Computer Programming

MENG2020

REVIEW

About our lecture

What Are Arrays?

The *array* is MATLAB's basic data structure.

- It can have any number of dimensions. Most common are:
  - *The vector* : one dimension (a single row or column)
  - *The matrix :* two or more dimensions (we will stop at 2).

Creating a row vector

To create a row vector from known numbers, just type a variable name, then the equal sign, then inside square brackets, the numbers separated by spaces or commas.

```
variable_name = [ n1, n2, n3 ]
```

Commas are optional

```
>> yr = [1984 1986 1988 1990 1992 1994 1996]

yr =

  1984 1986 1988 1990 1992 1994 1996
```

Note: MATLAB displays a row vector horizontally

Creating a Column Vector

To create a column vector from known numbers you can use one of two methods:

Method 1 : same as row vector but put semicolon after all but the last number

- variable_name = [n1; n2; n3]
- >> yr = [1984; 1986; 1988]
- yr =
  - 1984
  - 1986
  - 1988

Note: MATLAB displays a column vector vertically

Creating a column vector

- Method 2 : same as row vector but put an apostrophe (') after the closing bracket.
  - The apostrophe interchanges rows and columns. We will come back on this later.

```
variable_name = [ n1 n2 n3 ]'
```

```
>> yr = [1984 1986 1988 ]'

yr =

1984

1986

1988
```

Constant spacing vectors

To create a vector with specified constant spacing between elements, use the m:q:n command. MATLAB will create the vector automatically with the required number of elements.

```
variable_name = m:q:n
```

`m` is the first number
`n` is the last number
`q` is the difference between consecutive numbers

```
v = m:q:n
```

means

```
v = [ m m+q m+2q m+3q ... n ]
```

Constant spacing vectors

If q is omitted, the spacing is 1

```
v = m:n
```

means

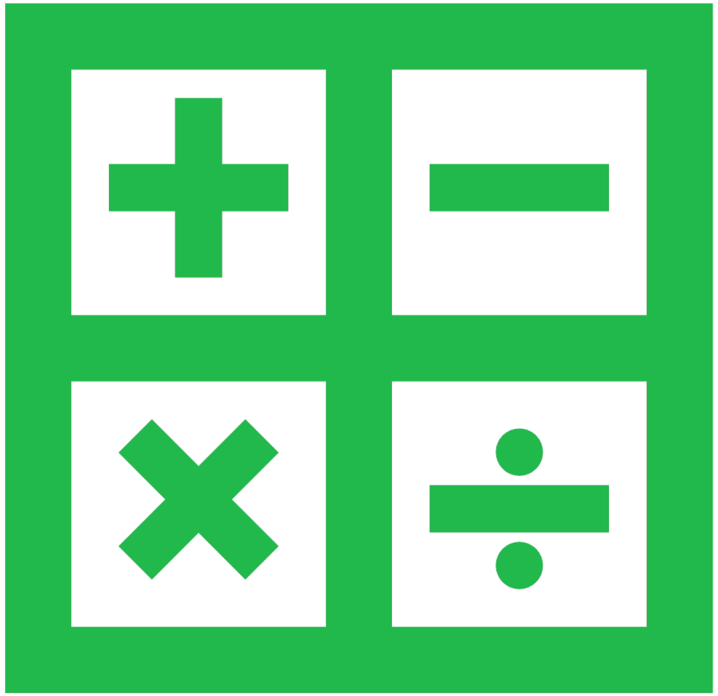
```
v = [ m m+1 m+2 m+3 ... n ]
```

```
>> x = 1:2:13
x = 1 3 5 7 9 11 13
>> y = 1.5:0.1:2.1
y = 1.5000 1.6000 1.7000 1.8000 1.9000
 2.0000 2.1000
```

Non-integer spacing

Q?

```
>> z = -3:7
```



Constant spacing vectors

```
>> z = -3:7
```

z = -3 -2 -1 0 1 2 3 4 5 6 7

Q?

```
>> xa = 21:-3:6
```

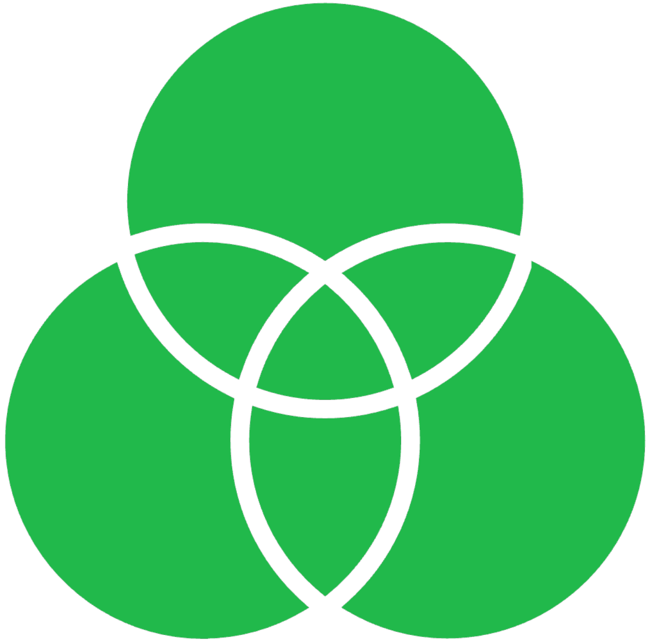
Constant spacing vectors

```
>> xa = 21:-3:6
```

```
xa = 21 18 15 12 9 6
```

Q?

```
>> fred = 7:-2:15
```



Constant spacing vectors

```
>> z = -3:7

z = -3 -2 -1 0 1 2 3 4 5 6 7

>> xa = 21:-3:6
```

Negative spacing

```
xa = 21 18 15 12 9 6

>> fred = 7:-2:15
```

Impossible spacing!!!

```
fred =  Empty matrix: 1-by-0
```

Fixed length vectors

To create a vector with a specific number of terms between the first and the last, we use the **linspace** command. MATLAB will take care of the spacing.

```
v = linspace(xi, xf, n)
```

- `xi` is the first number
- `xf` is the last number
- `n` is the number of terms (obviously must be a positive number) (100 is used if omitted)

TIP

```
>> va = linspace(0, 8, 6)

va = 0 1.6000 3.2000 4.8000 6.4000
8.0000
```

Six elements

```
>> va = linspace(30, 10, 11)

va = 30 28 26 24 22 20 18 16 14 12 10
```

Decreasing elements

`m:q:n` lets you directly specify spacing. `linspace()` lets you directly specify the number of terms.

Quick *linspace* Exercises

1. How would you use linspace to set up spacings for planting seedlets in a garden row (30 seedlets, each row 2 meters long).

2. Plan a 4285 km (Toronto to Los Angeles) road trip taking 21 days. How far would you travel each day?

Matrices

You can create a two-dimensional matrix like this:

```
m = [ row 1 numbers; row 2 numbers; ... ; last
  row numbers ]
```

- Each row is separated by a semicolon.
- All rows must have the same number of columns.

Q?

```
>> a = [5 35 43; 4 76 81; 21 32 40]
```

Matrices

You can create a two-dimensional matrix like this:

```
m = [ row 1 numbers; row 2 numbers; ... ; last
  row numbers ]
```

- Each row is separated by a semicolon. All rows must have the same number of columns.

```
>> a = [5 35 43; 4 76 81; 21 32 40]

a =

     5    35    43

     4    76    81

    21    32    40
```

Matrices

You can use expressions to build matrices.

```
>> cd=6; e=3; h=4;
```

Commas are optional

```
>> Mat=[e, cd*h, cos(pi/3);...
 h^2 sqrt(h*h/cd) 14]

Mat =

    3.0000   24.0000    0.5000
   16.0000    1.6330   14.0000
```

Matrices

You can also use m:p:n or linspace() to make rows.

- Make sure each row has the same number of columns!

```
>> A=[1:2:11; 0:5:25;...
 linspace(10,60,6); 67 2 43 68 4 13]
```

Matrices

You can also use m:p:n or linspace() to make rows.

- Make sure each row has the same number of columns!

```
>> A=[1:2:11; 0:5:25;...
 linspace(10,60,6); 67 2 43 68 4 13]

A =

     1     3     5     7     9    11

     0     5    10    15    20    25

    10    20    30    40    50    60

    67     2    43    68     4    13
```

Q?

```
>> B= [1:4; linspace(1,4,5)]
```

Matrices

What if the number of columns is different? You get an error!

Four columns Five columns

```
>> B= [1:4; linspace(1,4,5)]
??? Error using ==> vertcat
CAT arguments dimensions are not
 consistent.
```

Special Matrix Commands

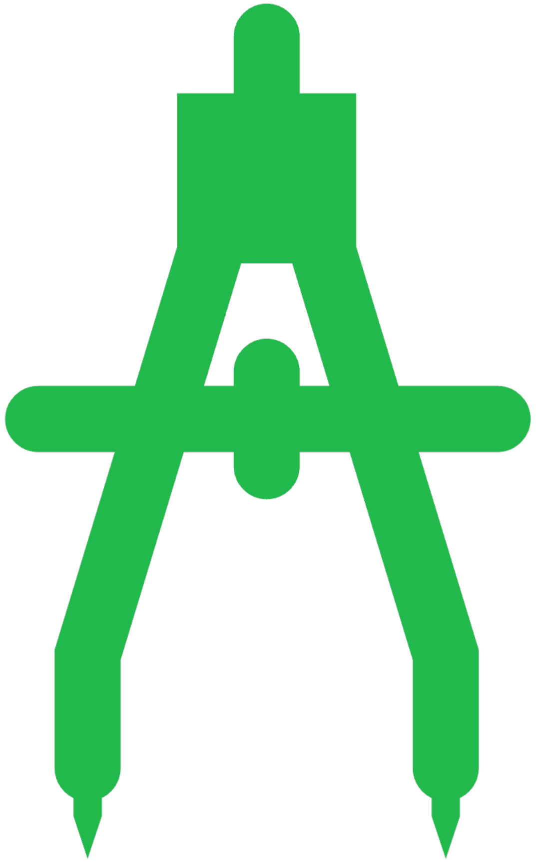
zeros(m,n) - makes a matrix of m rows and n columns, all with zeros.

ones(m,n) - makes a matrix of m rows and n columns, all with ones.

eye(n) - makes a square matrix of n rows and columns. Main diagonal (upper left to lower right) has ones, all other elements are zero.

Q?

```
+>> zr=zeros(3,4)
```



```
>> zr=zeros(3,4)

zr = 0   0   0   0

     0   0   0   0

     0   0   0   0
```

```
>> ne=ones(4,3)
```

```
>> zr=zeros(3,4)

zr = 0   0   0   0

     0   0   0   0

     0   0   0   0


>> ne=ones(4,3)

ne = 1   1   1

     1   1   1

     1   1   1

     1   1   1
```

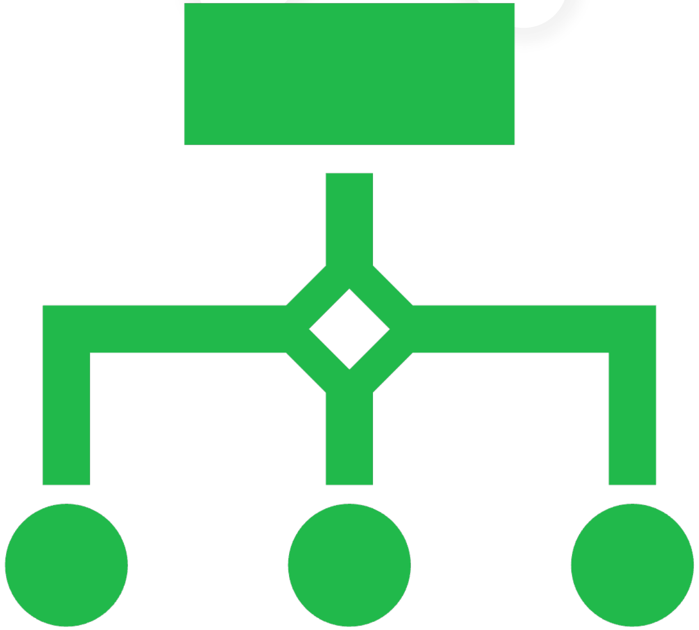
```
>> idn=eye(5)
```

```
>> zr=zeros(3,4)

zr = 0   0   0   0

     0   0   0   0

     0   0   0   0


>> ne=ones(4,3)

ne = 1   1   1

     1   1   1

     1   1   1

     1   1   1
```

```
>> idn=eye(5)

idn = 1   0   0   0   0

      0   1   0   0   0

      0   0   1   0   0

      0   0   0   1   0

      0   0   0   0   1
```

Q?

```
>> z=100*ones(3,4)
```



TIP

To make a matrix filled with a particular number, multiply `ones(m,n)` by that number

```
>> z=100*ones(3,4)

z =

   100   100   100   100
   100   100   100   100
   100   100   100   100
```

Q?

TASK

Create a variable named `lambdaEnd` (λ_{end}) that contains the value of the last wavelength in the recorded spectrum. You can calculate `lambdaEnd` with the equation $\lambda_{start} + (nObs - 1)\lambda_{delta}$.

Use `lambdaEnd` to make a vector named `lambda` (λ) containing the wavelengths in the spectrum, from λ_{start} to λ_{end}, in steps of λ_{delta}.

TASK

Create a variable named `lambdaEnd` (λ_{end}) that contains the value of the last wavelength in the recorded spectrum. You can calculate `lambdaEnd` with the equation $\lambda_{start} + (nObs - 1)\lambda_{delta}$.

Use `lambdaEnd` to make a vector named `lambda` (λ) containing the wavelengths in the spectrum, from λ_{start} to λ_{end}, in steps of λ_{delta}.

Task 1

```
lambdaEnd = lambdaStart + (nObs-1)*lambdaDelta
lambda = (lambdaStart:lambdaDelta:lambdaEnd)
```

Filling Arrays with Random Numbers

You can also fill vectors and matrices with random numbers.

MATLAB has three commands that create random numbers : **rand**, **randi**, and **randn**.

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

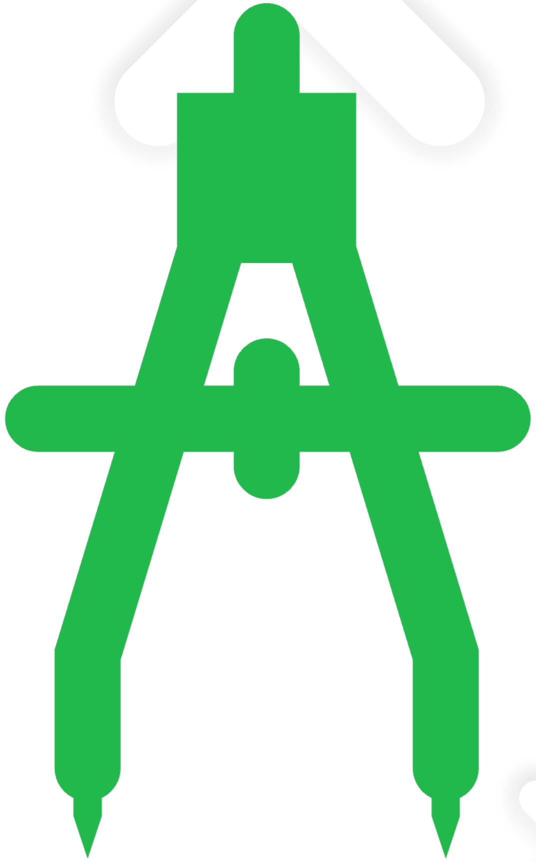
The *rand* command

rand generates random numbers uniformly distributed between 0 and 1

```
r = 3

>> rand(3,1)

ans =

 0.1622

 0.7943

  0.3112
```

Q?

+ Generate 5 Random numbers between 0 and 100?



Generate 5 Random numbers between 0 and 100?

```
>> rand(5,1).*100

ans =

52.8533
16.5649
60.1982
26.2971
65.4079
```

The *rand* command

rand generates random numbers uniformly distributed between 0 and 1

- To get numbers between a and b (inclusive), multiply the output of rand by b-a and add a.
  (b-a)*rand + a
- Use a=1 and b=6 to roll a die!

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
```

A row vector 1x10

The *randi* command

randi generates uniformly distributed random integers in a specified range

For example, to make a 3×4 matrix 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
```

The *randn* command

randn generates random numbers from a normal (bell curve) 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
```

About variables in MATLAB

You remember that all variables are arrays in MATLAB don’t you?

- *Scalar* : an array with only one element (one row, one column)
- *Vector* : an array with only one row or column
- *Matrix* : an array with multiple rows and columns

Note: You don't have to define a variable size before assigning to it, as you do in some older programming languages. Assigning new content to an existing variable changes its dimension to that is required to hold the new content.

The Transpose Operator

- You can transpose a variable (vector or matrix) by putting a single quote after its name, like **x'**

- Transposing a row vector changes it to a column vector and vice-versa.

- It switches rows and columns of a matrix: the first row of the original becomes the first column of the transposed, the second row of the original becomes the second column of the transposed, and so on and so forth...

Q?

```
>> aa=[3 8 1]
```

Create bb which a transpose of a.

Vector Transpose

```
>> aa=[3 8 1]
aa =     3     8     1
>> bb = aa'
bb = 3
     8
     1
```

bb is aa transposed

Matrix Transpose

```
>> C = [2 55 14 8; 21 5 32 11; 41 64 9 1]

C =  2      55      14       8

    21       5      32      11

    41      64       9       1

>> D = C'
```

Matrix Transpose

```
>> C = [2 55 14 8; 21 5 32 11; 41 64 9 1]

C =   2      55      14       8

     21       5      32      11

     41      64       9       1

>> D = C'

D =   2      21      41

     55       5      64

     14      32       9

      8      11       1
```

See the columns becoming rows and the rows becoming columns!

Accessing Elements

We can access (extract) elements or groups of elements in a vector or a matrix.

It is useful for changing an element or a subset of elements.

It is useful for making a new variable from an element or a subset of elements.

Accessing Vector Elements

- You can access any element by indexing the vector name with *parentheses* (indexing starts from 1, not from 0 as in other programming languages like C).

```
>> odd = [1:3:10]
   odd = 1    4    7    10

>> odd(3)
```

Accessing Vector Elements

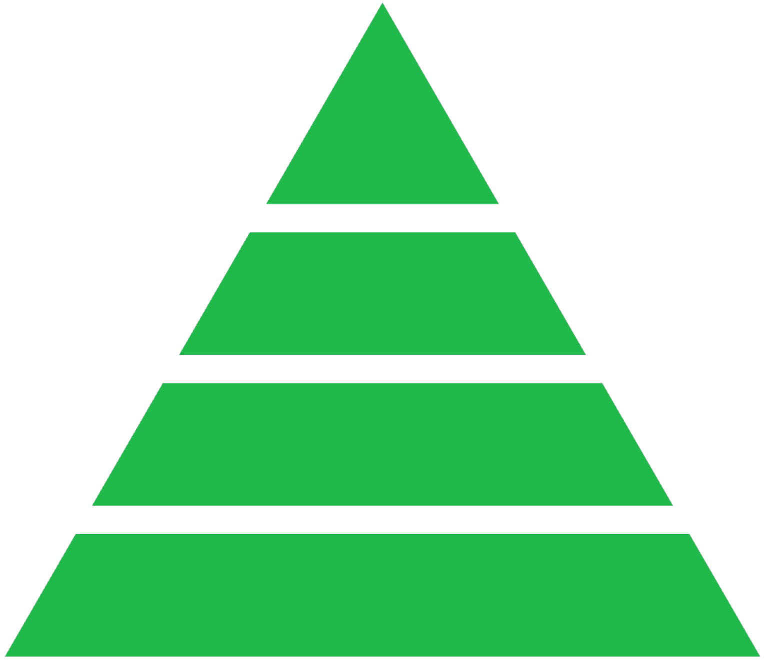
- You can access any element by indexing the vector name with *parentheses* (indexing starts from 1, not from 0 as in other programming languages like C).

```
>> odd = [1:3:10]
   odd = 1    4    7    10

>> odd(3)
   ans = 7
```

Accesses the third element of the vector **odd**

Q?

```
+>> odd(2:4)
```



Accessing Ranges of Elements

- We can also access a range of elements (a subset of the original vector) by using the colon operator and giving a starting and ending index.

```
>> odd(2:4)
    ans = 4     7     10
```

Get the elements between positions 2 and 4 inclusive from the vector **odd**

- Simple arithmetic between scalars and vectors is possible.

```
>> 10 + [1 2 3]
       ans = 11  12  13
```

Remember we did that with the **ones** matrix?

Adding and Subtracting Vectors

- Addition or subtraction of vectors is possible **as long as they are the same size** ([10, 11] + [1, 2, 3] would not work!).

```
>> [1 2 3] + [4 5 6]
   ans  =  5    7    9
>> x = [1 2 3] - [4 5 6]
   x  =  -3    -3    -3
```

These operations are called elementwise operations or element-by-element operations. It means that the operations are done in parallel for every element of the vectors. That is the reason why the sizes must be identical.

Dividing and Multiplying Vectors

- Element-by-element multiplication and division uses the **.*** and **./** operators. The * and / operators are used for complete matrix multiplication and division which we will not cover in detail in this class as it requires knowledge of linear algebra. The power operator ^ also requires the dot (**.^**) for element-by-element.

```
>> [1 2 3] .* [4 5 6]
>> [6 5 80] ./ [2 2 2]
>> [1 2 3] .^ [2 3 4]
```

Dividing and Multiplying Vectors

- Element-by-element multiplication and division uses the **.*** and **./** operators. The * and / operators are used for complete matrix multiplication and division which we will not cover in detail in this class as it requires knowledge of linear algebra. The power operator ^ also requires the dot (**.^**) for element-by-element.

```
>> [1 2 3] .* [4 5 6]
    ans = 4     10     18
>> [6 5 80] ./ [2 2 2]
    ans = 3.0000     2.5000    40.0000
>> [1 2 3] .^ [2 3 4]
    ans = 1     8    81
```

The * Operator

- The * operator without the dot is used for linear algrbra operations. For this class, you do not need to know how to do those without MATLAB (this is not a math course) but you should definitely know the difference between . And .* Now let's have a look at 3 examples.

Q?

+v1 = [1 2 3 4]; v2 = [5;6;7;8]; sp = v1 * v2;

The * Operator

- The * operator without the dot is used for linear algrbra operations. For this class, you do not need to know how to do those without MATLAB (this is not a math course) but you should definitely know the difference between . And .* Now let's have a look at 3 examples.
- Example 1: The * between two vectors is called the scalar product. The first vector must be a row vector, the second one a column vector. Both must have the same number of elements.

```
v1 = [1 2 3 4]; v2 = [5;6;7;8]; sp = v1 * v2;
sp = 70
```

Q?

- m1 = [2 2 2 2 ; 3 3 3 3];
- v2 = [5;6;7;8];
  mv = m1 * v2

The * Operator

- Example 2: The * between a vector and a matrix. The vector must be a column vector and have the same number of elements as the number of columns in the matrix. In this example both have 4.

```
m1 = [2 2 2 2 ; 3 3 3 3]; v2 = [5;6;7;8];
mv = m1 * v2
mv = 52
     78
```

Q?

- m2 = [2 2 2 ; 3 3 3];
  m3 = [5 5 5 5 ; 4 4 4 4 ; 6 6 6 6];
  mm = m2 * m3

The * Operator

- Example 3: The * between two matrices. This is full fledged matrix multiplication. The number of columns in the first matrix must must be the same as the number of rows in the second matrix. In this example both have 3.

```
m2 = [2 2 2 ; 3 3 3];
m3 = [5 5 5 5 ; 4 4 4 4 ; 6 6 6 6];
mm = m2 * m3
mv = 30 30 30 30
        45 45 45 45
```

Elementwise Operations on Matrices

```
>> [1 2 3 ; 4 5 6] + [0 1 2 ; 6 4 3]
>> [1 2 3 ; 4 5 6] - [0 1 2 ; 6 4 3]
```

Elementwise Operations on Matrices

- Element-by-element can also be applied on matrices. Again, the sizes must be identical.

```
>> [1 2 3 ; 4 5 6] + [0 1 2 ; 6 4 3]
   ans = 1    3    5
         10   9    9
>> [1 2 3 ; 4 5 6] - [0 1 2 ; 6 4 3]
   ans = 1    -1    0
         -2   1     3
>> [1 2 3 ; 4 5 6] .* [0 1 2 ; 6 4 3]
```

Elementwise Operations on Matrices

- Element-by-element can also be applied on matrices. Again, the sizes must be identical.

```
>> [1 2 3 ; 4 5 6] + [0 1 2 ; 6 4 3]
ans = 1    3    5
      10   9    9

>> [1 2 3 ; 4 5 6] - [0 1 2 ; 6 4 3]
ans = 1    -1   0
      -2   1    3

>> [1 2 3 ; 4 5 6] .* [0 1 2 ; 6 4 3]
ans = 0    2    6
      24   20   18
```

```
>> [4 6 6 ; 2 2 2 ; 1 1 1] ./ [1 1 1 ; 2 2 2 ; 1 1 1]
```

Elementwise Operations on Matrices

- Element-by-element can also be applied on matrices. Again, the sizes must be identical.

```
>> [1 2 3 ; 4 5 6] + [0 1 2 ; 6 4 3]

    ans = 1    3    5
          10   9    9

>> [1 2 3 ; 4 5 6] - [0 1 2 ; 6 4 3]

    ans = 1    -1    0
          -2   1     3

>> [1 2 3 ; 4 5 6] .* [0 1 2 ; 6 4 3]

    ans = 0    2    6
          24   20   18
```

```
>> [4 6 6 ; 2 2 2 ; 1 1 1] ./ [1 1 1 ; 2 2 2 ; 1 1 1]
    ans = 4  6  6
          1  1  1
          1  1  1

>> [1 2 ; 3 4] .^ [2 2 ; 3 3]
```

Elementwise Operations on Matrices

- Element-by-element can also be applied on matrices. Again, the sizes must be identical.

```
>> [1 2 3 ; 4 5 6] + [0 1 2 ; 6 4 3]
    ans = 1    3    5
          10   9    9

>> [1 2 3 ; 4 5 6] - [0 1 2 ; 6 4 3]
    ans = 1    -1    0
          -2   1     3

>> [1 2 3 ; 4 5 6] .* [0 1 2 ; 6 4
3]
    ans = 0    2    6
          24   20   18
```

```
>> [4 6 6 ; 2 2 2 ; 1 1 1] ./
[1 1 1 ; 2 2 2 ; 1 1 1]
    ans = 4  6  6
          1  1  1
          1  1  1

>> [1 2 ; 3 4] .^ [2 2 ; 3 3]
    ans =  1   4
           27  64
```

Creating Matrices from Vectors

- You can create new matrices or new vectors by sticking vectors together. Of course the sizes must be compatible in some cases.

```
>> v1 = [1 2 3 4];
>> v2 = [5 6 7 8];
>> m1 = [v1 ; v2]
```

Creating Matrices from Vectors

- You can create new matrices or new vectors by sticking vectors together. Of course the sizes must be compatible in some cases.

```
>> v1 = [1 2 3 4];
>> v2 = [5 6 7 8];
>> m1 = [v1 ; v2]
m1 = 1 2 3 4
     5 6 7 8
>> m2 = [v2 ; v1]
```

Creating Matrices from Vectors

- You can create new matrices or new vectors by sticking vectors together. Of course the sizes must be compatible in some cases.

```
>> v1 = [1 2 3 4];
>> v2 = [5 6 7 8];
>> m1 = [v1 ; v2]
m1 = 1 2 3 4
     5 6 7 8
>> m2 = [v2 ; v1]
m2 = 5 6 7 8
     1 2 3 4
```

```
>> m3 = [v1 ; v1]
```

Creating Matrices from Vectors

- You can create new matrices or new vectors by sticking vectors together. Of course the sizes must be compatible in some cases.

```
>> v1 = [1 2 3 4];
>> v2 = [5 6 7 8];
>> m1 = [v1 ; v2]
m1 = 1 2 3 4
     5 6 7 8
>> m2 = [v2 ; v1]
m2 = 5 6 7 8
     1 2 3 4
```

```
>> m3 = [v1 ; v1]
m1 = 1 2 3 4
     1 2 3 4

>> v3 = [v1 , v2]
```

Creating Matrices from Vectors

- You can create new matrices or new vectors by sticking vectors together. Of course the sizes must be compatible in some cases.

```
>> v1 = [1 2 3 4];

>> v2 = [5 6 7 8];

>> m1 = [v1 ; v2]

m1 = 1 2 3 4

     5 6 7 8

>> m2 = [v2 ; v1]

m2 = 5 6 7 8

     1 2 3 4
```

```
>> m3 = [v1 ; v1]
m1 = 1 2 3 4
     1 2 3 4

>> v3 = [v1 , v2]
v3 = 1 2 3 4 5 6 7 8
```

Appending to Vectors

- You can only append row vectors to row vectors and column vectors to column vectors.
  - If r1 and r2 are any row vectors,
    r3 = [r1 r2] is a row vector whose left part is r1 and right part is r2
  - If c1 and c2 are any column vectors,
    c3 = [c1; c2] is a column vector whose top part is c1 and bottom part is c2

Appending to vectors examples

```
>> v1 = [3 8 1 24];

>> v2 = 4:3:16;

>> v3 = [v1 v2]
```

Appending to vectors examples

```
>> v1 = [3 8 1 24];

>> v2 = 4:3:16;

>> v3 = [v1 v2]

v3 = 3 8 1 24 4 7 10 13 16

>> v4 = [v1'; v2']
```

Appending to vectors examples

```
>> v1 = [3 8 1 24];

>> v2 = 4:3:16;

>> v3 = [v1 v2]

v3 = 3 8 1 24 4 7 10 13 16

>> v4 = [v1'; v2']

v4 =        3
            8
            1
            24
            4
            7
            10
            13
            16
```

You understand what's going on here?

Creating Matrices from Matrices

- You can create new matrices from other matrices. Be careful about the sizes here!

```
>> mat1 = [1 2 ; 3 4]

   mat 1 =    1  2

              3  4

>> mat2 = [mat1 ; mat1]
```

Creating Matrices from Matrices

- You can create new matrices from other matrices. Be careful about the sizes here!

```
>> mat1 = [1 2 ; 3 4]
   mat 1 =   1  2

             3  4
>> mat2 = [mat1 ; mat1]
   mat2 =    1  2
             3  4
             1  2
             3  4
```

```
>> mat3 = [1 2 3 ; 4 5 6];

>> mat4 = [mat1 , mat3]
```

Creating Matrices from Matrices

- You can create new matrices from other matrices. Be careful about the sizes here!

```
>> mat1 = [1 2 ; 3 4]
   mat 1 =    1  2
              3  4

>> mat2 = [mat1 ; mat1]
   mat2 =     1  2
              3  4
              1  2
              3  4
```

```
>> mat3 = [1 2 3 ; 4 5 6];

>> mat4 = [mat1 , mat3]
   mat2 =   1  2  1  2  3
            3  4  4  5  5

>> mat5 = [mat1 ; mat3]
```

Creating Matrices from Matrices

- You can create new matrices from other matrices. Be careful about the sizes here!

```
>> mat1 = [1 2 ; 3 4]
    mat 1 =    1  2

               3  4

>> mat2 = [mat1 ; mat1]
    mat2 =     1  2
               3  4
               1  2
               3  4
```

```
>> mat3 = [1 2 3 ; 4 5 6];

>> mat4 = [mat1 , mat3]
    mat2 =    1  2  1  2  3
              3  4  4  5  5

>> mat5 = [mat1 ; mat3]
    ERROR!
```

Appending to Matrices

- If appending one matrix to the right side of another matrix, both must have same number of rows.
- If appending one matrix to the bottom of another matrix, both must have same number of columns

Appending to Matrices Examples

```
>> A2=[1 2 3; 4 5 6]

A2 = 1     2     3

     4     5     6

>> B2=[7 8; 9 10]

B2 = 7     8

     9    10

>> C2=eye(3)
```

Appending to Matrices Examples

```
>> A2=[1 2 3; 4 5 6]

A2 = 1     2     3

     4     5     6

>> B2=[7 8; 9 10]

B2 = 7     8

     9    10

>> C2=eye(3)

C2 = 1     0     0

     0     1     0

     0     0     1
```

```
>> Z=[A2 B2]
```

Appending to Matrices Examples

```
>> A2=[1 2 3; 4 5 6]

A2 = 1    2    3

     4    5    6

>> B2=[7 8; 9 10]

B2 = 7    8

     9   10

>> C2=eye(3)

C2 = 1    0    0

     0    1    0

     0    0    1
```

```
>> Z=[A2 B2]

Z =  1    2    3    7    8

     4    5    6    9   10

>> Z=[A2; C2]
```

Appending to Matrices Examples

```
>> A2=[1 2 3; 4 5 6]

A2 = 1    2    3

     4    5    6

>> B2=[7 8; 9 10]

B2 = 7    8

     9   10

>> C2=eye(3)

C2 = 1    0    0

     0    1    0

     0    0    1

>> Z=[A2 B2]

Z = 1    2    3    7    8

    4    5    6    9   10

>> Z=[A2; C2]

Z = 1    2    3

    4    5    6

    1    0    0

    0    1    0

    0    0    1

>> Z=[A2; B2]
```

Appending to Matrices Examples

```
>> A2=[1 2 3; 4 5 6]

A2 = 1     2     3

     4     5     6

>> B2=[7 8; 9 10]

B2 = 7     8

     9    10

>> C2=eye(3)

C2 = 1     0     0

     0     1     0

     0     0     1

>> Z=[A2 B2]

Z =  1     2     3     7     8

     4     5     6     9    10

>> Z=[A2; C2]

Z =  1     2     3

     4     5     6

     1     0     0

     0     1     0

     0     0     1

>> Z=[A2; B2]

??? Error using ==> vertcat
CAT arguments dimensions are
not consistent.
```

Trig Commands and Arrays

- When the argument of a trig function is an array, you get an array of the same size for an answer. It is the same for all other math functions as well, like sqrt.

Ex:

```
>> sin ([pi/4  pi/2  pi])
   ans = 0.7071   1.0000   0.0000
```

Saving and Loading Variables

- Use ***save*** to save variables to a file at any time you wish while using MATLAB..

>> save myfile mat1 mat2

saves matrices *mat1* and *mat2* to the file named *myfile.mat*

- To load saved variables back into the workspace, use ***load***.

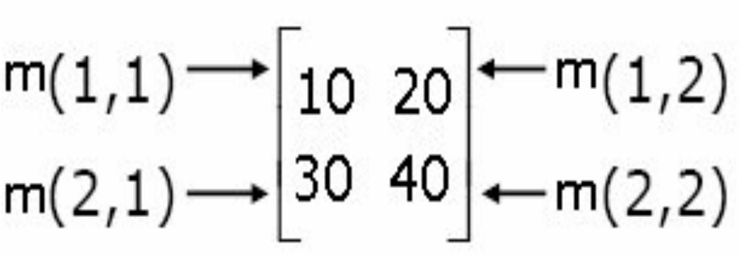
>> load myfile

You can write myfile.mat
but if you don't the .mat is
assumed.

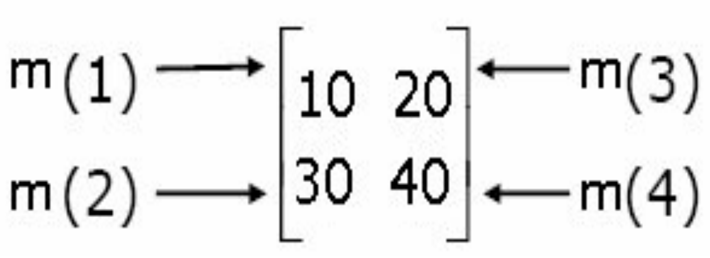
Index vs. Subscript

- You remember that indexes are a way to access elements. Remember that indexes start at 1 in MATLAB. Now let's do that with matrices. For a matrix you can access an element using an index or a pair of subscripts.
- For matrices, subscripts have the row and column numbers separated by a comma. Indexes, on the other hand, indicate the linear position from the start of the matrix. See the figure below for the difference.

subscripts



indexes



Matrix Indexing

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.

```
>> a = [10   20   30   40   50   60];
>> b = a([1 2 3 ; 4 5 6])
```

Matrix Indexing

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.

```
>> a = [10   20   30   40   50   60];
>> b = a([1 2 3 ; 4 5 6])
   b = 10   20  30
          40   50  60
```

This is a matrix of indexes. Get it?

Working Whole Rows or Columns

- The colon operator can select entire rows or columns of a matrix.

```
>> c = b (1,:)
```

```
b = 10   20  30
     40   50  60
```

Working Whole Rows or Columns

- The colon operator can select entire rows or columns of a matrix.

```
>> c = b (1,:)

   c = 10  20  30

>> d = b (:,2)
```

$$b = \begin{matrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{matrix}$$

Working Whole Rows or Columns

- The colon operator can select entire rows or columns of a matrix.

```
>> c = b (1,:)

    c = 10  20  30

>> d = b (:,2)

    d = 20
        50

>> b (:, 3) = [100   200]
```

```
b = 10   20  30
    40   50  60
```

Working Whole Rows or Columns

- The colon operator can select entire rows or columns of a matrix.

```
b = 10   20  30
    40   50  60
```

```
>> c = b (1,:)

   c = 10  20  30

>> d = b (:,2)

   d = 20
       50

>> b (:, 3) = [100   200]

   b = 10   20   100

       40   50   200
```

End of lesson