf{z} \quad \boxed{1 \quad -1 \quad .5 \quad 7} \end{array}$$

Matlab code: `z = x - y`

# Vectorized multiplication

$$\begin{array}{r} \mathbf{a} \\ \times \\ \hline \mathbf{b} \\ \hline \mathbf{c} \end{array}$$

2	1	.5	8
---	---	----	---

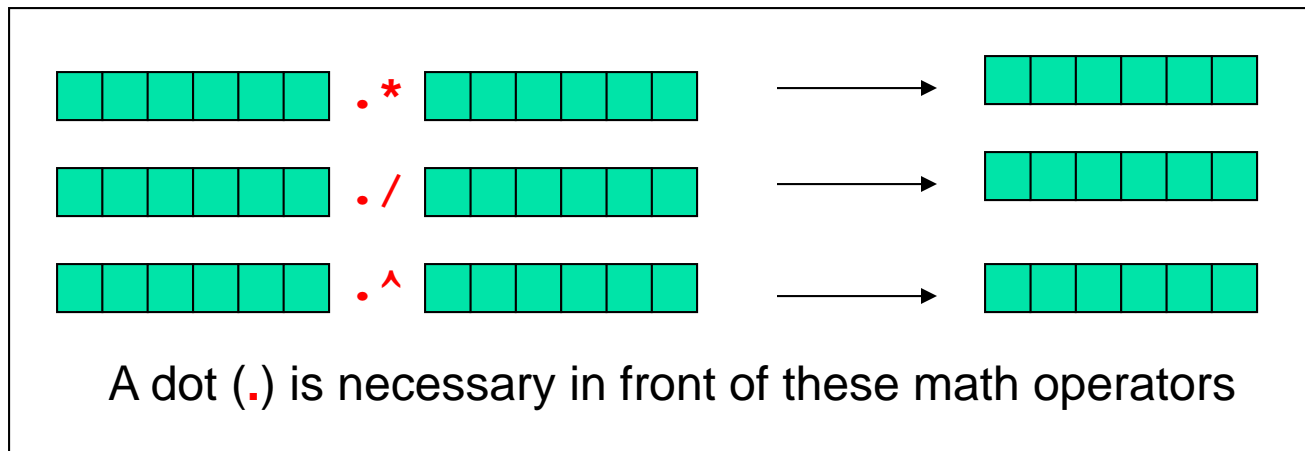
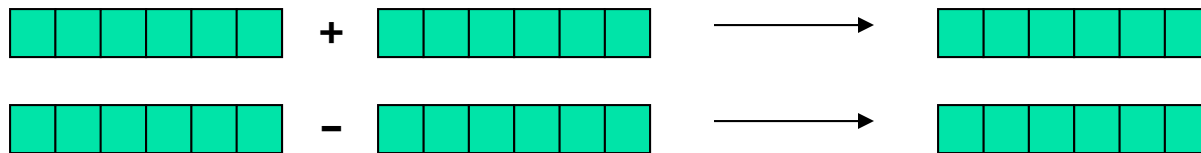
1	2	0	1
---	---	---	---

2	2	0	8
---	---	---	---

Matlab code: `c = a .* b`



# Vectorized element-by-element arithmetic operations on arrays



# Shift

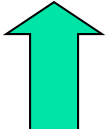
$$\begin{array}{r} \mathbf{x} \quad \boxed{3} \\ + \quad \mathbf{y} \quad \boxed{2 \quad 1 \quad .5 \quad 8} \\ \hline = \quad \mathbf{z} \quad \boxed{5 \quad 4 \quad 3.5 \quad 11} \end{array}$$

Matlab code: `z = x + y`

# Reciprocate

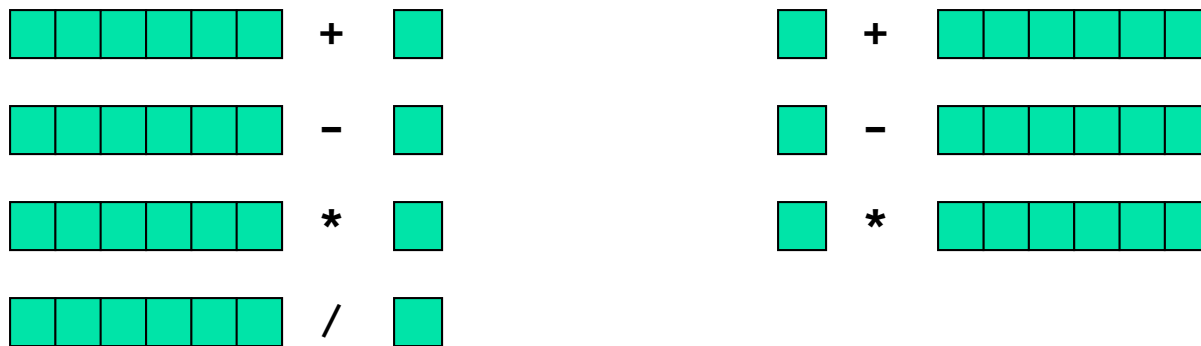
$$\begin{array}{r} \mathbf{x} \quad \boxed{1} \\ / \quad \mathbf{y} \quad \boxed{2 \quad 1 \quad .5 \quad 8} \\ \hline = \quad \mathbf{z} \quad \boxed{.5 \quad 1 \quad 2 \quad .125} \end{array}$$

Matlab code: `z = x ./ y`



# 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  $\text{array} \cdot^* \text{scalar}$  ,  $\text{scalar} \cdot^* \text{array}$  ,  $\text{array} \cdot^/ \text{scalar}$  not necessary but OK

See `plotComparison.m`

Can we plot this?

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

for  
 $-2 \leq x \leq 3$

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 = 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.”

Concatenating 2 vectors—copy 2 vectors into a new one

`% given row vectors x and y`

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given row vectors x and y
```

```
v= zeros(1,length(x)+length(y));
```

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given row vectors x and y
```

```
v= zeros(1,length(x)+length(y));
```

```
for k=1:length(x)
```

```
    v(k)= x(k);
```

```
end
```

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given row vectors x and y
v= zeros(1,length(x)+length(y));
for k=1:length(x)
    v(k)= x(k);
end
for k=1:length(y)
    v(length(x)+k)= y(k);
end
```

This is **non-vectorized code**—operations are performed on one component (scalar) at a time

Concatenating 2 vectors—copy 2 vectors into a new one

```
% given row vectors x and y
v= zeros(1,length(x)+length(y));
for k=1:length(x)
    v(k)= x(k);
end
for k=1:length(y)
    v(length(x)+k)= y(k);
end
```

Below is **vectorized code**—ops are performed on multiple components (a vector) at the same time:

**$v = [x \ y];$**

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
```

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split pt
x= zeros(1,s);
y= zeros(1,length(v)-s);
```



Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split pt
x= zeros(1,s);
y= zeros(1,length(v)-s);
for k=1:s
    x(k)= v(k);
end
```

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split pt
x= zeros(1,s);
y= zeros(1,length(v)-s);
for k=1:s
    x(k)= v(k);
end
for k=1:length(y)
    y(k)= v(s+k);
end
```

This is **non-vectorized code**—operations are performed on one component (scalar) at a time

Split a vector in 2—copy values into 2 vectors

```
% given row vector v
s= ceil(rand*length(v)); % split pt
x= zeros(1,s);
y= zeros(1,length(v)-s);
for k=1:s
    x(k)= v(k);
end
for k=1:length(y)
    y(k)= v(s+k);
end
```

Below is **vectorized code**:  
multiple components  
(subvectors) are  
affected/accessed at the same  
time:

```
x= v(1:s);
y= v(s+1:length(v));
```

End of  
Prelim 1 material

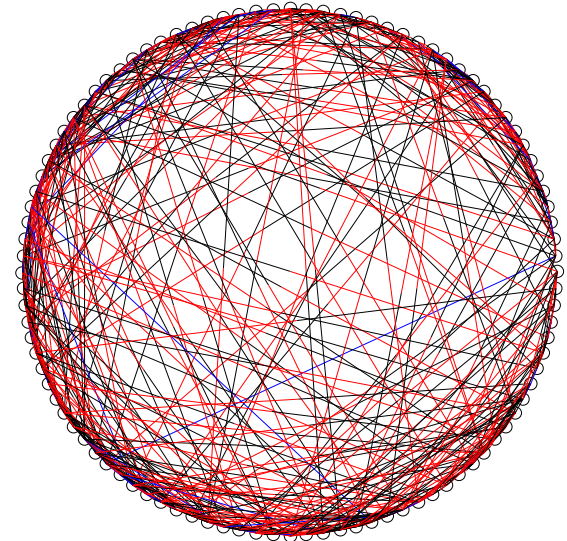
# Storing and using data in tables

A company has 3 factories that make 5 products with these costs:

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

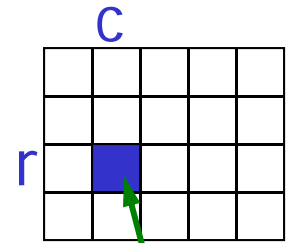
What is the best way to fill a given purchase order?



Connections  
between webpages

0	0	1	0	1	0	0
1	0	0	1	1	1	0
0	1	0	1	1	1	1
1	0	1	1	0	1	0
0	0	1	1	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	1	0

## 2-d array: **matrix**



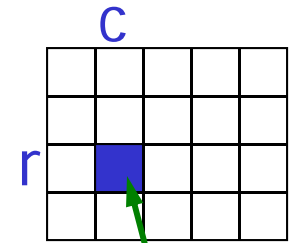
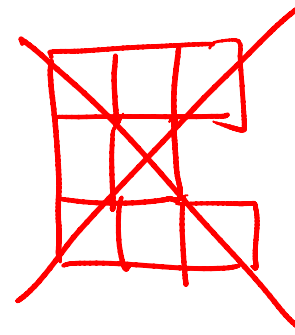
- An array is a **named** collection of **like** data organized into rows and columns
- A 2-d array is a table, called a **matrix**
- Two **indices** identify the position of a value in a matrix, e.g.,

`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

## 2-d array: **matrix**



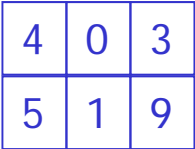
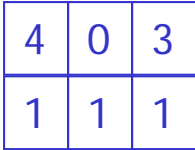
- An array is a **named** collection of **like** data organized into rows and columns
- A 2-d array is a table, called a **matrix**
- Two **indices** identify the position of a value in a matrix, e.g.,

**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



Working with a matrix:  
**size** and individual components

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

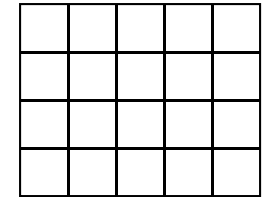
Given a matrix M

```
[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;
```

Example: minimum value in a matrix

function val = minInMatrix(M)

% val is the smallest value in matrix M



Example: minimum value in a matrix

function val = minInMatrix(M)

% val is the smallest value in matrix M

```
[nr, nc] = size(M);
```

```
val = M(1,1);
```

```
for r = 1:nr
```

```
    % At row r
```

```
    for c = 1:nc
```

```
        % At col c (at row r)
```

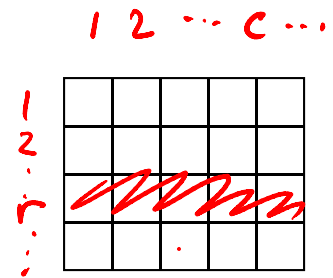
```
        if M(r,c) < val
```

```
            val = M(r,c);
```

```
        end
```

```
    end
```

```
end
```



## Pattern for traversing a matrix M

```
[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
end
end
```