Built-in Functions

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

AE121: Computational Method

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-05-28

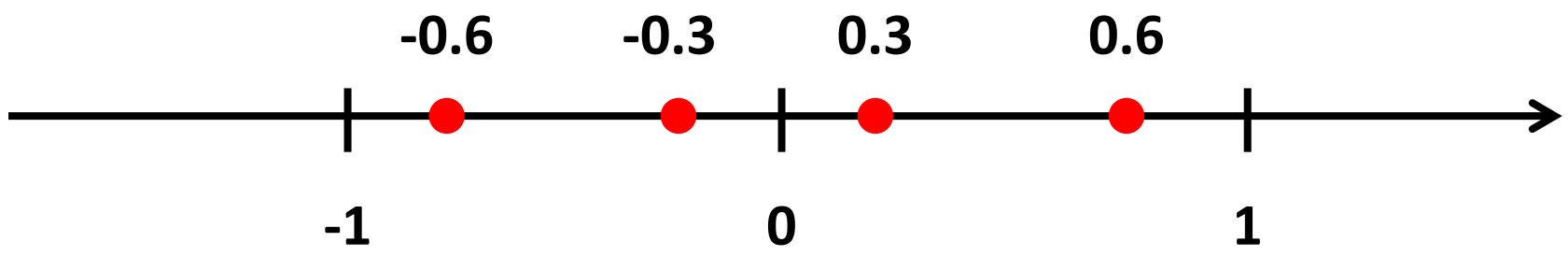
Rounding Functions

`round(x)`	Rounds **x** to the nearest integer.	`round(8.6)`

`fix(x)`	Truncates **x** to the nearest integer toward zero. Notice that 8.6 truncates to 8, not 9, with this function.	`fix(8.6)`
		`ans = 8`
		`fix(-8.6)`
		`ans = -8`

`floor(x)`	Rounds **x** to the nearest integer toward negative infinity.	`floor(-8.6)` `ans = -9`
`ceil(x)`	Rounds **x** to the nearest integer toward positive infinity.	`ceil(-8.6)` `ans = -8`

Example: Rounding Functions



	round	ceil	fix	floor
0.3	0	1	0	0
0.6	1	1	0	0
-0.3	0	0	0	-1
-0.6	-1	0	0	-1

`round(x)` Rounds **x** to the nearest integer.

`fix(x)` Rounds (or truncates) **x** to the nearest integer toward zero. Notice that 8.6 truncates to 8, not 9, with this function.

`floor(x)` Rounds **x** to the nearest integer toward negative infinity.

`ceil(x)` Rounds **x** to the nearest integer toward positive infinity.

‘Round’ in MATLAB

round

Round to nearest decimal or integer

Syntax

```
Y = round(X)
Y = round(X,N)
Y = round(X,N,type)

Y = round(t)
Y = round(t,unit)
```

Description

`Y = round(X)` rounds each element of `X` to the nearest integer. In the case of a tie, where an element has a fractional part of exactly `0.5`, the `round` function rounds away from zero to the integer with larger magnitude.

Input Arguments

collapse all

X — Input array
scalar | vector | matrix | multidimensional array

Input array, specified as a scalar, vector, matrix, or multidimensional array. For complex `X`, `round` treats the real and imaginary parts independently.

`X` must be `single` or `double` when you use `round` with more than one input.

`round` converts logical and `char` elements of `X` into `double` values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `logical`
Complex Number Support: Yes

```
>> vec = [1.2 2.5 3.67 4.1];
>> round(vec)

ans =

     1     3     4     4
```

https://www.mathworks.com/help/matlab/ref/round.html#buftmpz-X

Common Math Functions

`abs(x)`	Finds the absolute value of **x**.	`abs(-3)` `ans = 3`
`sqrt(x)`	Finds the square root of **x**.	`sqrt(85)` `ans = 9.2195`
`sign(x)`	Return –1 is x is less than zero, a value of 0 if x equals zero, and a value of +1 if x is greater than zero	`sign(-8)` `ans = -1`
`rem(x,y)`	Computes the remainder of **x/y**.	`rem(25,4)` `ans = 1`
`exp(x)`	Computes the value of e^x, where e is the base for natural logarithms, approximately 2.7183.	`exp(10)` `ans = 2.2026e+004`
`log(x)`	Computes ln(**x**), the natural logarithm of **x** (to the base e).	`log(10)` `ans = 2.3026`
`log10(x)`	Computes $\log_{10}$(**x**), the common logarithm of **x** (to the base 10).	`log10(10)` `ans = 1`

Example: Common Math Functions

```
val_pos = 3;
val_neg = -5;

val1 = abs(val_pos);
val2 = sign(val_pos) * val_pos; % same result

sign_val1 = -(val_pos<0) + (val_pos>0);
val3 = sign_val1 * val_pos;

val4 = abs(val_neg);
val5 = sign(val_neg) * val_neg;

sign_val2 = -(val_neg<0) + (val_neg>0);
val6 = sign_val2 * val_neg;
```

```
val1 = 3
val2 = 3
val3 = 3
val4 = 5
val5 = 5
val6 = 5
```

Example: Common Math Functions (Continue)

```
val_pos = 10;
val_neg = -10;
b = 4;

rem1 = rem(val_pos, b);
rem2 = val_pos - b*fix(val_pos/b); % doc rem
rem3 = rem(val_neg, b);
rem4 = val_neg - b*fix(val_neg/b);
```

2.5

```
rem1 = 2
rem2 = 2
rem3 = -2
rem4 = -2
```

`fix(x)` Rounds (or truncates) **x** to the nearest integer toward zero. Notice that 8.6 truncates to 8, not 9, with this function.

Quiz

```
vec1 = randi(100, 1, 30);
```

You need to write a script that extract all odd numbers in 'vec1' and assign a resulting vector to 'result1'.

For example, if 'vec1' is [1 2 4 5], 'result' becomes [1 5].

Function Used in Discrete Mathematics

`factor(x)`	Finds the prime factors of **x**.	`factor(12)` `ans = 2 2 3`
`gcd(x,y)`	Finds the greatest common denominator of **x** and **y**.	`gcd(10,15)` `ans = 5`

`lcm(x,y)`	Finds the least common multiple of **x** and **y**.	`lcm(2,5)`
		`ans = 10`
		`lcm(2,10)`
		`ans = 10`

`factorial(x)`	Finds the value of **x** factorial (**x**!). A factorial is the product of all the integers less than **x**. For example, $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$	`factorial(6)` `ans = 720`

Example: Function Used in Discrete Mathematics

```
rng(110)

val1 = 120;
f = factor(val1)
val2 = prod(f)

num1 = randi(100)
num2 = randi(100)

val3 = gcd(num1, num2) % the greatest common denominator
rem_num1_val3 = rem(num1, val3)
rem_num2_val3 = rem(num1, val3)

val4 = lcm(num1, num2) % the least common denominator
rem_val4_num1 = rem(val4, num1)
rem_val4_num2 = rem(val4, num2)
```

```
f = 1×5
     2     2     2     3     5

val2 = 120

num1 = 12
num2 = 66

val3 = 6
rem_num1_val3 = 0
rem_num2_val3 = 0

val4 = 132
rem_val4_num1 = 0
rem_val4_num2 = 0
```

Some of the Available Trigonometric Functions

`deg2rad`	Converts degrees to radians.	`deg2rad(90)` `ans = 1.5708`
`rad2deg`	Converts radians to degrees.	`rad2deg(pi)` `ans = 180`
`sin(x)`	Finds the sine of **x** when **x** is expressed in radians.	`sin(0)` `ans = 0`
`cos(x)`	Finds the cosine of **x** when **x** is expressed in radians.	`cos(pi)` `ans = −1`
`tan(x)`	Finds the tangent of **x** when **x** is expressed in radians.	`tan(pi)` `ans = −1.2246e−016`
`asin(x)`	Finds the arcsine, or inverse sine, of **x**, where **x** must be between −1 and 1. The function returns an angle in radians between $\pi/2$ and $-\pi/2$.	`asin(−1)` `ans = −1.5708`
`sind(x)`	Finds the sin of **x** when **x** is expressed in degrees.	`sind(90)` `ans = 1`
`asind(x)`	Finds the inverse sin of **x** and reports the result in degrees.	`asind(1)` `ans = 90`

Example: Trigonometric Functions

```
% You always check if your input angle is expressed in radians or degrees.

ang_rad = pi/6;
ang_deg = rad2deg(ang_rad);

val1 = sin(ang_rad); % sine (radian)
val2 = sind(ang_deg); % sine (degree)
val3 = cos(ang_rad); % cosine (radian)
val4 = cosd(ang_deg) % cosine (degree)
val5 = tan(ang_rad); % tagent (radian)

val6 = val1/val3; % relation for tangent
val7 = val1^2  + val3^2; % relation for cosine and sine
```

```
val4 =
   0.866025403784439
val1 =
   0.500000000000000
val2 =
   0.500000000000000
val3 =
   0.866025403784439
val4 =
   0.866025403784439
val5 =
   0.577350269189626
val6 =
   0.577350269189626
val7 =
     1
```

Example: Trigonometric Functions (Continue)

```
ang_deg = rad2deg(pi/6);
val1 = sind(ang_deg); % sine (degree)
val2 = asind(val1); % inverse sine
val3 = cosd(ang_deg); % cosine (degree)
val4 = acosd(val3); % inverse cosine
```

```
ang_deg =
   29.999999999999996
val1 =
    0.500000000000000
val2 =
   29.999999999999996
val3 =
    0.866025403784439
val4 =
   29.999999999999993
```

Why isn't 30 degrees?

Pi is irrational and has infinite number of digits in its decimal representation

is_true = (ang_deg == 30) **not working**

is_true = (round(ang_deg) == 30) **working**

is_true = (ang_deg - 30) < 10^-5; **recommendation**

Please do not directly check the equality between numbers in double precision.

Example: Computational Limit

```
% precision
val6 = 0.3-0.2-0.1;
val7 = 0;
is_equal_val6_val7 = (val6==val7);

val8 = tand(30)-sind(30)/cosd(30);
val9 = 0;
is_equal_val8_val9 = (val8==val9);

tol = eps * 10^4; % small number
is_equal_val6_val7_tol = (abs(val6-val7)<tol);
is_equal_val8_val9_tol = (abs(val8-val9)<tol);
```

```
val6 =
    -2.775557561562891e-17
val7 =
     0
val8 =
     1.110223024625157e-16
val9 =
     0
```

```
is_equal_val6_val7 = logical
   0
is_equal_val6_val7_tol = logical
   1
is_equal_val8_val9 = logical
   0
is_equal_val8_val9_tol = logical
   1
```

Recommendation: Do not use '==' when you compare numbers in a double precision

Sum in MATLAB

sum

R2019a

Sum of array elements

collapse all in page

Syntax

```
S = sum(A)
S = sum(A,'all')
S = sum(A,dim)
S = sum(A,vecdim)
S = sum( __ ,outtype)
S = sum( __ ,nanflag)
```

Description

example

`S = sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1.

- If `A` is a vector, then `sum(A)` returns the sum of the elements.
- If `A` is a matrix, then `sum(A)` returns a row vector containing the sum of each column.

example

`S = sum(A,'all')` computes the sum of all elements of `A`. This syntax is valid for MATLAB® versions R2018b and later.

example

`S = sum(A,dim)` returns the sum along dimension `dim`. For example, if `A` is a matrix, then `sum(A,2)` is a column vector containing the sum of each row.

Sum in MATLAB (Continue)

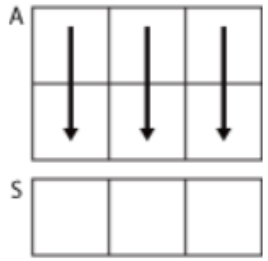
`dim` — Dimension to operate along
positive integer scalar

Dimension to operate along, specified as a positive integer scalar. If no value is specified, then the default is the first array dimension whose size does not equal 1.

Dimension `dim` indicates the dimension whose length reduces to 1. The `size(S,dim)` is `1`, while the sizes of all other dimensions remain the same.

Consider a two-dimensional input array, `A`:

- `sum(A,1)` operates on successive elements in the columns of `A` and returns a row vector of the sums of each column.



sum(A,1)

- `sum(A,2)` operates on successive elements in the rows of `A` and returns a column vector of the sums of each row.

sum(A,2)

`sum` returns `A` when `dim` is greater than `ndims(A)` or when `size(A,dim)` is `1`.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

1	2	3
4	5	6
7	8	9

12	15	18

1	2	3
4	5	6
7	8	9

6
15
24

Sums and Products

`sum(x)`

Sums the elements in **vector x**. For example, if $x = [1\ 5\ 3]$, the sum is 9.

```
x = [1, 5, 3];
sum(x)
ans = 9
```

Computes a row vector containing the sum of the elements in each column of a **matrix x**. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, then the sum of column 1 is 3, the sum of column 2 is 9, and the sum of column 3 is 9.

```
x = [1, 5, 3; 2, 4, 6];
sum(x)
ans = 3 9 9
```

`prod(x)`

Computes the product of the elements of a **vector x**. For example, if $x = [1\ 5\ 3]$, the product is 15.

```
x = [1, 5, 3];
prod(x)
ans = 15
```

Computes a row vector containing the product of the elements in each column of a **matrix x**. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, then the product of column 1 is 2, the product of column 2 is 20, and the product of column 3 is 18.

```
x = [1, 5, 3; 2, 4, 6];
prod(x)
ans = 2 20 18
```

Example: Sum and Product

```
vec1= 1:6;
val1= vec1(1) + vec1(2) + vec1(3) ...
    + vec1(4) + vec1(5) + vec1(6);
val2= sum(vec1);

val3= vec1(1) * vec1(2) * vec1(3)...
    * vec1(4) * vec1(5) + vec1(6);
val4 = prod(vec1);

mat1 = reshape(vec1, 2, 3);
val5 = sum(mat1);
val6 = sum(mat1,1);
val7 = sum(mat1,2);
val8 = sum(mat1,'all');
val9 = sum(mat1(:));
```

```
vec1 = 1×6
     1     2     3     4     5     6

mat1 = 2×3
     1     3     5
     2     4     6
```

1	3	5
2	4	6

```
val1 = 21
val2 = 21
val3 = 126
val4 = 720
val5 = 1×3
     3     7    11

val6 = 1×3
     3     7    11

val7 = 2×1
     9
    12

val8 = 21
val9 = 21
```

Averages

mean(x)

Computes the mean value (or average value) of a **vector x**. For example, if $x = [1\ 5\ 3]$, the mean value is 3.

```
x = [1, 5, 3];
mean(x)
ans = 3.0000
```

Returns a row vector containing the mean value from each column of a **matrix x**. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, then the mean value of column 1 is 1.5, the mean value of column 2 is 4.5, and the mean value of column 3 is 4.5.

```
x = [1, 5, 3; 2, 4, 6];
mean(x)
ans = 1.5   4.5   4.5
```

median(x)

Finds the median of the elements of a **vector x**. For example, if $x = [1\ 5\ 3]$, the median value is 3.

```
x = [1, 5, 3];
median(x)
ans = 3
```

Returns a row vector containing the median value from each column of a **matrix x**. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \\ 3 & 8 & 4 \end{bmatrix}$, then the median value from column 1 is 2, the median value from column 2 is 5, and the median value from column 3 is 4.

```
x = [1, 5, 3;
2, 4, 6;
3, 8, 4];
median(x)
ans = 2   5   4
```

mode(x)

Finds the value that occurs most often in an array. Thus, for the array $\underline{x} = [1, 2, 3, 3]$ the mode is 3.

```
x = [1, 2, 3, 3]
mode(x)
ans = 3
```

```
>> median([1 3 7])

ans =

     3

>> median([1 3 7 11])

ans =

     5
```

Example: Averages

```
rng(200)
vec1 = randi(5,1,20)
mat1 = reshape(vec1, 4, 5)

% mean
val11 = mean(vec1)
val12 = mean(mat1)
val13 = mean(mat1,1)
val14 = mean(mat1,2)
val15 = mean(mat1,'all')
val16 = mean(mat1(:))

% median
val21 = median(vec1)
val22 = median(mat1)
val23 = median(mat1,1)
val24 = median(mat1,2)
val25 = median(mat1,'all')
val26 = median(mat1(:))

% mode
val31 = mode(vec1)
val32 = mode(mat1)
val33 = mode(mat1,1)
val34 = mode(mat1,2)
val35 = mode(mat1,'all')
val36 = mode(mat1(:))
```

```
vec1 = 1×20
     5     2     3     3     4     1     2     5     3     5     5     5     5     2     5     1     4     2     1     5

mat1 = 4×5
     5     4     3     5     4
     2     1     5     2     2
     3     2     5     5     1
     3     5     5     1     5
```

```
val11 = 3.4000
val12 = 1×5
    3.2500    3.0000    4.5000    3.2500    3.0000

val13 = 1×5
    3.2500    3.0000    4.5000    3.2500    3.0000

val14 = 4×1
    4.2000
    2.4000
    3.2000
    3.8000

val15 = 3.4000
val16 = 3.4000
```

5	4	3	5	4
2	1	5	2	2
3	2	5	5	1
3	5	5	1	5

Example: Averages (Continue)

```
rng(200)
vec1 = randi(5,1,20)
mat1 = reshape(vec1, 4, 5)

% mean
val11 = mean(vec1)
val12 = mean(mat1)
val13 = mean(mat1,1)
val14 = mean(mat1,2)
val15 = mean(mat1,'all')
val16 = mean(mat1(:))

% median
val21 = median(vec1)
val22 = median(mat1)
val23 = median(mat1,1)
val24 = median(mat1,2)
val25 = median(mat1,'all')
val26 = median(mat1(:))

% mode
val31 = mode(vec1)
val32 = mode(mat1)
val33 = mode(mat1,1)
val34 = mode(mat1,2)
val35 = mode(mat1,'all')
val36 = mode(mat1(:))
```

```
vec1 = 1×20
     5     2     3     3     4     1     2     5     3     5     5     5     5     2     5     1     4     2     1     5

mat1 = 4×5
     5     4     3     5     4
     2     1     5     2     2
     3     2     5     5     1
     3     5     5     1     5
```

```
val21 = 3.5000
val22 = 1×5
    3.0000    3.0000    5.0000    3.5000    3.0000

val23 = 1×5
    3.0000    3.0000    5.0000    3.5000    3.0000

val24 = 4×1
     4
     2
     3
     5

val25 = 3.5000
val26 = 3.5000
```

5	4	3	5	4
2	1	5	2	2
3	2	5	5	1
3	5	5	1	5

Example: Averages (Continue)

```
rng(200)
vec1 = randi(5,1,20)
mat1 = reshape(vec1, 4, 5)

% mode
val31 = mode(vec1)
val32 = mode(mat1)
val33 = mode(mat1,1)
val34 = mode(mat1,2)
val35 = mode(mat1,'all')
val36 = mode(mat1(:))
```

```
vec1 = 1×20
     5     2     3     3     4     1     2     5     3     5     5     5     5     2     5     1     4     2     1     5

mat1 = 4×5
     5     4     3     5     4
     2     1     5     2     2
     3     2     5     5     1
     3     5     5     1     5
```

```
val31 = 5
val32 = 1×5
     3     1     5     5     1

val33 = 1×5
     3     1     5     5     1

val34 = 4×1
     4
     2
     5
     5

val35 = 5
val36 = 5
```

5	4	3	5	4
2	1	5	2	2
3	2	5	5	1
3	5	5	1	5

Maxima and Minima

max(x)

Finds the largest value in a **vector x**. For example, if $x = [1\ 5\ 3]$, the maximum value is 5.

```
x = [1, 5, 3];
max(x)
ans = 5
```

Creates a row vector containing the maximum element from each column "of a **matrix x**. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, then the maximum value in column 1 is 2, the maximum value in column 2 is 5, and the maximum value in column 3 is 6.

```
x = [1, 5, 3; 2, 4, 6];
max(x)
ans = 2    5    6
```

[a,b] = max(x)

Finds both the largest value in a **vector x** and its location in vector **x**. For $x = [1\ \ 5\ \ 3]$ the maximum value is named **a** and is equal to 5. The location of the maximum value is element 2 and is named **b**.

```
x = [1, 5, 3];
[a,b] = max(x)
a = 5
b = 2
```

Creates a row vector containing the maximum element from each column of a matrix **x** and returns a row vector with the location of the maximum in each column of matrix **x**. For example, if $= \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, then the maximum value in column 1 is 2, the maximum value in column 2 is 5, and the maximum value in column 3 is 6. These maxima occur in row 2, row 1, and row 2, respectively.

```
x = [1, 5, 3; 2, 4, 6];
[a,b] = max(x)
a = 2    5    6
b = 2    1    2
```

Maxima and Minima (Continue)

max(x,y)

Creates a matrix the same size as **x** and **y**. (Both **x** and **y** must have the same number of rows and columns.) Each element in the resulting matrix contains the maximum value from the corresponding positions in **x** and **y**. For example,

if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ and $y = \begin{bmatrix} 10 & 2 & 4 \\ 1 & 8 & 7 \end{bmatrix}$, then the resulting matrix will be $x = \begin{bmatrix} 10 & 5 & 4 \\ 2 & 8 & 7 \end{bmatrix}$.

```
x = [1, 5, 3; 2, 4, 6];
y = [10,2,4; 1, 8, 7];
max(x,y)
ans = 10  5  4
      2  8  7
```

min(x)

The syntax for the `min` function is the same as that for the `max` function, except that minimums are returned.

Example: Maxima and Minima

```
vec1 = [1:22 24 23];

val1 = max(vec1);
[val2, val2_loc] = max(vec1);
val3 = max(10,20); % compare only 10 and 20
val4 = max(1:10, 8); % compare 1:10 to 8
```

```
val1 = 24
val2 = 24
val2_loc = 23
val3 = 20
val4 = 1×10
       8     8     8     8     8     8     8     8     9    10
```

Example: Maxima and Minima (Continue)

```
mat1 = reshape(vec1, 4, 6);
[val5, loc_val5] = max(mat1); % find
[val6, loc_val6] = max(mat1, [], 1);
[val7, loc_val7] = max(mat1, [], 2);
val8 = max(mat1, [], 'all'); % not s
[val9, loc_val9] = max(mat1(:));
```

```
mat1 = 4×6
     1     5     9    13    17    21
     2     6    10    14    18    22
     3     7    11    15    19    24
     4     8    12    16    20    23
```

```
val5 = 1×6
     4     8    12    16    20    24

loc_val5 = 1×6
     4     4     4     4     4     3

val6 = 1×6
     4     8    12    16    20    24

loc_val6 = 1×6
     4     4     4     4     4     3

val7 = 4×1
    21
    22
    24
    23

loc_val7 = 4×1
     6
     6
     6
     6

val8 = 24
val9 = 24

loc_val9 = 23
```

Quiz: Rounding Error Analysis

```
mat1 = (rand(7,4)- 0.5)*200;
mat1 = round(mat1,4); % round values to the nearest 4 decimal digits
```

You need to write a script that creates a variable named:

(a) 'mat2', which is copied from 'mat1'. The first coumn of 'mat2' is rounded towards negative infinity. 'mat2' is a 7 x 4 matrix.

(b) 'mat3', which is copied from 'mat2'. The second column of 'mat3' is then rounded to the nearest integer toward zero. 'mat3' is a 7 x 4 matrix.

(c) 'mat4', which is copied from 'mat3'. The third column of 'mat4' is then rounded to positive infinity. 'mat4' is a 7 x 4 matrix.

(d) 'round_error', which contains the absolute value of the difference between elements in 'mat1' and 'mat4'. 'round_error' is a 7 x 4 matrix.

For example, 'mat1' and 'mat4' are [1.1 -2.1;-1.1 2.2] and [1 -2;-1 2], 'round_error' becomes [0.1 0.1;0.1 0.2].

(e) 'error_means', which determines the mean of each row of 'round_error'. 'error_means' is a 7 x 1 vector.

(f) 'error_sum', which determines the sum of all values in 'round_error'. 'error_sum' is a scalar.

(g) 'error_max', which determines the maximum value in each row of 'round_error'. 'error_max' is a 7 x 1 row vector.

(h) 'error_min', which determines the miniimum value in each column of 'round_error'. 'error_min' is a 4 x 1 row vector.

Question

```
mat_test = [1 2 3; 7 8 9;4 5 6];
[a, b] = max(mat_test);
[c, d] = max(a);
[b(d) d]  % what does this value indicate?
```

Sorting Functions

`sort(x)`

Sorts the elements of a vector **x** in ascending order. For example, if $x = [1 \quad 5 \quad 3]$, the resulting vector is $x = [1 \quad 3 \quad 5]$.

```
x = [1,5,3];
sort(x)
ans = 1 3 5
```

Sorts the elements in each column of a matrix **x** in ascending order. For example, if $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, the resulting matrix is

$$x = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 5 & 6 \end{bmatrix}$$

```
x = [1,5,3; 2,4,6];
sort(x)
ans = 1 4 3
  2 5 6
```

First dimension => column

`sort(x,'descend')`

Sorts the elements in each column in descending order.

$$x = \begin{bmatrix} 2 & 5 & 6 \\ 1 & 4 & 3 \end{bmatrix}$$

```
x = [1,5,3; 2,4,6];
sort(x,'descend')
ans = 2 5 6
  1 4 3
```

Sorting Functions (Continue)

`sortrows(x)`

Sorts the rows in a matrix in ascending order on the basis of the values in the first column, and keeps each row intact.

$$x = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 9 & 3 \\ 4 & 3 & 6 \end{bmatrix} \qquad x = \begin{bmatrix} 1 & 9 & 3 \\ 3 & 1 & 2 \\ 4 & 3 & 6 \end{bmatrix}$$

```
x = [3,1,2; 1,9,3;
4, 3, 6]
sortrows(x)
ans = 1 9 3
  3 1 2
  4 3 6
```

`sortrows(x,n)`

Sorts the rows in a matrix on the basis of the values in column *n*. If *n* is negative, the values are sorted in descending order. If *n* is not specified, the default column used as the basis for sorting is column 1.

$$x = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 9 & 3 \\ 4 & 3 & 6 \end{bmatrix} \qquad x = \begin{bmatrix} 3 & 1 & 2 \\ 4 & 3 & 6 \\ 1 & 9 & 3 \end{bmatrix}$$

```
sortrows(x,2)
ans = 3 1 2
  4 3 6
  1 9 3
```

Example: sort

```
rng(10);

num = 15;
vec1 = 1:num;
rand_idx = randperm(num,num);
vec1 = vec1(rand_idx);

out01 = sort(vec1, 'ascend');
out02 = sort(vec1, 'descend');
```

```
vec1 = 1×15
    14    10     6     5     1     4     9    15    12     8     3     2    13    11     7

out01 = 1×15
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15

out02 = 1×15
    15    14    13    12    11    10     9     8     7     6     5     4     3     2     1
```

```
mat1 = reshape(vec1, 3, 5);
out10 = sort(mat1, 'ascend');
out11 = sort(mat1, 'descend');
out12 = sort(mat1, 1, 'ascend');
out13 = sort(mat1, 2, 'ascend');
```

```
mat1 = 3×5
    14     5     9     8    13
    10     1    15     3    11
     6     4    12     2     7
```

```
out10 = 3×5
     6     1     9     2     7
    10     4    12     3    11
    14     5    15     8    13

out11 = 3×5
    14     5    15     8    13
    10     4    12     3    11
     6     1     9     2     7

out12 = 3×5
     6     1     9     2     7
    10     4    12     3    11
    14     5    15     8    13

out13 = 3×5
     5     8     9    13    14
     1     3    10    11    15
     2     4     6     7    12
```

Example: sortrows

```
rng(10);

num = 15;
vec1 = 1:num;
rand_idx = randperm(num,num);
vec1 = vec1(rand_idx);
mat1 = reshape(vec1, 3, 5);

mat2 = sortrows(mat1);
```

```
mat4 = sortrows(mat1, 3);

col3 = mat1(:,3);
[~, col3_idx] = sort(col3);
mat5 = mat1(col3_idx,:);
```

```
mat1 = 3×5
    14     5     9     8    13
    10     1    15     3    11
     6     4    12     2     7
```

```
mat2 = 3×5
     6     4    12     2     7
    10     1    15     3    11
    14     5     9     8    13
```

```
mat4 = 3×5
    14     5     9     8    13
     6     4    12     2     7
    10     1    15     3    11
```

```
mat5 = 3×5
    14     5     9     8    13
     6     4    12     2     7
    10     1    15     3    11
```

Quiz: Bulls and Cows

Bulls and Cows is a mind game played by two players. In the game, a random, 4-digit number is chosen and it's values are compared to those of another trial number. **All four digits of the number are different**. If any digit in the chosen number is the exact same value and in the exact same position as any digit in the trial number, this is called a bull. If the digit is present in both the trial number and chosen number, but is not in the same location, this is called a cow.

Quiz: Bulls and Cows (Continue)

```
clear; clc;
true_num = randperm(10,4)-1;
test_num1 = randperm(10,4)-1;
test_num2 = randperm(10,4)-1;
%--------------------------------------------------------------------------
% student code


%--------------------------------------------------------------------------
str_true_num = sprintf('%d', true_num); % Converts true_num to a string
fprintf('True number : %s\n', str_true_num) % Prints string form of true_num

str_test_num1 = sprintf('%d', test_num1);
fprintf('Test number1: %s \t %dB  %dC\n', str_test_num1, bull1, cow1) % Prints string form of test_num1 and the number of bulls and cows

str_test_num2 = sprintf('%d', test_num2);
fprintf('Test number1: %s \t %dB  %dC\n', str_test_num2, bull2, cow2) % Prints string form of test_num1 and the number of bulls and cows

```

```
true_num = 1×4
     8     2     3     7

test_num1 = 1×4
     8     5     7     6
```

```
True number : 8237

Test number1: 8576      1B  1C
```

Functions Used with Complex Numbers

`abs(x)`

Computes the absolute value of a complex number, using the Pythagorean theorem. This is equivalent to the radius if the complex number is represented in polar coordinates.

```
x = 3 + 4i;
abs(x)
ans =
     5
```

`angle(x)`

For example, if $x = 3 + 4i$, the absolute value is $\sqrt{3^2 + 4^2} = 5$.

Computes the angle from the horizontal in radians when a complex number is represented in polar coordinates.

```
x = 3 + 4i;
angle(x)
ans =
    0.9273
```

`complex(x,y)`

Generates a complex number with a real component x and an imaginary component y.

```
x = 3;
y = 4;
complex(x,y)
ans =
    3.0000+
    4.0000i
```

Functions Used with Complex Numbers (Continue)

`real(x)`	Extracts the real component from a complex number.	`x = 3 + 4i;`
		`real(x)`
		`ans =`
		`3`

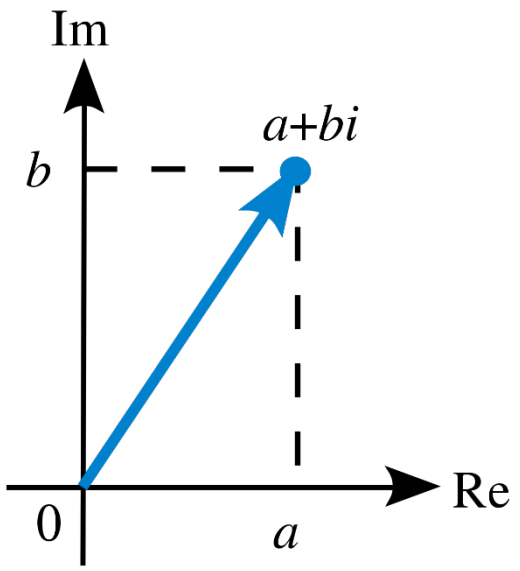
`imag(x)`	Extracts the imaginary component from a complex number.	`x = 3 + 4i;`
		`imag(x)`
		`ans =`
		`4`

`isreal(x)`	Determines whether the values in an array are real. If they are real, the function returns a 1; if they are complex, it returns a 0.	`x = 3 = 4i;`
		`isreal(x)`
		`ans =`
		`0`

`conj(x)`	Generates the complex conjugate of a complex number.	`x = 3 + 4i;`
		`conj(x)`
		`ans =`
		`3.0000 -`
		`4.0000i`

Example: Complex Numbers

```
val0 = 3 + 4i
val01 = 3 + 4*sqrt(-1)
val02 = complex(3, 4)
val03 = (i == sqrt(-1))

val04 = angle(val0)
val05 = atan(imag(val0)/real(val0))
```



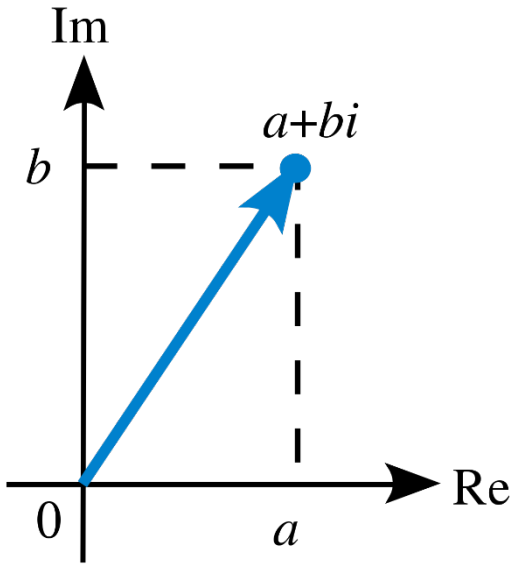

```
val0 = 3.0000 + 4.0000i

val01 = 3.0000 + 4.0000i
val02 = 3.0000 + 4.0000i
val03 = logical
   1
```

```
val04 = 0.9273
val05 = 0.9273
```

Example: Complex Numbers (Continue)

```
val0 = 3 + 4i
val01 = 3 + 4*sqrt(-1)
```

```
val10 = conj(val0)
val11 = complex(real(val0), -imag(val0))
val12 = sqrt(val0*val10)
val13 = abs(val0)
```




```
val10 = 3.0000 - 4.0000i
val11 = 3.0000 - 4.0000i
val12 = 5
val13 = 5
```

Computational Limits

`realmax`	Returns the largest possible floating-point number used in MATLAB®.
`realmin`	Returns the smallest possible floating-point number used in MATLAB®.
`intmax`	Returns the largest possible integer number used in MATLAB®.
`intmin`	Returns the smallest possible integer number used in MATLAB®.

```
realmax
ans =
    1.7977e + 308
realmin
ans =
    2.2251e − 308
intmax
ans =
    2147483647
intmin
ans =
  −2147483648
```

eps : Floating-point relative accuracy

https://www.mathworks.com/matlabcentral/answers/57444-faq-why-is-0-3-0-2-0-1-not-equal-to-zero

Array Operations

- **reshape** changes dimensions of a matrix to any matrix with the same number of elements
- **diag** create diagonal matrix or get diagonal elements of matrix
- **rot90** rotates a matrix 90 degrees counter-clockwise
- **fliplr** flips columns of a matrix from left to right
- **flipud** flips rows of a matrix up to down
- **flip** flips a row vector left to right, column vector or matrix up to down
- **repmat** replicates an entire matrix; it creates *m x n* copies of the matrix
- **repelem** replicates each element from a matrix in the dimensions specified

Example: Create and Index Arrays

```
mat1 = zeros(3,3)
mat2 = ones(3,3)
mat3 = mat1 + 1;

mat4 = ones(6,6)
mat5 = repmat(mat2, 2, 2)

mat6 = eye(3,3)
mat7 = diag(ones(3,1))
```

```
mat1 = 3×3
     0     0     0
     0     0     0
     0     0     0

mat2 = 3×3
     1     1     1
     1     1     1
     1     1     1

mat4 = 6×6
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
```

```
mat5 = 6×6
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1

mat6 = 3×3
     1     0     0
     0     1     0
     0     0     1

mat7 = 3×3
     1     0     0
     0     1     0
     0     0     1
```

Example: Combine and Transform Array (horcat, vertcat, cat)

```
% cat, horzcat, vertcat
mat01 = reshape(1:9, 3, 3)
mat02 = horzcat(mat01, mat01)
mat03 = cat(2,mat01,mat01)
```

[mat01 mat01]

```
mat01 = 3×3
     1     4     7
     2     5     8
     3     6     9

mat02 = 3×6
     1     4     7     1     4     7
     2     5     8     2     5     8
     3     6     9     3     6     9

mat03 = 3×6
     1     4     7     1     4     7
     2     5     8     2     5     8
     3     6     9     3     6     9
```

```
mat05 = vertcat(mat01, mat01)
mat06 = cat(1,mat01,mat01)
```

[mat01; mat01]

```
mat05 = 6×3
     1     4     7
     2     5     8
     3     6     9
     1     4     7
     2     5     8
     3     6     9

mat06 = 6×3
     1     4     7
     2     5     8
     3     6     9
     1     4     7
     2     5     8
     3     6     9
```

Readability

Example: Cat

Problem Summary

This program is to create and manipulate a row or column vector, 2D matrix, and 3D matrix.

The following variables are created in advance using 'randi' and will be used for solving problems.

```
n = randi([4 10]);
val1 = randi(100);
val2 = randi(100);
vec_row1 = randi(100, 1, n);
vec_row2 = randi(100, 1, n);
vec_col1 = randi(100, n, 1);
vec_col2 = randi(100, n, 1);
mat1 = randi(100,n);
mat2 = randi(100,n);
```

```
mat_g = zeros(n,n,2);
mat_g(:,:,1) = mat1;
mat_g(:,:,2) = mat2;
```

(g) 'mat_g' that contains a 3D matrix having n x n x 2 array joining 'mat1' and 'mat2' in the third direction and in order (For example, if mat1 = [1,2 ; 4,5] and mat2 = [7,8 ; 10,11], mat_e = [[1,2 ; 4,5], [7,8 ; 10,11]])

```
mat_g = cat(3, mat1, mat2)
```

Example: Combine and Transform Array (repelem)

repelem replicates each element from a matrix in the dimensions specified

```
% repelem
mat08 = cat(2, ones(2,2), ones(2,2)+1)
mat08 = cat(1,mat08,mat08+2)
mat09 = repelem([1 2;3 4], 2,2)
```

```
mat08 = 4×4
     1     1     2     2
     1     1     2     2
     3     3     4     4
     3     3     4     4

mat09 = 4×4
     1     1     2     2
     1     1     2     2
     3     3     4     4
     3     3     4     4
```

1	1
1	1

2	2
2	2

1	1	2	2
1	1	2	2

1	1	2	2
1	1	2	2

3	3	4	4
3	3	4	4

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

1	2
3	4

1	1	2	2
1	1	2	2

3	3	4	4
3	3	4	4

Example: Combine and Transform Array

```
%flip, flipud, fliplr
mat10 = flip(mat01, 1)
mat11 = flipud(mat01)

mat12 = flip(mat01, 2)
mat13 = fliplr(mat01)

mat14 = transpose(mat01)
mat15 = mat01'
mat16 = flipud(rot90(mat01))
```

```
mat10 = 3×3
     3     6     9
     2     5     8
     1     4     7

mat11 = 3×3
     3     6     9
     2     5     8
     1     4     7

mat12 = 3×3
     7     4     1
     8     5     2
     9     6     3

mat13 = 3×3
     7     4     1
     8     5     2
     9     6     3
```

```
mat14 = 3×3
     1     2     3
     4     5     6
     7     8     9

mat15 = 3×3
     1     2     3
     4     5     6
     7     8     9

mat16 = 3×3
     1     2     3
     4     5     6
     7     8     9
```

Quiz: Create Matrices

M1 = 8×8

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

M2 = 8×8

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

M3 = 8×8

1	1	1	1	33	41	49	57
1	1	1	1	34	42	50	58
1	1	1	1	35	43	51	59
1	1	1	1	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

M4 = 8×8

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	0	0	44	52	60
5	13	21	0	0	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

M5 = 8×8

1	1	17	25	33	41	49	57
1	1	18	26	34	42	50	58
3	11	1	1	35	43	51	59
4	12	1	1	36	44	52	60
5	13	21	29	1	1	53	61
6	14	22	30	1	1	54	62
7	15	23	31	39	47	1	1
8	16	24	32	40	48	1	1

M6 = 8×8

1	9	17	25	33	41	1	1
2	10	18	26	34	42	1	1
3	11	19	27	1	1	51	59
4	12	20	28	1	1	52	60
5	13	1	1	37	45	53	61
6	14	1	1	38	46	54	62
1	1	23	31	39	47	55	63
1	1	24	32	40	48	56	64

M7 = 8×8

0	0	17	25	33	41	49	57
0	0	18	26	34	42	50	58
3	11	0	0	35	43	51	59
4	12	0	0	36	44	52	60
5	13	21	29	0	0	53	61
6	14	22	30	0	0	54	62
7	15	23	31	39	47	0	0
8	16	24	32	40	48	0	0

M8 = 8×8

1	9	100	100	100	100	100	100
2	10	100	100	100	100	100	100
100	100	19	27	100	100	100	100
100	100	20	28	100	100	100	100
100	100	100	100	37	45	100	100
100	100	100	100	38	46	100	100
100	100	100	100	100	100	55	63
100	100	100	100	100	100	56	64

M9 = 8×8

1	9	0	0	0	0	0	0
2	10	0	0	0	0	0	0
0	0	19	27	0	0	0	0
0	0	20	28	0	0	0	0
0	0	0	0	37	45	0	0
0	0	0	0	38	46	0	0
0	0	0	0	0	0	55	63
0	0	0	0	0	0	56	64

Quiz: Create Matrices (Hint)

eye(4,4)

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

repelem(eye(4,4), 2,2)

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

Slide Credits and References

- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition