# Matrix and Vector Operations

Spring 2019

# `length` and `size` functions

- `length` - returns the number of elements in a vector
- `size` - returns the number of rows and columns in a vector or matrix.

# length and size functions

- length - returns the number of elements in a vector
- size - returns the number of rows and columns in a vector or matrix.

```
>> vec = -2:2
vec =
    -2    -1     0     1     2
>> length(vec)
ans =
     5
>> size(vec)
ans =
     1     5
```

# length and size functions - Matrix case

```
>> M = [1:4;5:8]'
M =
     1     5
     2     6
     3     7
     4     8
>> [r,c]=size(M)
r =
     4
c =
     2
>> length(M)
ans =
     4
```

- `size` - returns the number of rows and columns
- `length` - returns the number of rows or columns (whichever is the largest one).

# Matrix and Array Operations

- Matrix operations follow the rules of linear algebra
- Array operations execute element by element operations on elements of vectors, matrices or multi-dimensional arrays.
- The period character `(.)` distinguishes array operations from matrix operations.

| Op | Purpose | Description |
|---|---|---|
| `+` | Addition | `A+B` adds `A` and `B` |
| `+` | Unitary plus | `+A` returns `A` |
| `-` | Subtraction | `A-B` subtracts `B` from `A` |
| `-` | Unitary minus | `-A` negates `A` |
| `*` | product | `A*B` is the usual matrix product |
| `.*` | Elmt-wise multiplication | `A.*B` is elmt-by-elmt product of `A` and `B` |
| `.^` | Elmt-wise multiplication | `A.^ B` has elements `A(i,j)` raised to `B(i,j)` |
| `./` | Right array division | `A./B` has elements `A(i,j)/B(i,j)` |

# Matrix operations

Define `>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5]; d= [0,1];`

- We can add vectors of the same dimension

```
>> a+b
ans =
     4     6     8
```

# Matrix operations

Define `>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5]; d= [0,1];`

- We can add vectors of the same dimension

```
>> a+b
ans =
     4     6     8
```

- If the dimensions are not the same, we get dimension errors

```
>> a+c
Error using +
Matrix dimensions must agree.
>> a+d
Error using +
Matrix dimensions must agree.
```

# Matrix operations

Define `>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- We can scale vectors

```
>> -2*a
ans =
      -2    -4    -6
```

# Matrix operations

Define >> `a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- We can scale vectors

```
>> -2*a
ans =
     -2     -4     -6
```

- We can multiply vectors of appropriate dimensions

```
>> a*c
ans =
25
```

# Element-wise/ Component-wise operations

Define `>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- Component-wise multiplication on vectors of the same dimension

```
>> a.*b
ans =
      3     8    15
```

# Element-wise/ Component-wise operations

Define >> `a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- Component-wise multiplication on vectors of the same dimension

```
>> a.*b
ans =
      3      8     15
```

- If the dimensions are not equal

```
>> a.*c
Error using .*
Matrix dimensions must agree.
```

# Component-wise operations

Define `>> a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- Square every component of a vector

```
>> a.^2
ans =
1     4     9
```

# Component-wise operations

Define >> `a=[1 2 3]; b=[3 4 5]; c=[2;4;5];`

- Square every component of a vector

```
>> a.^2
ans =
1     4     9
```

- $a(i)^{b(i)}$

```
>> a.^b
ans =
1    16   243
```

# Matrix square & Component-wise square

Define >> `A=[1,2;3,4]; B=[0,1;1,0];`

- The square of a matrix i.e $A^2$.

```
>> A^2
ans =
     7    10
    15    22
```

- Component-wise square of $A$ i.e $A(i,j)^2$

```
>> A.^2
ans =
     1     4
     9    16
```

# Component-wise operations

Define >> `A=[1,2;3,4]; B=[0,1;1,0];`

- $A(i,j)^{B(i,j)}$

```
>> A.^B
ans =
     1     2
     3     1
```

# Functions acting on matrices or vectors

- All actions are automatically done component-wise, e.g, given a matrix with random entries on $[0, 1]$

```
>> M=rand(4)
M =
    0.8722    0.9585    0.0591    0.4272
    0.0522    0.7900    0.7409    0.1687
    0.2197    0.4519    0.5068    0.7517
    0.4596    0.3334    0.1999    0.3684
```

- We can round off each entry to create a random binary matrix:

```
>> round(M)
ans =
     1     1     0     0
     0     1     1     0
     0     0     1     1
     0     0     0     0
```

# Functions acting on Matrices or vectors

- Define $M$ as

```
>> M=rand(4)
M =
     0.8722    0.9585    0.0591    0.4272
     0.0522    0.7900    0.7409    0.1687
     0.2197    0.4519    0.5068    0.7517
     0.4596    0.3334    0.1999    0.3684
```

- Compute $e^M$

```
>> exp(M)
ans =
     2.3923    2.6079    1.0609    1.5329
     1.0536    2.2035    2.0978    1.1838
     1.2457    1.5713    1.6600    2.1206
     1.5835    1.3957    1.2213    1.4453
```

# Functions acting on Matrices or vectors: Exercise

- Create a vector of alternating 1s and 0s

# Functions acting on Matrices or vectors: Exercise

- Create a vector of alternating 1s and 0s

```
>> vec =1:10
vec =
1       2       3       4       5       6       7       8       9       10
>> mod(vec,2)
ans =
1     0       1       0       1       0       1       0       1       0
```

# Functions acting on Matrices or vectors: Exercise

- What about random bits?

# Functions acting on Matrices or vectors: Exercise

- What about random bits?

```
>> vec = round(rand(1,10)*10)
vec =
8     9     1     9     6     1     3     5     10    10
>> mod(vec,2)
ans =
0     1     1     1     0     1     1     1     0     0
```

# reshape

The MATLAB functions `reshape`, `fliplr`, `flipud` and `rot90` can change the dimensions or configuration of matrices.

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95    63    73     2
     2    54    10    30
    83    66    88    18
```

- Reshape to $2 \times 6$ (`reshape` - iterates through $M$ column-wise)

```
>> reshape(M,2,6)
ans =
95    83    54    73    88    30
 2    63    66    10     2    18
```

# fliplr

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95    63    73     2
     2    54    10    30
    83    66    88    18
```

- fliplr - "flips" the matrix from left to right

```
>> fliplr(M)
ans =
     2    73    63    95
    30    10    54     2
    18    88    66    83
```

# flipup

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95    63    73     2
     2    54    10    30
    83    66    88    18
```

- `flipup` - "flips" the matrix from up to down

```
>> flipud(M)
ans =
    83    66    88    18
     2    54    10    30
    95    63    73     2
```

# rot90

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95    63    73     2
     2    54    10    30
    83    66    88    18
```

- rot90 - counterclockwise rotation of 90 degrees

```
>> rot90(M)
ans =
     2    30    18
    73    10    88
    63    54    66
    95     2    83
```

# repmat

- e.g define $M$ - a matrix of 12 random integers on $[0, 100]$.

```
>> M=randi(100,3,4)
M =
    95    63    73     2
     2    54    10    30
    83    66    88    18
```

- repmat will duplicate a matrix, e.g.

```
>> repmat(M,2,2)
ans =
    95    63    73     2    95    63    73     2
     2    54    10    30     2    54    10    30
    83    66    88    18    83    66    88    18
    95    63    73     2    95    63    73     2
     2    54    10    30     2    54    10    30
    83    66    88    18    83    66    88    18
```

# Three-dimensional matrices

Think about printing $2D$ matrices on sheets of paper and stacking them.

- Create $M$ with entries $1 - 20$ as

```
>> M=reshape(1:20,4,5)
M =
      1     5     9    13    17
      2     6    10    14    18
      3     7    11    15    19
      4     8    12    16    20
```

- Add a second matrix on top on $M$ as `>> M(:,:,2)= fliplr(M);`
- Check the size of $M$

```
>> size(M)
ans =
     4     5     2
```

# Three-dimensional matrices

- Check the contents of the matrix $M$:

```
>> M
M(:,:,1) =
     1     5     9    13    17
     2     6    10    14    18
     3     7    11    15    19
     4     8    12    16    20
M(:,:,2) =
    17    13     9     5     1
    18    14    10     6     2
    19    15    11     7     3
    20    16    12     8     4
```

# 3D matrices and images – RGB color model

**RGB model**

- Create an array of colors by combining various ratios of red, green and blue.
- The RGB values are integers in $[0, 255]$.
- We can store the R, B, G values in the form of a 3D matrix where each layer corresponds to a color channel.