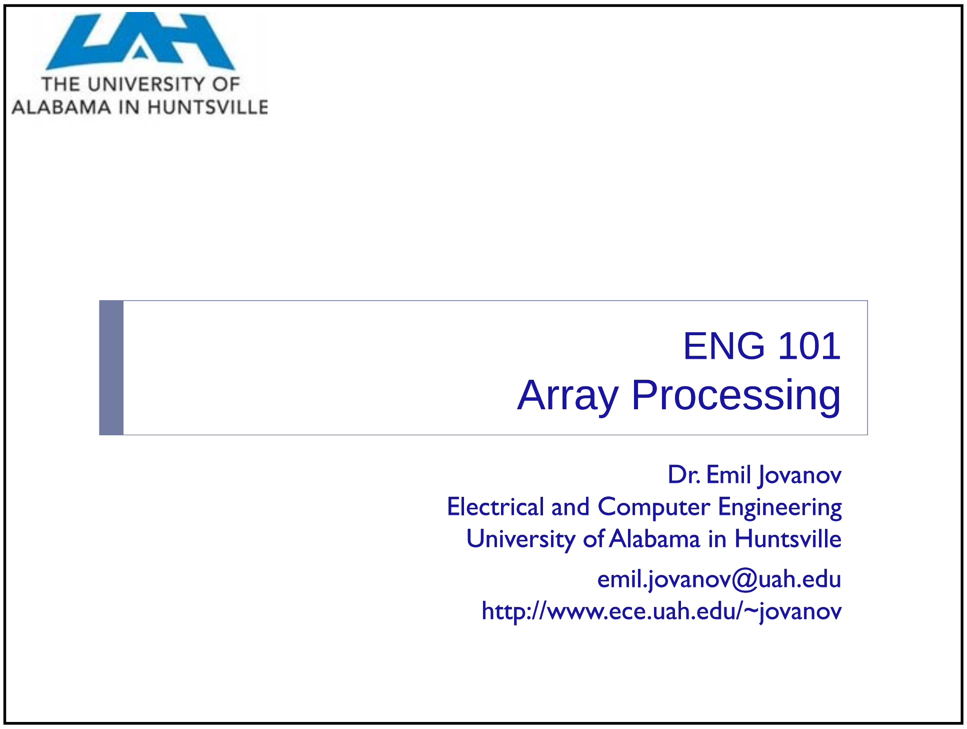
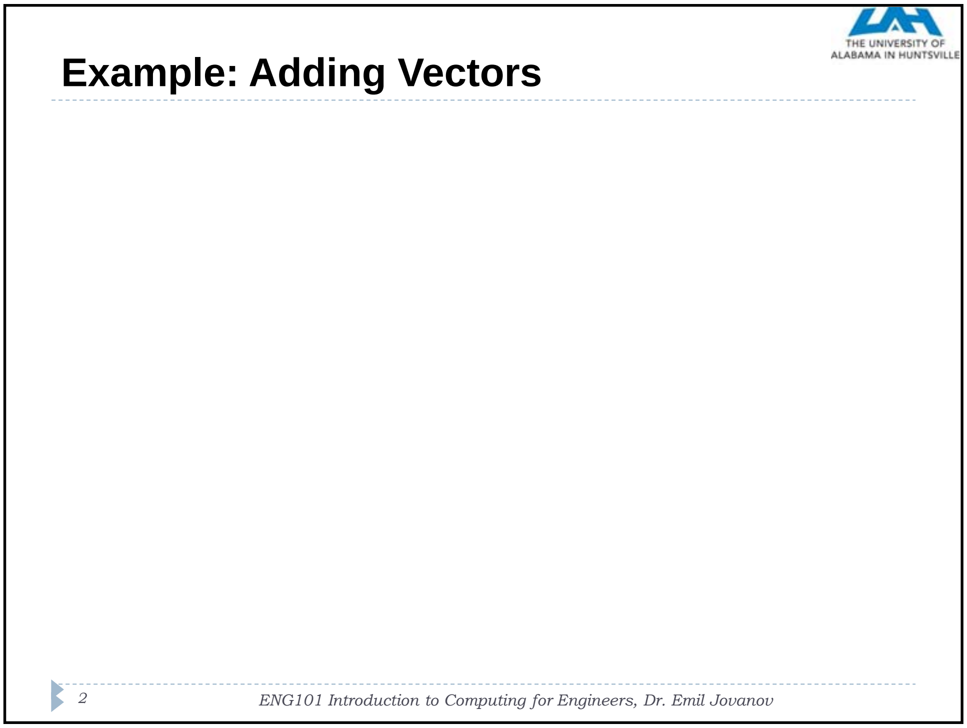


ENG 101
Array Processing

Dr. Emil Jovanov
Electrical and Computer Engineering
University of Alabama in Huntsville

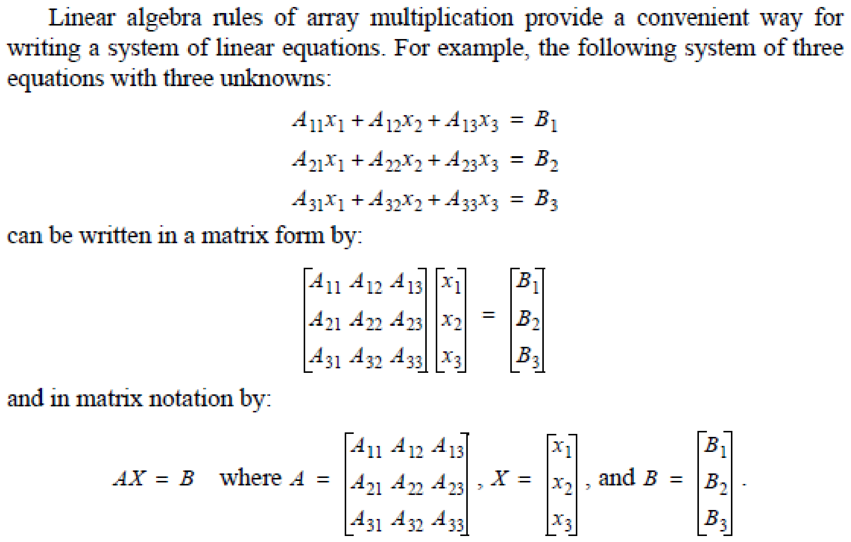
emil.jovanov@uah.edu
http://www.ece.uah.edu/~jovanov



Example: Adding Vectors

Addition and Subtraction

- When adding a scalar to an array, MATLAB adds the scalar to every element of the array
- When subtracting a scalar from an array, MATLAB subtracts the scalar from every element of the array
- Example: scalar c and matrix
  $A = [A_{11} \quad A_{12} \quad A_{13}$
  $A_{21} \quad A_{22} \quad A_{23}]$
- $A+c = \quad [c+A_{11} \quad c+A_{12} \quad c+A_{13}$
  $c+A_{21} \quad c+A_{22} \quad c+A_{23}]$



Linear algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the following system of three equations with three unknowns:

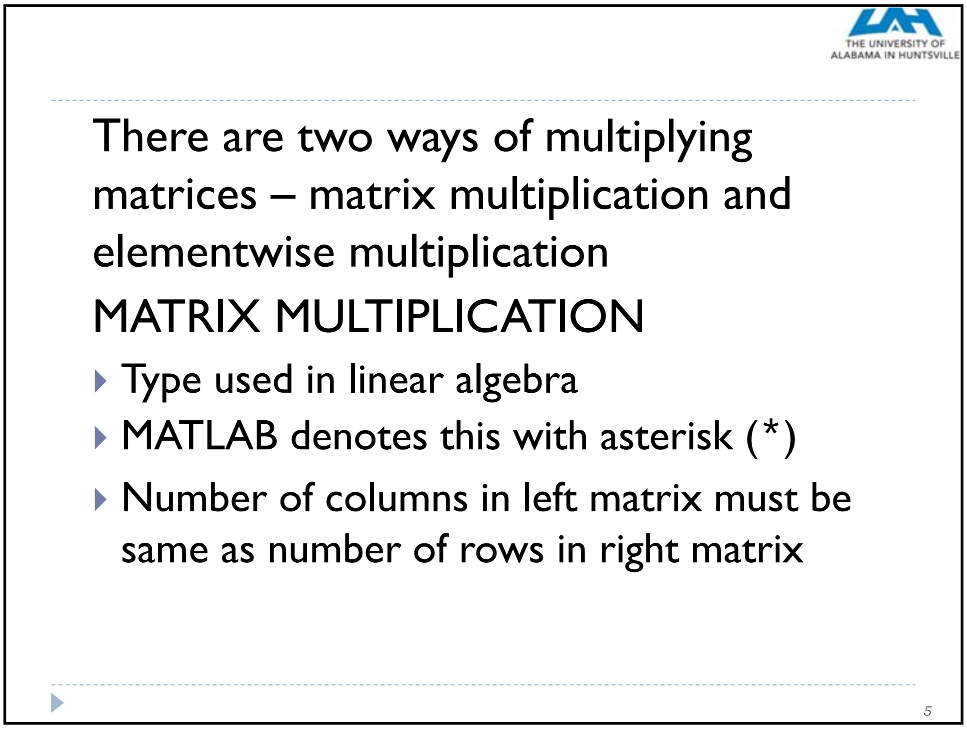
$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + A_{13}x_3 &= B_1 \\ A_{21}x_1 + A_{22}x_2 + A_{23}x_3 &= B_2 \\ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 &= B_3 \end{aligned}$$

can be written in a matrix form by:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

and in matrix notation by:

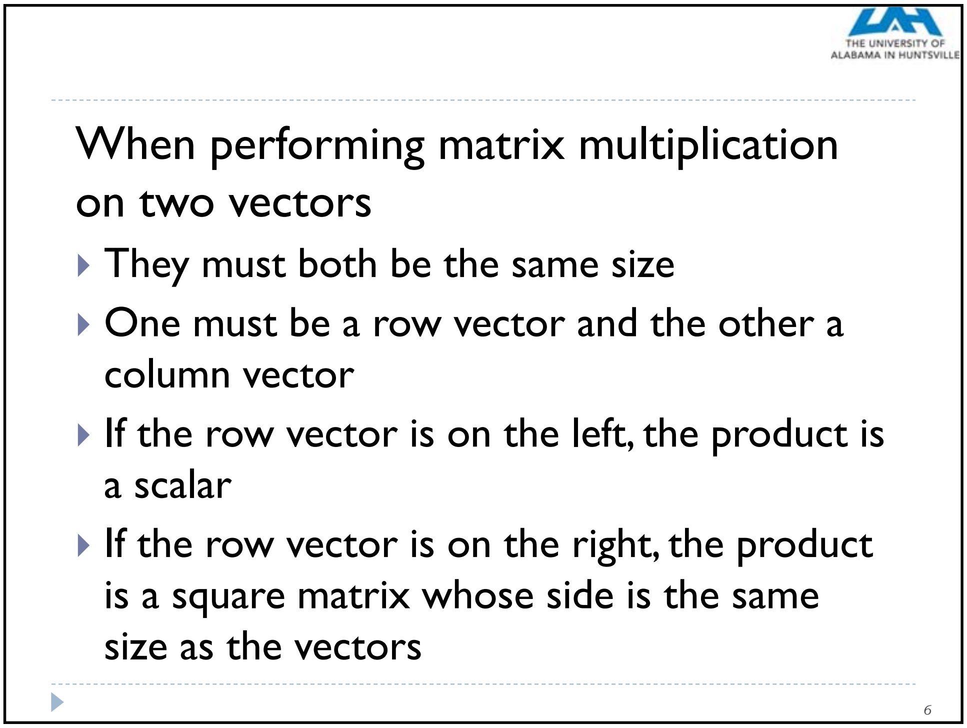
$$AX = B \quad \text{where } A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \text{ and } B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}.$$



There are two ways of multiplying matrices – matrix multiplication and elementwise multiplication

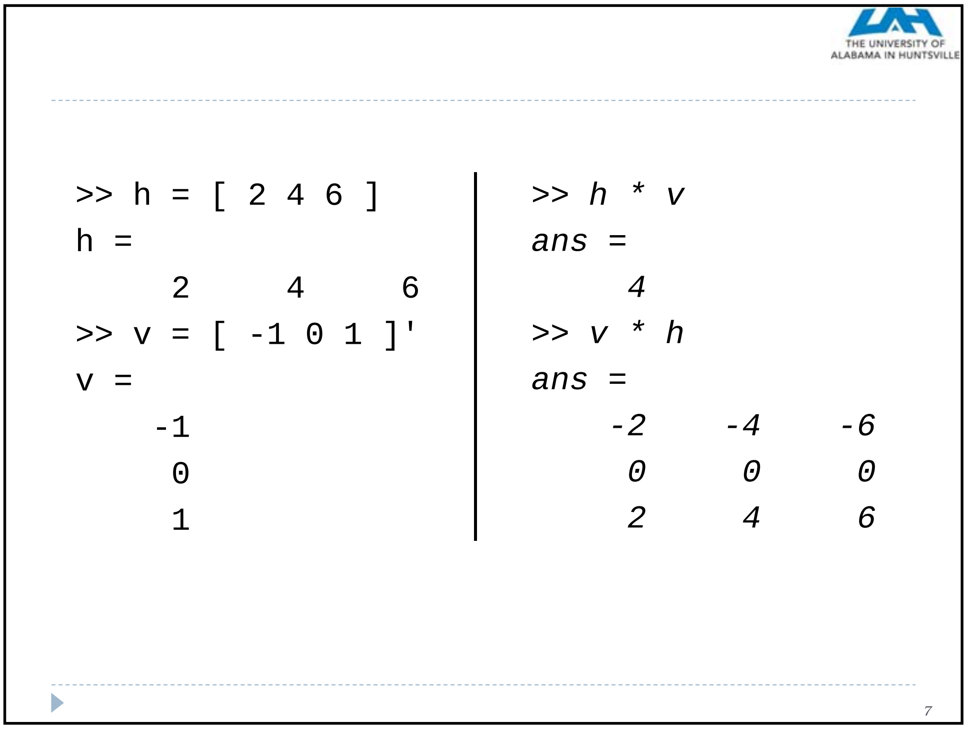
MATRIX MULTIPLICATION

- Type used in linear algebra
- MATLAB denotes this with asterisk (*)
- Number of columns in left matrix must be same as number of rows in right matrix



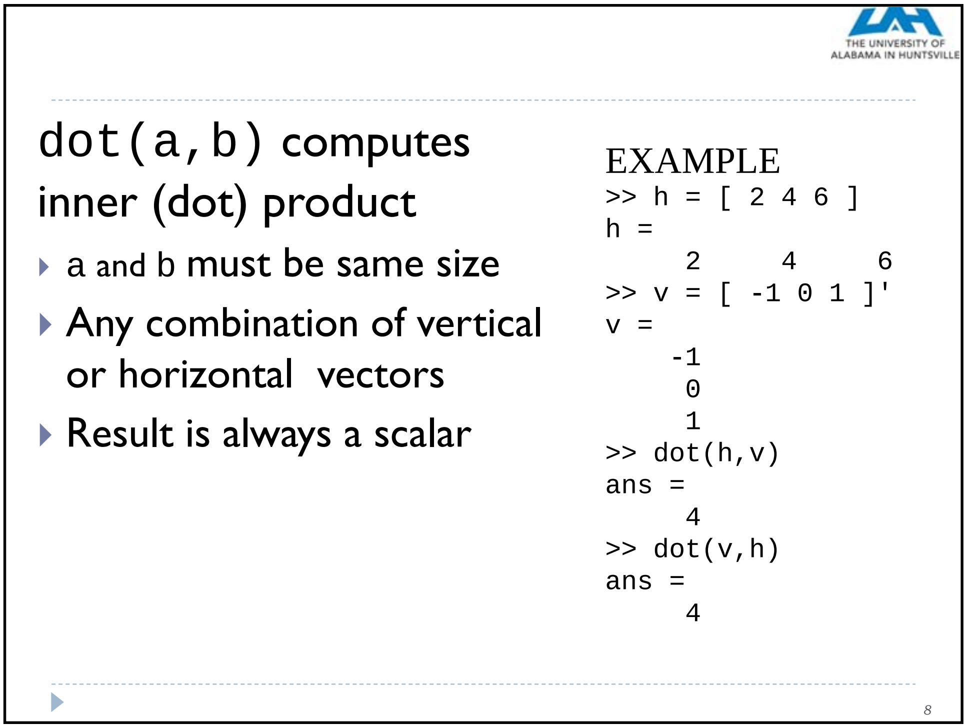
When performing matrix multiplication on two vectors

- They must both be the same size
- One must be a row vector and the other a column vector
- If the row vector is on the left, the product is a scalar
- If the row vector is on the right, the product is a square matrix whose side is the same size as the vectors



```
>> h = [ 2 4 6 ]
h =
     2     4     6
>> v = [ -1 0 1 ]'
v =
    -1
     0
     1
```

```
>> h * v
ans =
     4
>> v * h
ans =
    -2    -4    -6
     0     0     0
     2     4     6
```

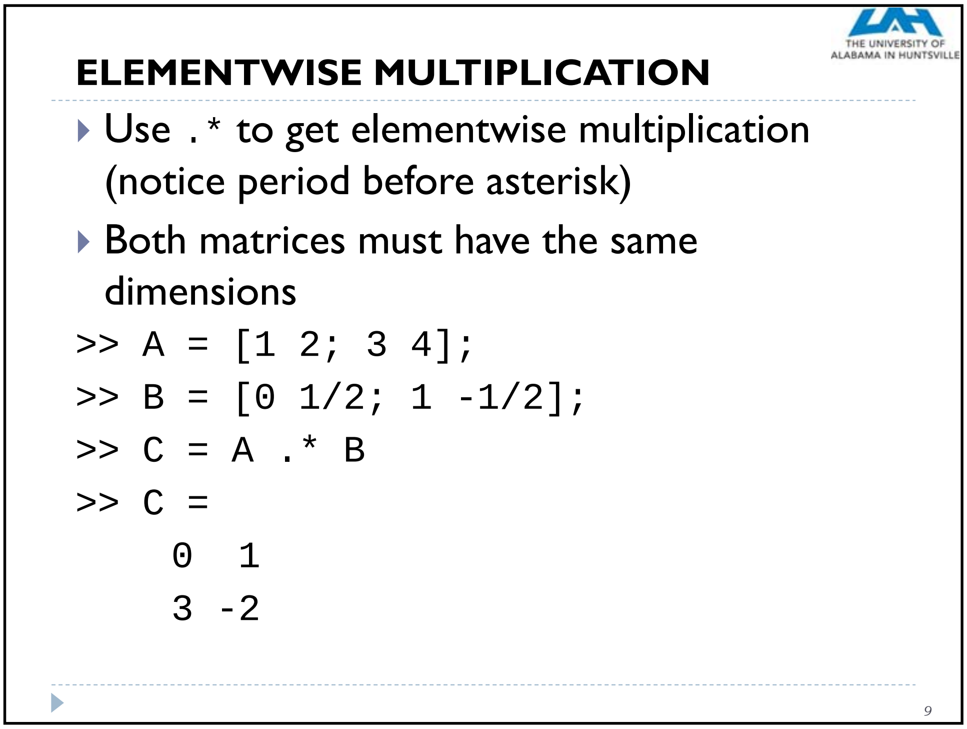


`dot(a,b)` computes inner (dot) product

- a and b must be same size
- Any combination of vertical or horizontal vectors
- Result is always a scalar

EXAMPLE

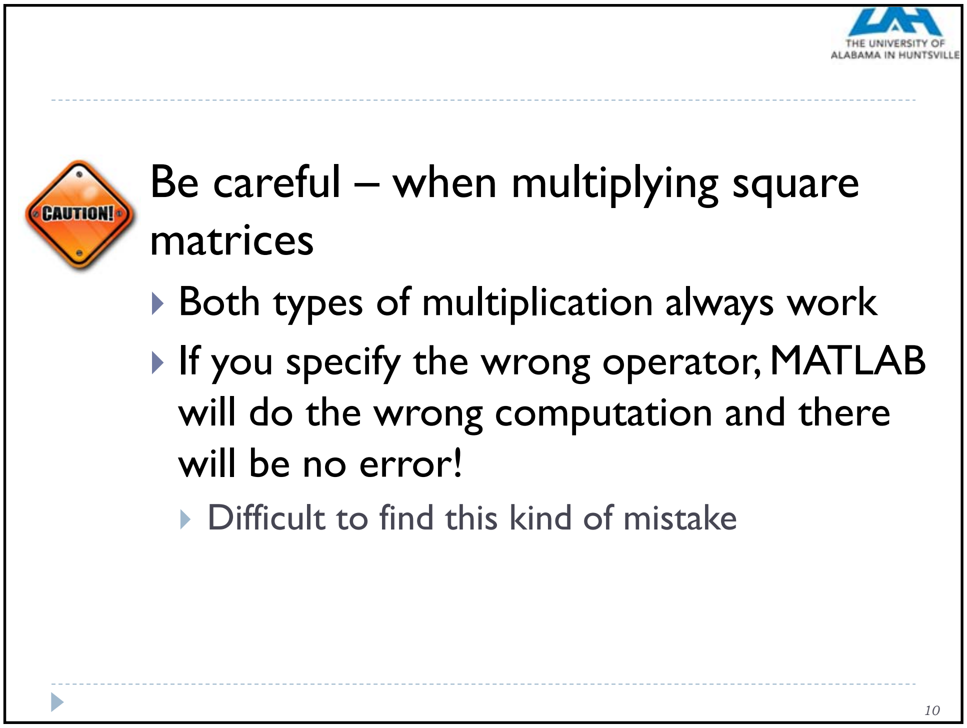
```
>> h = [ 2 4 6 ]
h =
     2     4     6
>> v = [ -1 0 1 ]'
v =
    -1
     0
     1
>> dot(h,v)
ans =
     4
>> dot(v,h)
ans =
     4
```



ELEMENTWISE MULTIPLICATION

- Use `.*` to get elementwise multiplication (notice period before asterisk)
- Both matrices must have the same dimensions

```
>> A = [1 2; 3 4];
>> B = [0 1/2; 1 -1/2];
>> C = A .* B
>> C =
     0  1
     3 -2
```



Be careful – when multiplying square matrices

- Both types of multiplication always work
- If you specify the wrong operator, MATLAB will do the wrong computation and there will be no error!
  - Difficult to find this kind of mistake

EXAMPLE

```
>> A = [1 2; 3 4];
>> B = [0 1/2; 1 -1/2];
>> A .* B
>> ans
    0  1
    3 -2
>> A * B
ans =
    2.0000  -0.5000
    4.0000  -0.5000
```

Random numbers often used in MATLAB engineering applications

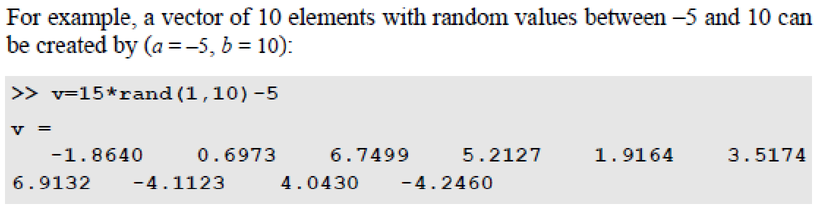
- Simulate noise
- Useful in certain mathematical computations, such as Monte Carlo simulations

MATLAB has three commands that create random numbers – `rand`, `randn`, `randi`

- All can create scalars, vectors, or matrices of random numbers

`rand` generates random numbers uniformly distributed between 0 and 1

- To get numbers between `a` and `b`, multiply output of `rand` by `b-a` and add `a`, i.e., `(b-a)*rand + a`



For example, a vector of 10 elements with random values between –5 and 10 can be created by (a = –5, b = 10):

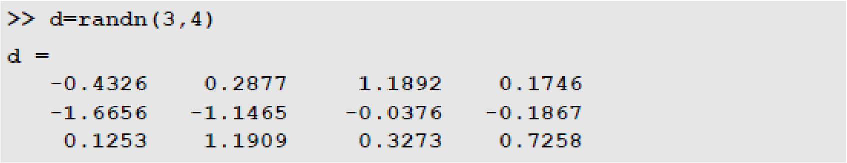
```
>> v=15*rand(1,10)-5
v =
   -1.8640    0.6973    6.7499    5.2127    1.9164    3.5174
6.9132   -4.1123    4.0430   -4.2460
```

`randi` generates uniformly distributed random integers in a specified range

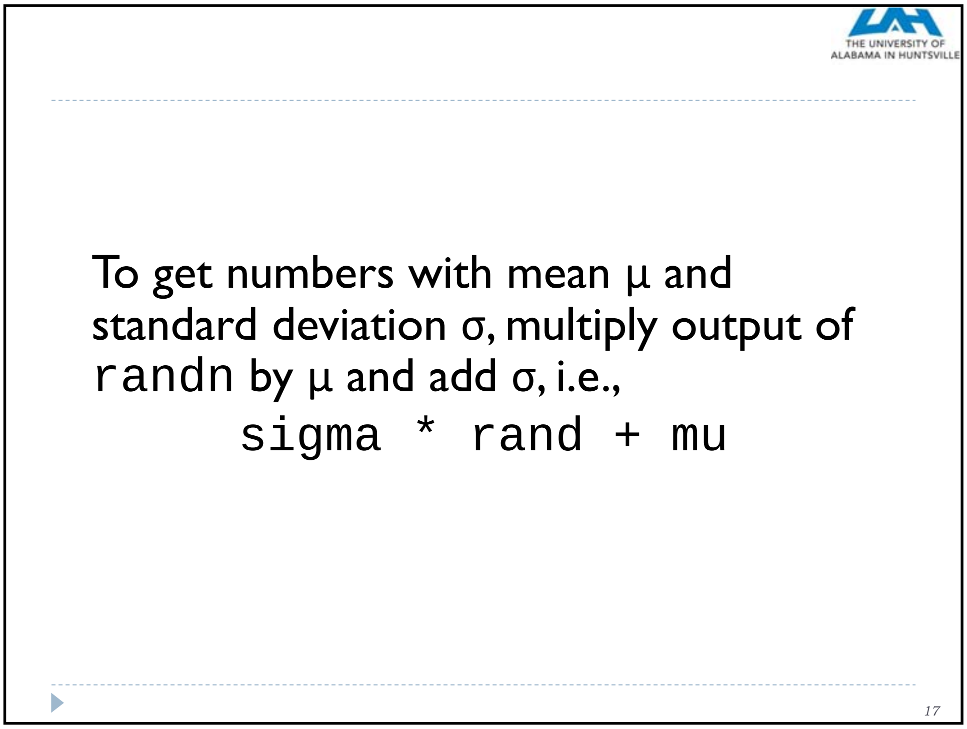
For example, to make a *3 ×4* of random numbers between 50 and 90

```
>> d=randi( [50 90],3,4)
d =
    57   82   71   75
    66   52   67   61
    84   66   76   67
```

`randn` generates random numbers from a normal distribution with mean 0 and standard deviation 1

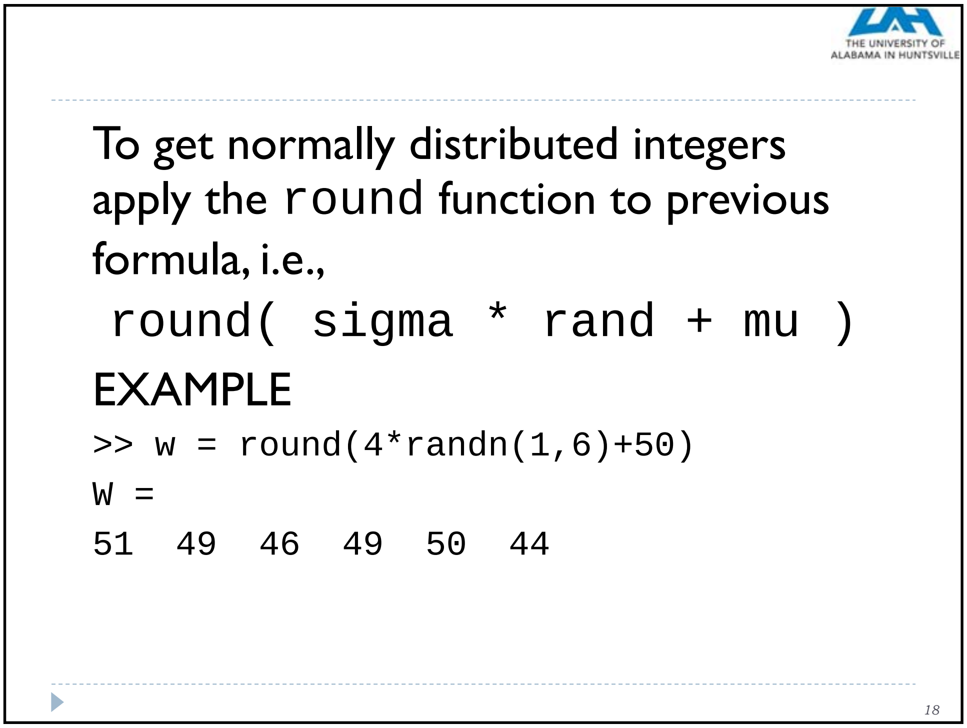


```
>> d=randn(3,4)
d =
   -0.4326    0.2877     1.1892     0.1746
   -1.6656   -1.1465    -0.0376    -0.1867
    0.1253    1.1909     0.3273     0.7258
```



To get numbers with mean μ and standard deviation σ, multiply output of `randn` by μ and add σ, i.e.,

```
      sigma * rand + mu
```



To get normally distributed integers apply the `round` function to previous formula, i.e.,

```
 round( sigma * rand + mu )
```

EXAMPLE

```
>> w = round(4*randn(1,6)+50)
W =
51  49  46  49  50  44
```