BME1901 - Introductory Computer Sciences

Laboratory Handout - 1

Objectives

Learn about;

- Vectors
- Matrices
- Indices of Vectors/Matrices
- Vector/Matrix Manipulation
- Built-in Functions [ones(), zeros(), length(), size(), sin(), cos()]
- Graphing [plot()] function

TOOLS

Vectors and Matrices

All MATLAB variables are multidimensional arrays, no matter what type of data. To create a vector with four elements in a single row, seperate the elements with either a comma (,) or a space.

```
a = [1 2 3 4]
```

```
a = 1×4
     1     2     3     4
```

```
a = [1, 2, 3, 4]
```

```
a = 1×4
     1     2     3     4
```

A matrix is a two-dimentional array often used for linear algebra. To create a matrix that has multiple rows, seperate the rows with semicolons (;).

```
a = [1 2 3; 4 5 6; 7 8 9]
```

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

```
% EXERCISE: Create a 4x4 integer matrix

a = []
```

```
a =

     []
```

Another way to create a matrix is to use a function, such as **ones()**, **zeros()**.

```
% Create a 5-by-1 column vector of zeros
```

```
z = zeros(5,1)
```

```
z = 5×1
     0
     0
     0
     0
     0
```

```
% EXERCISE: Create a 2-by-4 matrix of ones

```

Vector/Matrix Manipulation

MATLAB allows you to process all of teh values in a matrix using a single arithmetic operator or function.

```
a = [1 2 3; 4 5 6; 7 8 9]
```

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

```
% Adding a value to each element of a matrix
a + 10
```

```
ans = 3×3
    11    12    13
    14    15    16
    17    18    19
```

```
% Mutliplying each element of a matrix
a * 10
```

```
ans = 3×3
    10    20    30
    40    50    60
    70    80    90
```

Every variable in MATLAB is an array that can hold many numbers. When you want to access selected elements of an array, use indexing. For example, consider the 4-by-4 magic() square A. The was to refer to a particular element in an arras is to specify row (i) and column (j) subscripts [A(i,j)].

```
A = magic(4)
```

```
A = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
A(4,2)
```

```
ans = 14
```

```
% Exercise: What is the value of the element located in the 4th column and 5th row of 5-by-5 ma

```

When you want to manipulate or change the value of an element in an array, you may again use indexing.

```
A(1,2) = 17
```

```
A = 4×4
    16    17     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

To refer to a range of elements in an array, use the colon (:) operator, which allows you to specify the range of the form **start:end**. The colon (:) alone, without start or end values, specifies all of the elements in that dimension.

```
A(1:3,1)
```

```
ans = 3×1
    16
     5
     9
```

```
A(3,:)
```

```
ans = 1×4
     9     7     6    12
```

The colon (:) operator also allows you to create an equally spaced vector of values using the more general form **start:step:end**. If you omit the **step**, MATLAB uses the default step value of 1.

```
B = 0:10:100
```

```
B = 1×11
     0    10    20    30    40    50    60    70    80    90   100
```

```
C = 0:10
```

```
C = 1×11
     0     1     2     3     4     5     6     7     8     9    10
```

```
A(1:2:end,:)
```

```
ans = 2×4
    16    17     3    13
     9     7     6    12
```

MATLAB Built-in Functions

MATLAB provied a large number of functions that perform common computational tasks. Functions are equivalent to subroutines or methods in other programming languages. Functions has two important parts, first is the function name and the secone is its input arguments to call a function write the function's name and enclose its input arguments in parantheses after it (**function_name(input1, input2,...)**).

L = length(X) function returns the length of the array dimension in X. For vectros, the length is simply the number of elements. The length of an empty array is zero.

```
v = 5:10
```

```
v = 1×6
     5     6     7     8     9    10
```

```
L = length(v)
```

```
L = 6
```

```
a = ones(3,5)
```

```
a = 3×5
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
len = length(a)
```

```
len = 5
```

[m,n] = size(X) return a row vector whose elements contain the length of the corresponding dimension of a matrix.

```
B = ones(3,5)
```

```
B = 3×5
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
[row, col] = size(B)
```

```
row = 3
col = 5
```

Y = sin(X) or **Y = cos(X)** returns the sine of the elements of X in radians. The sin and cos functions operates element-wise on arrays.

```
X = 0:0.5:2
```

```
X = 1×5
         0    0.5000    1.0000    1.5000    2.0000
```

```
sinX = sin(X)
```

```
sinX = 1×5
         0    0.4794    0.8415    0.9975    0.9093
```

```
cosX = cos(X)
```

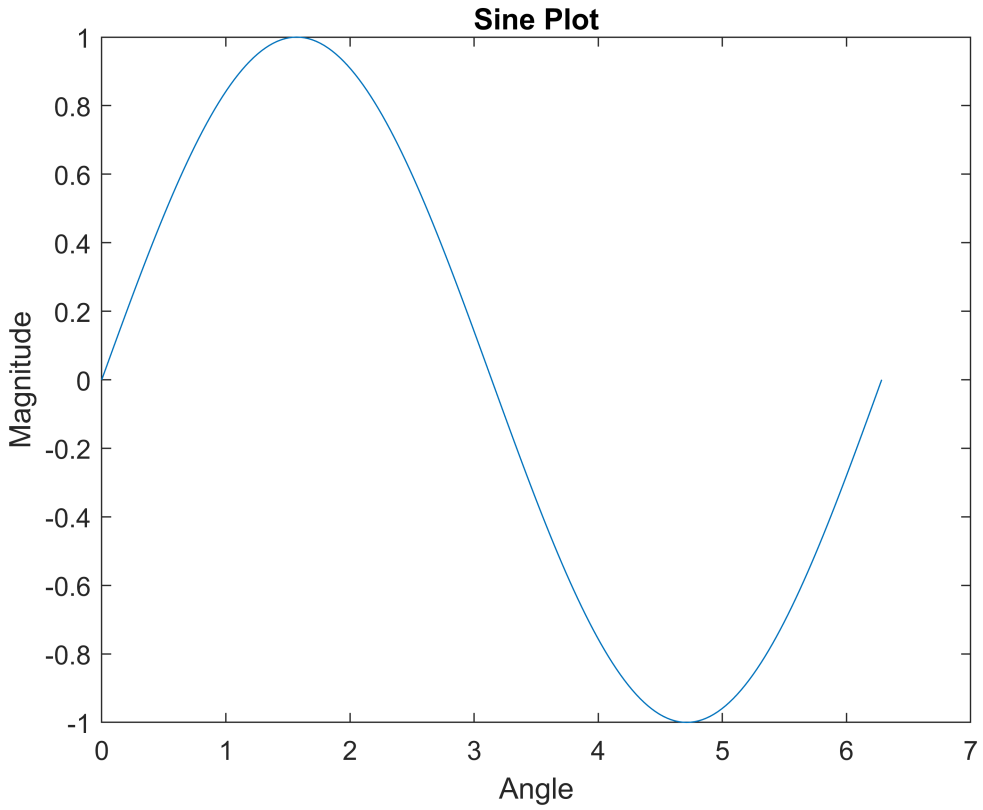
```
cosX = 1×5
    1.0000    0.8776    0.5403    0.0707   -0.4161
```

Plotting

plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.

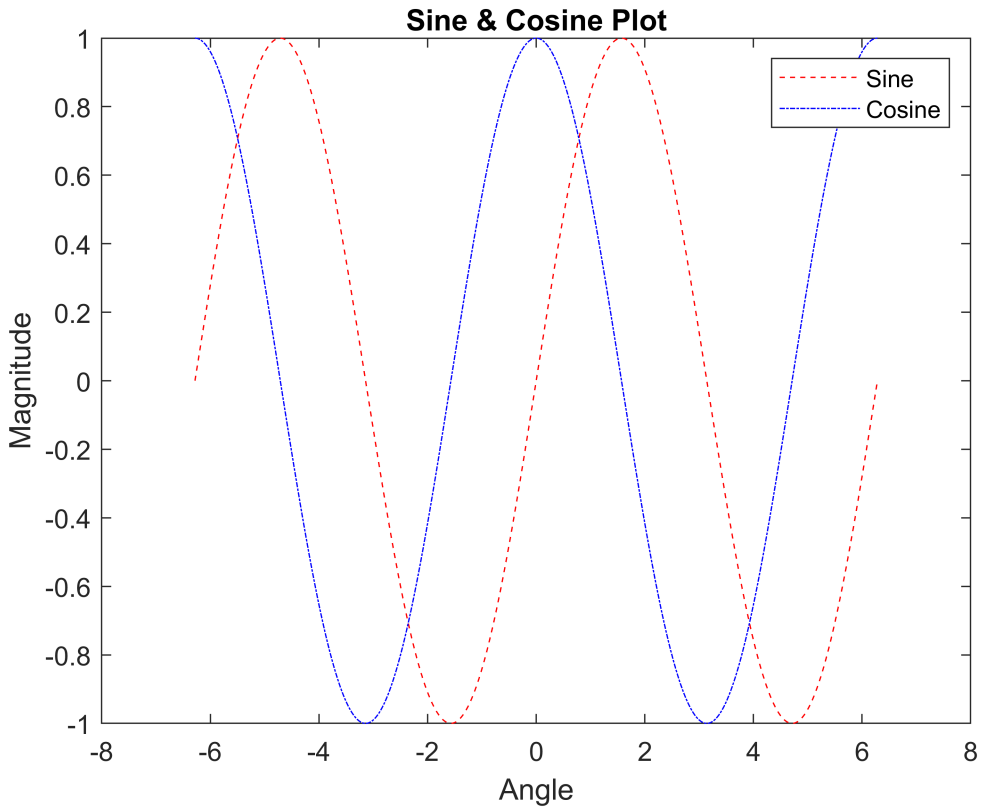
Example: Create x as a vector of linearly spaced values between 0 and 2π. Use an increment of π/100 between the values. Create y as sine values of x. Create a line plot of the data.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
title('Sine Plot')
xlabel('Angle')
ylabel('Magnitude')
```

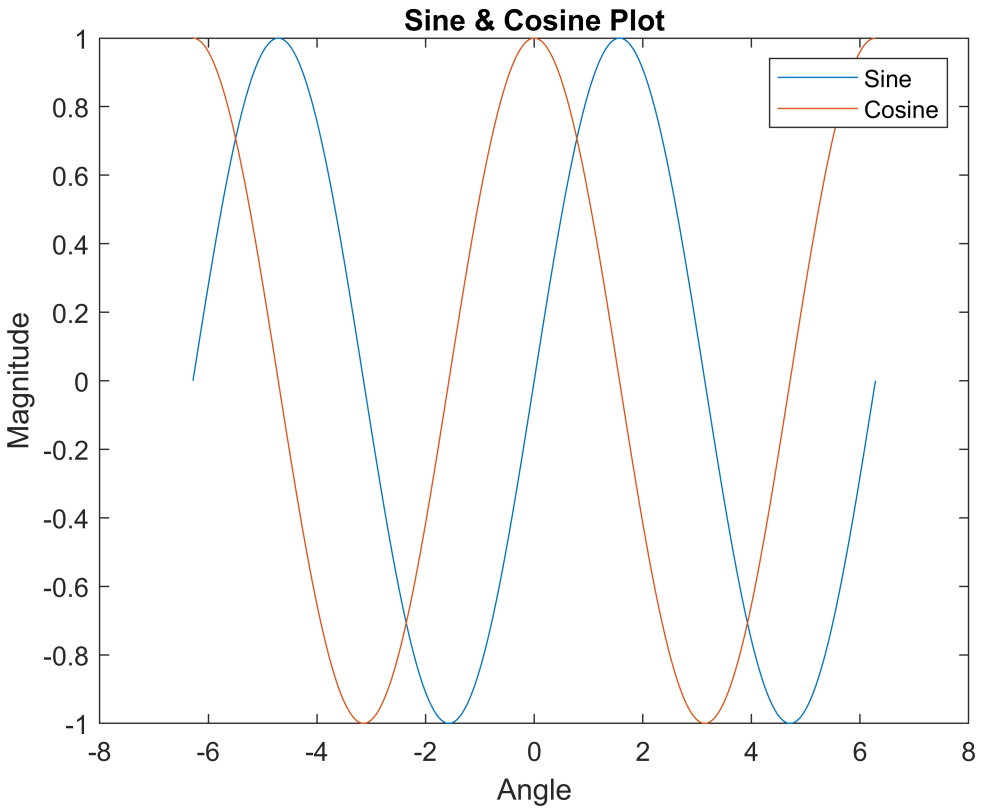


Example: Create x as a vector of linearly spaced values between -2π and 2π. Use an increment of π/100 between the values. Create y1 and y2 as sine and cosine values of x, respectively. Create a line plot of both sets of data.

```
x = -2*pi:pi/100:2*pi;
y1 = sin(x);
y2 = cos(x);

% Plotting both plots in a single line
plot(x, y1, 'r--', x, y2, 'b-.')
title('Sine & Cosine Plot')
xlabel('Angle')
ylabel('Magnitude')
legend('Sine', 'Cosine')
```




```
% Plotting each function seperately on the same graph
plot(x, y1)
hold on
plot(x,y2)
title('Sine & Cosine Plot')
xlabel('Angle')
ylabel('Magnitude')
legend('Sine', 'Cosine')
```



PROBLEMS

1. First find the result of the following statements by hand. Then write them to command window to check your results.

a. 1-6+2*5+8/2^3-3

b. (9/(3-2))^(1/2)*(6+4)

2. Create a 3x3 matrix using “A = magic(3)” in command window. Then do the following.

a. Take the first and second rows of matrix A and add them together to create vector B.

b. Take the first and third column of the matrix A and multiply them element-wise to create vector C (Hint: if you only use (*) you will encounter an error. Search help section of MATLAB for “element-wise multiplication”).

c. Find matrix multiplications of B*C and C*B (additionally check the sizes of both B and C matrices to notice that they are 1x3 and 3x1, respectively).

3. Create a time vector t from 0 to 10 seconds with 0.01 second steps. Create an acceleration of a at 1.1334 m/s^2 and an initial velocity of v0 at 8.5487 m/s. Using the formula for velocity v = v0 + a × t calculate velocity v

for all t. Plot the resulting velocity against time. Then using disp() function show velocity values for each second (0, 1, 2, ..., 10) on command window.

4. Create a time vector t from 0 to 10 seconds with 0.1 second steps, then create a frequency of f at 0.5 Hz. Create magnitudes of A and B at 2 and 5, respectively. Additionally, crate a phase of p at 0.2π.

a. Write the statement to find signal y1 such that it is equal to “A sin(2π f t)”.

b. Write the statement to find signal y2 such that it is equal to “B sin(2π f t + p)”.

c. Plot the signals y1 and y2 on the same graph against time.

d. Name the graph title as “Problem 3 Sine Graph”.

e. Name the y axis label as “Magnitude”.

f. Name the x axis label as “Time”.

g. Create legend for both signals with names “Sine 1” and “Sine 2”.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 2

OBJECTIVES

Learn about;

- Creating a script (m-file)
- Creating a live script (mlx-file)
- Character arrays
- ASCII character list
- Array casting [char()] function
- Built-in functions [max(), min(), sum(), round(), ceil(), floor(), input(), find()]
- Random number generators [rand(), randn()]
- Finding prime numbers [isprime()]
- Histogram graphing [histogram()] function

TOOLS

Creating a script (m-file)

Scripts are the simplest kind of program file because they have no input or output arguments. They are useful for automating series of MATLAB commands, such as computations that you have to perform repeatedly from the command line or series of commands you have to reference. You can create a new script by clicking the "New Script" button on the "Home" tab.

You can find more information about scripts here.

Example:

Create a new script file. Copy the code given below to the new script and save it as "numGenerator.m".

```
% This code generates 10000 random numbers from 0 through 100
% and plots the histogram of the generated numbers

columns = 10000;
rows = 1;

list = 100*rand(rows, columns);
histogram(list)
```

After you save your script you can run the code by clicking the "Run" button on the "Editor" tab.

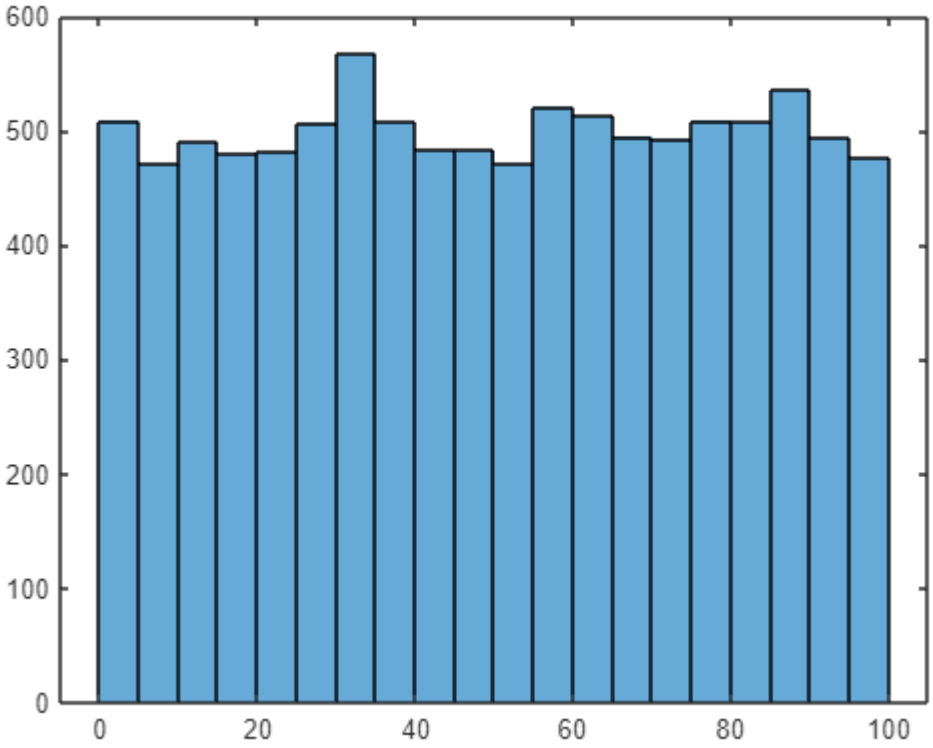
Creating a live script (mlx-file)

Live scripts provide an interactive medium for writing and executing code. They can be used to easily display the code and the output together in a single file. Text fields can also be added to live scripts which allows clear documentation and chronological explanation of the implemented code.

You can find more information about live scripts here.

This file is a live script. numGenerator.m script that was created before is implemented in a live script code cell below.

```
% This code generates 10000 random numbers from 0 through 100
% and plots the histogram of the generated numbers

columns = 10000;
rows = 1;

list = 100*rand(rows, columns);
histogram(list)
```



Character arrays

In MATLAB, you can store text in character arrays. A typical use for character arrays is to store pieces of text as character vectors. MATLAB displays character vectores with single quotes.

To store a 1-by-n squence of characters as a character vector, using the char data type, enclose it in single quotes. Character vectors have two principal uses, one is to specify single pieces of text, such as file names and plot labels, second is to represent data that is encoded using characters, in such cases, you might need easy access to individual characters. You can access individual characters or subsets of characters by indexing, just as you would index into a numeric array.

You can find more information about character arrays here.

```
chr = 'Hello world'
```

```
chr =
'Hello world'
```

```
seq = 'GCTAGAATCC';
seq(4:6)
```

```
ans =
'AGA'
```

ASCII character list

ASCII stands for American Standard Code for Information Interchange. Below is the ASCII character table, including descriptions of the first 32 characters. ASCII was originally designed for use with teletypes, and so the descriptions are somewhat obscure and their use is frequently not as intended.

Although today Unicode is the standard for character definition, it is defined back-compatible with ASCII. Therefore ASCII codes of the characters below are still usable almost everywhere.

In the table below, in the first columns the decimal values (base 10) and in the second columns the characters are given.

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html

As an example, 'd' has the integer value of 100 and '1' has the integer value of 49. If we calculate 'd' - '1' it evaluates to 100-49, or the integer value of 51 and the character equivalent of '3'.

```
a = 'n'; % ASCII: 110
b = '2'; % ASCII: 50

c = a - b
```

```
c = 60
```

```
a = 'n'; % ASCII: 110
b = 23;

c = a - b
```

```
c = 87
```

Array casting [char()] function

Function **char()** converts a numeric array into a character array using ASCII table.

```
a = 98;

char(a)
```

```
ans =
'b'
```

```
A = [77 65 84 76 65 66];

char(A)
```

```
ans =
'MATLAB'
```

Built-in functions

max(): Function max(X) returns the maximum elements of an array X. If X is a vector, then max(X) returns the maximum of whole vector. If X is a matrix, then max(X) is a row vector containing the maximum value of each column.

```
X = [23 42 37 18 52];

max(X)
```

```
ans = 52
```

```
X = [2 8 4; 7 3 9];

max(X)
```

```
ans = 1×3
```

```
     7     8     9
```

min(): Function min(X) returns the minimum elements of an array X. If X is a vector, then min(X) returns the minimum of whole vector. If X is a matrix, then min(X) is a row vector containing the minimum value of each column.

```
X = [23 42 37 18 52];

min(X)
```

```
ans = 18
```

```
X = [2 8 4; 7 3 9];

min(X)
```

```
ans = 1×3
     2     3     4
```

sum(): Function sum(X) return the sum of the elements of X along the first array dimension whose size does not equal 1. If X is a vector, then sum(X) returns the sum of all the elements. If X is a matrix, then sum(X) returns a row vector containing the sum of each column.

```
X = [23 42 37 18 52];

sum(X)
```

```
ans = 172
```

```
X = [2 8 4; 7 3 9];

sum(X)
```

```
ans = 1×3
     9    11    13
```

round(): Function 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 an integer with a larger magnitude.

```
X = [2.11 3.5; -3.5 0.78; -0.23 2]
```

```
X = 3×2
    2.1100    3.5000
   -3.5000    0.7800
   -0.2300    2.0000
```

```
round(X)
```

```
ans = 3×2
     2     4
```

```
    -4     1
     0     2
```

ceil(): Function ceil(X) rounds each element of X to the nearest integer which is greater in value.

```
X = [2.11 3.5; -3.5 0.78; -0.23 2]
```

```
X = 3×2
    2.1100    3.5000
   -3.5000    0.7800
   -0.2300    2.0000
```

```
ceil(X)
```

```
ans = 3×2
     3     4
    -3     1
     0     2
```

floor(): Function floor(X) rounds each element of X to the nearest integer which is smaller in value.

```
X = [2.11 3.5; -3.5 0.78; -0.23 2]
```

```
X = 3×2
    2.1100    3.5000
   -3.5000    0.7800
   -0.2300    2.0000
```

```
floor(X)
```

```
ans = 3×2
     2     3
    -4     0
    -1     2
```

Finding prime numbers [isprime()]

Function Y = isprime(X) returns a logical array of the same size as X. Value at Y(i) is true (1) when X(i) is a prime number. Otherwise, the value is false (0).

```
Y = isprime([2 3 0 6 10 11 25])
```

```
Y = 1×7 logical array
   1   1   0   0   0   1   0
```

input(): Function X = input(prompt) displays the text in **prompt** and waits for the user to input a value and press the Return (Enter) key. The user can enter expression, like pi/4 or magic(3) and can use variables in the workspace. If the user presses the Return key without entering anything, then input returns an empty matrix. If the user enters an invalid expression at the prompt, then MATLAB displays the relevant error message and redisplays the prompt.

Example: Create a new script file (m-file) with the code given below, and name it "multiply_by_10.m". Run the script and try different inputs, such as 1, magic(4), 'c' etc.

```
% This code takes an input from the user and displays input*10

prompt = 'What is the original value? ';

x = input(prompt);
y = x*10;

disp('Answer is:')
disp(y)
```

find(): Function k = find(X **r** Z) returns a vector of linear indices of elements in X which satisfies the relational condition set by the relational operation "**r**" (==, >, >=, <, <=, >~) and relational operand "Z".

```
X = magic(3)
```

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

```
k = find(X<=5)
```

```
k = 5×1
     2
     3
     4
     5
     9
```

```
X(k)
```

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

Example: Take a matrix X = magic(3) and replace all values smaller than 4 with 0.

```
X = magic(3);

k = find(X<4);

X(k) = 0;
disp(X)
```

```
     8     0     6
     0     5     7
     4     9     0
```

Random number generators

rand(): Function X = rand(r,c) returns an array of uniformly distributed random numbers with "r" rows and "c" columns between 0 and 1. (Note that each time this function is called, even with same inputs, the output will be different.)

```
X = rand(3,8)
```

```
X = 3×8
    0.6218    0.9520    0.5287    0.4964    0.3754    0.1378    0.0899    0.1749
    0.2576    0.7794    0.5741    0.9610    0.4789    0.6445    0.5323    0.4276
    0.6018    0.7014    0.6295    0.5614    0.5612    0.6386    0.7874    0.9441
```

randn(): Function X = randn(r,c) returns an array of normally distributed random numbers drawn from the standard normal distribution with "r" rows and "c" columns. (Note that each time this function is called, even with same inputs, the output will be different.)

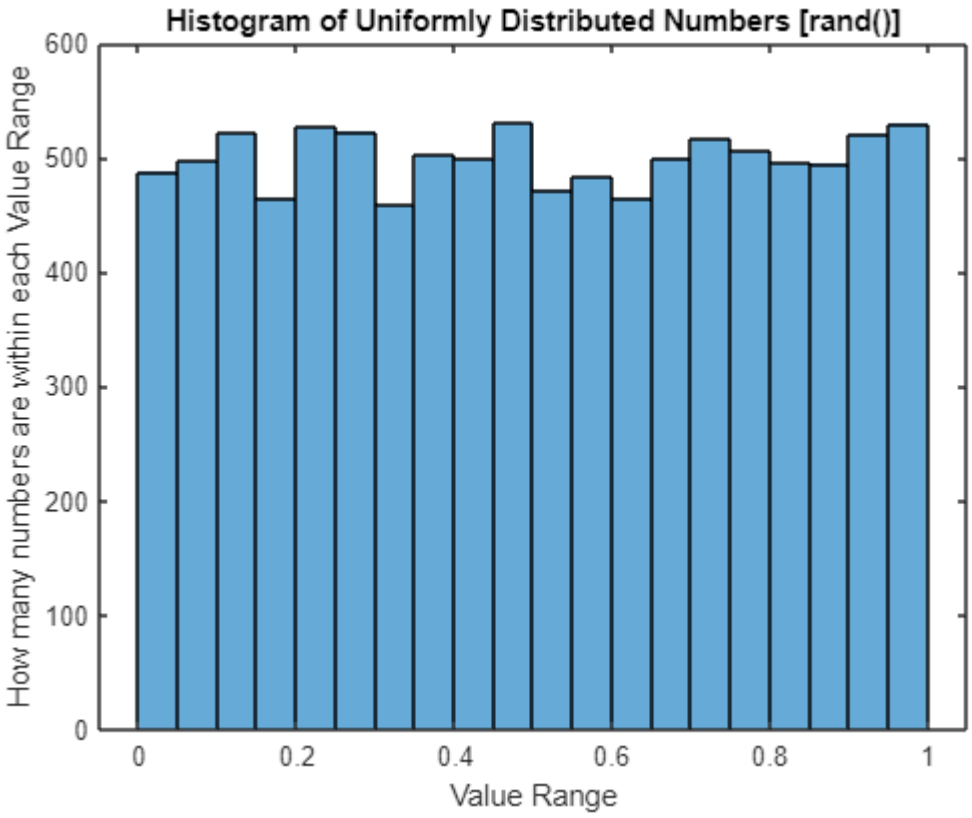
```
X = randn(4, 7)
```

```
X = 4×7
   -1.2411   -1.2713    0.4897   -0.2575   -0.4344   -0.9538   -1.1655
   -1.5513   -0.2673    1.0582   -0.2500    0.0305    0.3788    1.4370
    0.3084   -0.3956    1.8271   -0.0349    0.4700    0.4462   -0.0247
    1.3095    0.8741   -1.0624   -1.5778   -0.9858    0.7391   -0.4662
```

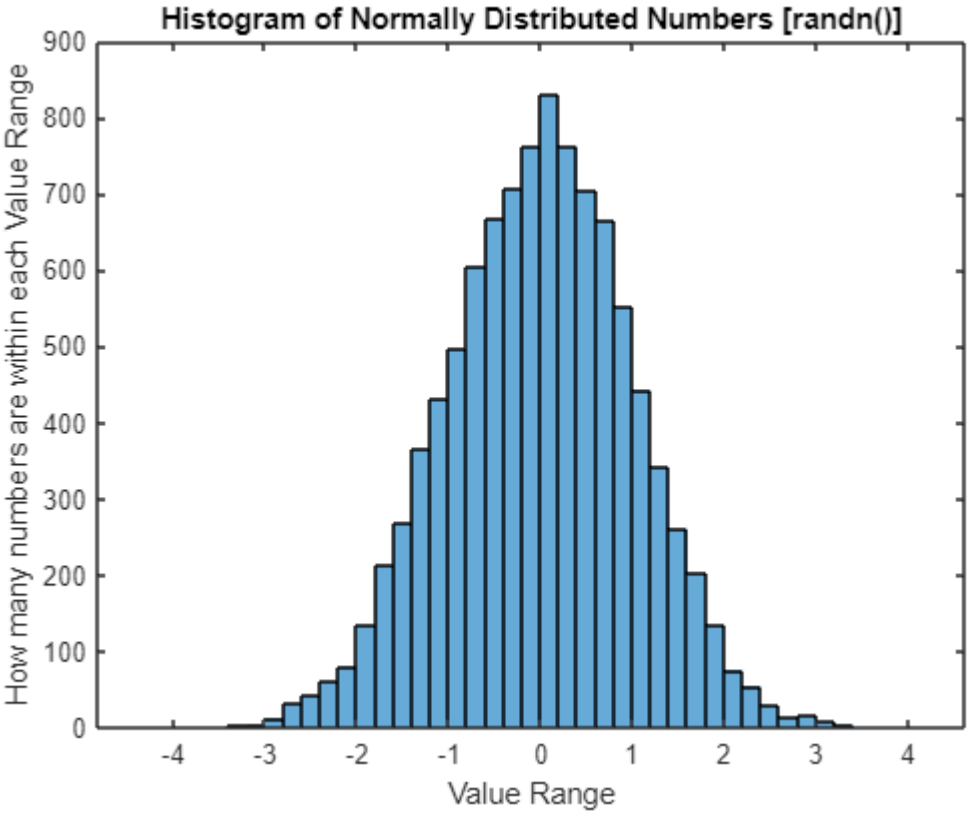
Histogram graphing [histogram()] function

Histograms are a type of bar plot for numeric data that group the data into bins. Function histogram(X) creates a histogram plot of X. The histogram function uses an automatic binning algorithm that return bins with a uniform width, chosen to cover the range of elements in X and reveal the underlying shape of the distribution. Function histogram() displays the bins as rectangles such that the height of each rectangle indicates the number of elements in the bin.

```
X = rand(1, 10000);
histogram(X)
title('Histogram of Uniformly Distributed Numbers [rand()]')
xlabel('Value Range')
ylabel('How many numbers are within each Value Range')
```




```
X = randn(1, 10000);
histogram(X)
title('Histogram of Normally Distributed Numbers [randn()]')
xlabel('Value Range')
ylabel('How many numbers are within each Value Range')
```



PROBLEMS

1. Write a script (m-file) called “int_col.m” that requests input “n”, which is an integer greater than 1, from the user. Then creates a column vector “v” of length “n” that contains all the integers between and including 1 and “n”. Then create a new vector “p” containing all the prime numbers in vector “v”. Then calculate the summation “s” of all values of vector “p”.

2. Write a script (m-file) called “rich.m” that computes how much money we have. It requests input “c”, which is a row vector whose 4 elements specify the number of coins 5 kr., 10 kr., 25 kr., and 50 kr. -in given order- that we have, from the user. Then calculates how much money we have in TL, and displays it.

3. In a university, there are 1000 students taking a course. Write a script (m-file) called “course_grade.m”, then follow the steps below.

a. Let’s assume the final grades of all the students are distributed normally. Create a normally distributed random row vector “grade”, such a way that the minimum value is 0, the maximum value is 100 and all values are integers.

b. Create a histogram graph of the grades.

c. Let’s assume the gender of the students are uniformly distributed as female “F” and male “M”. Create a uniformly distributed random row vector “gender”, such a way that it only contains characters “F” and “M” to represent female and male students, respectively.

d. Find the number of female and male students and display them.

e. Let’s assume the passing grade is 40 (if a student gets 40 or more, they pass). Create a 2x2 matrix “Pass_Fail”, such a way that the first column is the passed students, the second column is the failed students, the first row is female students and the second row is male students. Then display this matrix.

f. Find and display the summation of all elements in matrix “Pass_Fail” to check your code. The result should be equal to the total number of students in the beginning.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 3

OBJECTIVES

Learn about;

- Making music [sound()] function
- Built-in functions [disp(), num2str()]

TOOLS

ASCII character list

ASCII stands for American Standard Code for Information Interchange. Below is the ASCII character table, including descriptions of the first 32 characters. ASCII was originally designed for use with teletypes, and so the descriptions are somewhat obscure and their use is frequently not as intended.

Although today Unicode is the standard for character definition, it is defined back-compatible with ASCII. Therefore ASCII codes of the characters below are still usable almost everywhere.

In the table below, in the first columns the decimal values (base 10) and in the second columns the characters are given.

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html

As an example, 'd' has the integer value of 100 and '1' has the integer value of 49. If we calculate 'd' - '1' it evaluates to 100-49, or the integer value of 51 and the character equivalent of '3'.

```
a = 'n'; % ASCII: 110
b = '2'; % ASCII: 50

c = a - b
```

```
c = 60
```

```
a = 'n'; % ASCII: 110
b = 23;

c = a - b
```

```
c = 87
```

input(): Function X = input(prompt) displays the text in **prompt** and waits for the user to input a value and press the Return (Enter) key. The user can enter expression, like pi/4 or magic(3) and can use variables in the

workspace. If the user presses the Return key without entering anything, then input returns an empty matrix. If the user enters an invalid expression at the prompt, then MATLAB displays the relevant error message and redisplays the prompt.

Example: Create a new script file (m-file) with the code given below, and name it "multiply_by_10.m". Run the script and try different inputs, such as 1, magic(4), 'c' etc.

```
% This code takes an input from the user and displays input*10

prompt = 'What is the original value? ';

x = input(prompt);
y = x*10;

disp('Answer is:')
disp(y)
```

find(): Function k = find(X **r** Z) returns a vector of linear indices of elements in X which satisfies the relational condition set by the relational operation "**r**" (==, >, >=, <, <=, >~) and relational operand "Z".

```
X = magic(3)
```

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

```
k = find(X<=5)
```

```
k = 5×1
     2
     3
     4
     5
     9
```

```
X(k)
```

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

Example: Take a matrix X = magic(3) and replace all values smaller than 4 with 0.

```
X = magic(3);

k = find(X<4);

X(k) = 0;
disp(X)
```

```
     8     0     6
     0     5     7
     4     9     0
```

disp(): Function disp(X) displays the value of variable X without printing the variable name. If a variable contains an emtpy array, disp returns without displaying anything.

```
A = [15 150];
disp(A)
```

```
    15   150
```

```
S = 'Hello World.';
disp(S)
```

```
Hello World.
```

num2str(): Function Y = num2str(X) converts a numeric array into a character array that represents the numbers. The output format depends on the magnitudes of the original values. num2str() is useful for labeling and titling plots with numeric values. (This function does not use ASCII conversion table.)

```
s = num2str(pi)
```

```
s =
'3.1416'
```

```
a = 5;
s = num2str(a)
```

```
s =
'5'
```

```
b = 2;
c = 6;
s = num2str(b+c)
```

```
s =
'8'
```

Function num2str() is used in disp() to display text on the screen dependent on a numeric variable.

```
A = 15;
disp(['Value of variable A is ', num2str(A), '.'])
```

```
Value of variable A is 15.
```

Random number generators

rand(): Function X = rand(r,c) returns an array of uniformly distributed random numbers with "r" rows and "c" columns between 0 and 1. (Note that each time this function is called, even with same inputs, the output will be different.)

```
X = rand(3,8)
```

```
X = 3×8
    0.4241    0.9754    0.4431    0.9141    0.0554    0.7945    0.3212    0.1151
    0.5199    0.0828    0.4960    0.8955    0.0425    0.8777    0.6728    0.5107
    0.9685    0.1584    0.6850    0.0253    0.4104    0.4830    0.9488    0.2505
```

randn(): Function X = randn(r,c) returns an array of normally distributed random numbers drawn from the standard normal distribution with "r" rows and "c" columns. (Note that each time this function is called, even with same inputs, the output will be different.)

```
X = randn(4, 7)
```

```
X = 4×7
   -1.3080    0.5944   -0.9674    0.6598   -0.9413   -0.3476   -0.7456
   -1.6914   -1.1978   -0.4278    0.8977    0.8163   -0.8359    0.6944
    0.0408   -1.7686    2.0658   -1.2627    1.0396    0.2643   -0.6389
    0.6605   -0.1920    2.7623   -0.5091   -0.4929    0.0179   -0.2719
```

Making music [sound()] function

Function sound(X) converts an array of signal data to sound by sending audio signal X to the speaker at the default sampling rate of 8192 hertz.

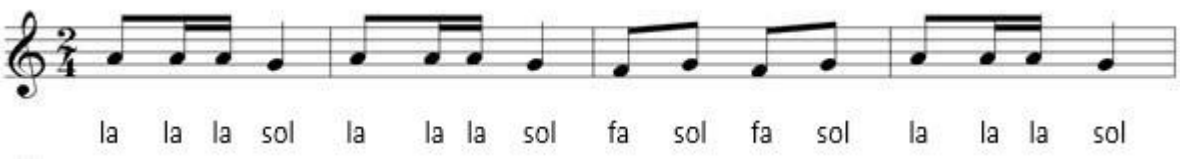
```
load gong.mat;

sound(y)
```

Example: Make a pure A4 (la) note, with frequency of 440Hz "fn", using 8192Hz of sampling rate "fs" and a total of 10000 samples "n".

```
n = 1:1:10000;
fs = 8192;
fn = 440;

z = cos(2*pi*n*fn/fs);

sound(z,fs)
```

PROBLEMS

1. Write a script (m-file) called “melody.m” that plays the tune given below. A4 (la) note has the frequency of 440 Hz, G4 (sol) note has the frequency of 392 Hz and F4 (fa) note has the frequency of 349.2 Hz. Use a sampling frequency of 8192 Hz. Use 1250 samples for sixteenth notes, 2500 samples for eighth notes and 5000 samples for quarter notes. Use 125 zeros between notes to separate them.



2. Write a script (m-file) called “dice_throw.m” that rolls random dice (6 sided dice) for 4 people and displays results for each person and the winner. Program first should ask the user to give how many dice will be thrown by each player, then roll the dice randomly and decide the winner depending on the sum of the results of dice thrown by each player.

a. Repeat this process 5 times and record the winners.

b. Let’s say one of the players is cheating by using weighted dice which half of the time comes up as 6. Modify your code to simulate this effect. Repeat this process 5 times and record the winners.

c. Compare the winners of sub-questions a and b. Who won more? Is there a difference in winnings?

3. Write script (m-file) called “num_2_star.m” that requires input from the user which is a character array; then displays the resulting character array. Then the code finds all the numbers in this array and replaces them with “*” characters; then display the resulting character array.

4. Create a script (m-file) called “circle_dots.m” then follow the steps below.

a. Request a radius “r” (equal or less than 10) as input from the user to draw a circle centered around origin.

b. Write a code that would create points of a circle in cartesian coordinates using polar to cartesian coordinate conversions given below.

$$x_c = r.\cos(\theta),\ y_c = r.\sin(\theta)$$

c. Uniformly distribute 20 dots with cartesian coordinates of “xd” and “yd” between -10 and 10 for each coordinate

d. Plot the resulting circle and dots on the same graph with axis limitations from -10 to 10 for each coordinate.

e. Write a code that will find and displays how many dots are within the circle using the circle equation given below.

$$r^2 = x^2 + y^2$$

BME1901 - Introductory Computer Sciences

Laboratory Handout - 4

OBJECTIVES

Learn about;

- if-elseif-else conditional assignments
- Solving various questions

TOOLS

if-elseif-else Conditional Assignments

if Conditional Assignment

"if" executes the statements given below it when condition given in expression is true. "end" is needed to mark the end of statements related to "if".

```
if (expression)
    statements
end
```

Example: Write a code that checks whether user input "a" (real number" is greater than 0, then displays "Given number is positive" if its true, and does nothing in other situations.

```
a = input('Enter a number: ');

if a>0
    disp('Given number is positive.')
end
```

```
Given number is positive.
```

if-else Conditional Assignment

"if-else" executes the statements given below the "if" when condition given in the expression is true, and executes statements below "else" when the condition given in expression is false. "end" is needed to mark the end of the statements related to "if-else".

```
if (expression)
    statements
else
    statements
end
```

Example: Write a code that checks whether user input "a" (real number) is greater than 0, then displays "Given number is positive." if it is true, and displays "Given number is negative or zero." in other conditions.

```
a = input('Enter a number: ');

if a>0
```

```
    disp('Given number is positive.');
else
    disp('Given number is negative or zero.')
end
```

```
Given number is negative or zero.
```

if-elseif-else Conditional Assignment

"if-elseif-else" executes statements given below the "if" when condition given in if's expression is true, when if's expression is false and elseif's expression is true then statements given below the "elseif" are executed, when all expressions are false then statements given below "else" are executed. "if-elseif-else" codes may have any number of "elseif" conditional assingmnets but only one of each "if" and "else". "end" is needed to mark end of the statements related to "if-elseif-else"

```
if (expression)
    statements
elseif (expression)
    statements
      .
      .
      .
elseif (expression)
    statements
else
    statements
end
```

Example: Write a code that checks whether user input "a" (real number) is greater than 0, then displays "Given number is positive." if it is true, if previous check is false then checks whether "a" is less than 0 and displays "Given number is negative." and displays "Given number is zero." in other conditions.

```
a = input('Enter a number: ');

if a>0
    disp('Given number is positive.');
elseif a<0
    disp('Given number is negative.');
else
    disp('Given number is zero.');
end
```

```
Given number is zero.
```

PROBLEMS

1. Write a script (m-file) called "grade.m" that requests student's grade (between 0-100) from the user then displays the letter grade of the student with respect to the table given below.

A	B	C	D	F
100 – 80	79 – 60	59 – 40	39 – 20	19 – 0

2. Write a script (m-file) called “salary.m” that request a three element vector input from the user, where first element is the hourly wage of the employee, second element is the weekly working hours (normal weekly working hours is 40 hours anything more is counted as overtime working hours), third element is the weekend overtime working hours. Normal working hours pay the regular hourly wage, weekday overtime working hours pay 1.5 times of the hourly wage and weekend overtime working hours pay double the hourly wage. After the request of the input, code should calculate and display the weekly salary of the employee.

3. Write script (m-file) called “nucleic_acid.m” that requests a character array input from the user in the form of a nucleic acid chain (e.g.: ATGCAACGATTCG). Then checks and displays the appropriate message with respect to given conditions below.

a. Displays “Given chain is a DNA” if components are A (adenine), T (thymine), G (guanine), C (cytocine).

b. Displays “Given chain is an RNA” if components are A (adenine), U (uracil), G (guanine), C (cytocine).

c. Displays “Given chain may be a DNA or an RNA” if components include A (adenine), G (guanine), C (cytocine) bot do not include T (thymine) and U (uracil).

d. Displays “Given chain cannot be a nucleic acid chain” in other conditions (i.e.: if components include T (thymine) and U (uracil) at the same time, or any other characters other than nucleic acids).

BME1901 - Introductory Computer Sciences

Laboratory Handout - 5

OBJECTIVES

Learn about;

- for and while loops
- fprintf() function
- Solving various questions

TOOLS

for Loops

"for" loops execute a group of given statements for a specified number of times. Index values for the "for" loops may be given in the following forms;

- initVal:endVal - Increment the index variable from initVal to endVal by 1, and repeat the execution of statements until index is greater than endVal.
- initVal:step:endVal - Increment hte index variable by the value "step" in each iteration.
- valArray - Create a column vector, index, from subsequent columns of array valArray on each iteration. The input valArray can be of any MATLAB data type.

"end" is needed to mark the end of statements related to a for loop.

```
for (index = values)
   statements
end
```

Example: Write a code that gives the even indexed values in array B from the input array A.

```
A = input('Enter a number array: ');

B = zeros(floor(length(A)/2), 1);
for i = 2:2:length(A)
    B(i/2) = A(i);
end

disp(B);
```

```
    2
    4
```

while Loops

"while" loops repeat the execuiton of a group of statements while the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false. "end" is needed to mark the end of statements related to a while loop.

```
while (expression)
    statements
end
```

Example: Write a code that calculates the factorial of the input.

```
N = input('Enter a number for calculation of its factorial: ');

F = N;

while N>1
    N = N-1;
    F = F*N;
end

disp(F);
```

```
   120
```

ASCII Character List

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

fprintf() Function

fprintf(format, A1, A2,...,An) function displays the results on the screen from data given in "A1, A2,...,An" as described in "format". fprintf() function uses special characters in "format" section as a placeholder for data given in "A1, A2,...,An", these placeholder characters also specify the type of data. Additional characters may also be used in "format" to shape the output. Some characters and their descriptions are given in Table 1.

Table 1 - fprintf() function special characters

Character	Description
%i or %d	Base 10 signed integer
%u	Base 10 unsigned integer
%f or % X.Yf	Fixed-point notation (for numbers with a decimal point).
	In fixed-point notation precisionoperators of X (field width) and Y (precision) may be used.
%c	Single character
%s	String array
\n	New line

```
a = 10;
fprintf('a is equal to %d', a)
```

```
a is equal to 10
```

```
b = -4;
fprintf('b is equal to %d', b)
```

```
b is equal to -4
```

```
c = pi;
fprintf('c is equal to %f', c)
```

```
c is equal to 3.141593
```

```
d = pi;
fprintf('d is equal to %3.2f', d)
```

```
d is equal to 3.14
```

```
e = 'AB';
fprintf('character %c', e)
```

```
character Acharacter B
```

```
f = 'AB';
fprintf('string %s', f)
```

```
string AB
```

```
fprintf('Without new line.'); fprintf('Both texts on same line.')
```

```
Without new line.Both texts on same line.
```

```
fprintf('With new line character.\n'); fprintf('Texts on different lines.')
```

```
With new line character.
Texts on different lines.
```

PROBLEMS

1. Write a script (m-file) called “diamond.m” that requests number of stars on one edge of a diamond shape as an input from the user then then displays a dimond shape using star (*) and space characters.

2. Let’s assume you and your friend want to send secret messages to each other. Therefore you come up with an encryption/decryprition method of swapping each letter of the English alphabet (26 letters) from their beginning position to their position from the end for uppercase and lowercase letters seperately (e.g.: “a” becomes “z”, “b” becomes “y”, “C” becomes “X”, “W” becomes “D”, …) while keeping all other characters unchanged. Write a script (m-file) called “encrypt_decrypt.m” that request a character array input from the user to accomplish this encryption/decrption. First give the program a character array to encrypt the message. Then take the output from part a and give it to the program to decrypt the message.

3. Write a script (m-file) called “fibonacci_rabbits.m” that would simulate the reproduction cycle of an alien rabbit spicies that come to Earth. This alien rabbit species do not require a mate to reproduce (so a single rabbit can create more rabbits alone one at a time) and they are immortal (they never die). Only condition for a rabbit from this alien spicies to reproduce is to grow up. When a baby alien rabbit is born it takes one month to grow up and after that it may reproduce. A baby alien rabbit comes to Earth on an asteroid. When the population of the alien rabbits surpass the human population on Earth they will invade the whole Earth. How long would it take for them to accomplish this goal and what would their population be at that time? You may assume human population on Earth around 8 billion.

4. There’s a famous legend about the origin of chess that goes like this. When the inventor of the game showed it to the emperor of India, the emperor was so impressed by the new game, that he said to the man “Name your reward!”. The man responded, “Oh emperor, my wishes are simple. I only wish for this. Give me one grain of rice for the first square of the chessboard, two grains for the next square, four for the next, eight for the next and so on for all 64 squares, with each square having double the number of grains as the square before.” The emperor agreed, amazed that the man had asked for such a small reward - or so he thought. After a week, his treasurer came back and informed him that the reward would add up to an astronomical sum, far greater than all the rice that could conceivably be produced in many many centuries! Write a script (m-file) called “story_of_chess.m” that would calculate how many rice grains should be on each square on a 8 by 8 chess board. Then calculate the total number of rice grains. Taking the average weight of a rice grain as 0.029 grams find the total weight of all rice grains. Assuming 1 billion tonnes of rice being produced globally per year how long would it take to whole world today to pay the inventor of chess.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 6

OBJECTIVES

Learn about;

- User defined functions
- factorial() function
- break statement
- Solving various questions

TOOLS

User Defined Functions

A user defined dunction with syntax "[y1,...,yN] = myFunc(x1,...,xM)" declares a function named "myFunc" that accepts inputs "x1,...,xM" and returns outputs "y1,...yN". This declaration statement must be the first executable line of the function. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores. After the declaration of the functon user may define any statement as part of the function. Built-in MATLAB functions, user defined functions, for & while loops, and if-elseif-else clauses may be called within a function file. "end" may be used to mark the end of statements related to a user defined function.

```
Function [outputs] = FunctionName(inputs)
  statements
end
```

Example: Write a function "add" that accepts two inputs, adds them together, and returns the result.

```
function c = add(a,b)
  c = a + b;
end
```

```
add(2,3)
```

```
ans = 5
```

Example: Write a function "average" that accepts an input vector, calculates the average of the values, and returns a single result.

```
function ave = average(x)
  ave = sum(x)/length(x);
end
```

```
z = 1:99;
y = average(z)
```

```
y = 50
```

Example: Write a function "stat" that returns the mean and standard deviation of an input vector.

```
function [m,s] = stat(x)
   n = length(x);
   m = sum(x)/n;
   s = (sum((x-m).^2/n))^(1/2)
end
```

```
values = [12.7, 45.4, 98.9, 26.6, 53.1];
[ave, stdev] = stat(values)
```

```
ave = 47.3400
stdev = 29.4124
```

factorial() Function

Function factorial(n) returns the product of all positive integers less than or equal to n, where n is a nonnegative integer value. If n is an array, then the output contains the factorial of each value of n. The data type ans size of the output is the same as that of n. The factorial of n is commonly written in math notation using the exclamation point character as n!. *Note that n! is not a valid MATLAB syntax for calculation the factorial of n.*

```
factorial(10)
```

```
ans = 3628800
```

break Statement

"break" statement terminates the execution of a for or while loop. Statements in the loop after the "break" statement do not execute (statements before break execute). In nested loops, break exits only from the loop in which it occurs. Control passes to the statement that follows the end of that loop.

Example: Write a script (m-file) "sum_rand.m" which sums a squence of uniformly distributed random numbers between 0 and 1 until the nex random number is greater than a given upper limit. Then exit the loop using a break statement.

```
limit = 0.8;

s = 0;
n = 0;

while 1
  tmp = rand(1);
  if tmp > limit
     break
  end
  s = s + tmp;
  n = n+1;
end

disp(s)
```

```
    4.5566
```

```
disp(n)
```

```
    11
```

PROBLEMS

1. Write a function called “sort3” that accepts three number inputs, then sorts them in increasing order in a vector, and returns the vector. (You may not use any built-in MATLAB functions.)

2. Write a function called “taylor_exp” that would calculate the exponential function given below using Taylor Series approximation.

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Function should accept two number inputs. First is the exponent “x” and second is the upper limit of the last term in the sum (e.g.: 10^{-6}). When the last term is smaller than the given limit program should end returning the approximation for ex as the first output and how many iterations it took as the second output. (Use for loop from 1 to infinity in your solution).

3. Write a function called “guess_number” which does not have any inputs or outputs. Function should create a random integer between 0 and 10 from a uniformly distributed population. Then function should require an input from the user as their guess for the number. If the user’s number is smaller or bigger then the function’s number, the program should print a relevant message (Too big/small. Try again.) on the screen. This process should continue until the user guesses the number correctly or user runs out of three tries. If the user correctly guesses the number in three tries the program should end printing success with how many tries that it took. If the user runs out of three tires the program should end priting failure.

4. Write a function called “throw_dice” that accepts how many dice (6 sided dice) to be rolled randomly and returns each of their value as a vector. Then write another function called “marmut” that accepts how many players are playing. This function should roll two dice (using throw_dice function) and record the sum of dice for each player. Before each roll the program should request the user to hit enter, then roll the dice and display the result for each player. In any moment if a player rolls 10, 11 or 12 as the sum of his/her dice that player direcly wins the game and function ends printing that player as the winner with his/her result. If no one wins via rolling 10, 11 or 12 then the winner(s) is the player(s) who rolled the highest sum. Hence, function prints his/her player number(s) and result(s) before ending.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 7

OBJECTIVES

Learn about;

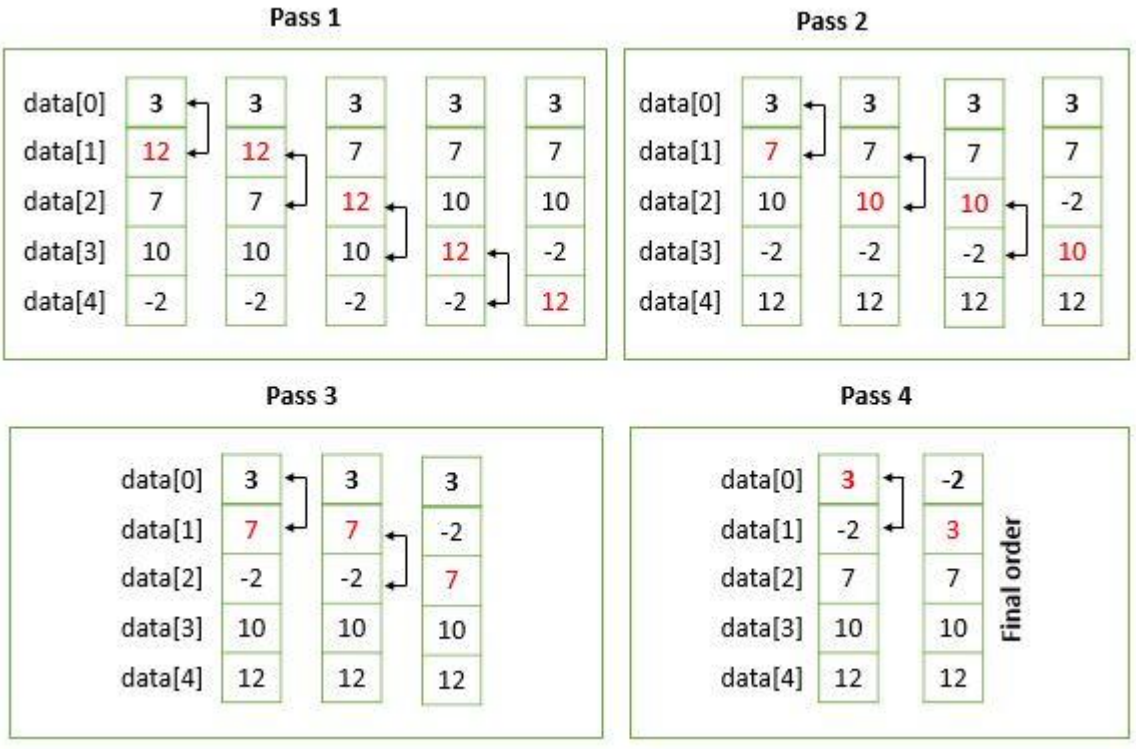
- Bubble sort algorithm
- Built-in sort() function
- tic toc time keeping functions
- Solving various questions

TOOLS

Bubble sort algorithm (1, 2, 3)

Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. In each pass only a single value is put in the correct place. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list. Although the algoritm is simple, it is too slow and impractical for most problems. Therefore this algorithm is not suitable for large datasets.

Figure 1: How bubble sort works.



Built in sort() function

Built in sort(A) function in MATLAB sorts the elements of the input "A" in ascending order. If A is a vector, then sort(A) sorts the vector elements. If A is a matrix, then sort(A) treats the columns of as vectors and sorts each column. Using sort(A, direction) returns the sorted elements of A in the order specified by direction using any of the previous syntaxes. 'ascend' indicates ascending order (default), and 'descend' indicates descending order.

```
A = [9 0 -7 5 3 8 -10 4 2];

B = sort(A)
```

```
B = 1×9
   -10    -7     0     2     3     4     5     8     9
```

```
B = sort(A, 'descend')
```

```
B = 1×9
     9     8     5     4     3     2     0    -7   -10
```

```
A = [10 -12 4 8; 6 -9 8 0; 2 3 11 -2; 1 1 9 3];

B = sort(A)
```

```
B = 4×4
     1   -12     4    -2
     2    -9     8     0
     6     1     9     3
    10     3    11     8
```

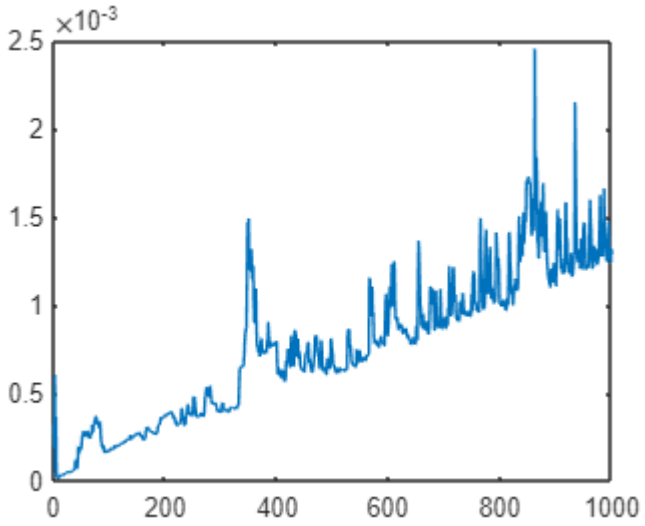
Time keeping functions (tic, toc)

"tic" starts a stopwatch timer to measure performance. the function records the internal time at execution of the "tic" command. Display the elapsed time with the "toc" function.

"toc" reads the elapsed time from the stopwatch timer started by the "tic" function. The function reads the internal time at the execution of the "toc" command, and displays the elapsed time since the most recent call to the "tic" function that had no output, in seconds.

```
tic
statements
toc
```

```
for n = 1:1:1000
    tic;
    for j = 1:1:n
        factorial(j);
    end
    t(n) = toc;
end

plot(t)
```



PROBLEMS

1. Write a function called “bubble_sort” that accepts a vector input, then using bubble sort algorithm sorts the input in ascending order, and returns the sorted vector. (You may not use any built-in MATLAB functions.)

2. Write a script (m-file) called “sort_compare.m” that compares the sorting times of bubble_sort() function from question 1 and built-in MATLAB sort() function. Program should create a random vector and sort it in ascending order using both functions while keeping execution times for both. Then the program should print the vector length, time for bubble sort function and time for built-in sort function on the screen. The program should repeat this process for random vectors with length 10, 100, 1000 and 10000.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 8

OBJECTIVES

Learn about;

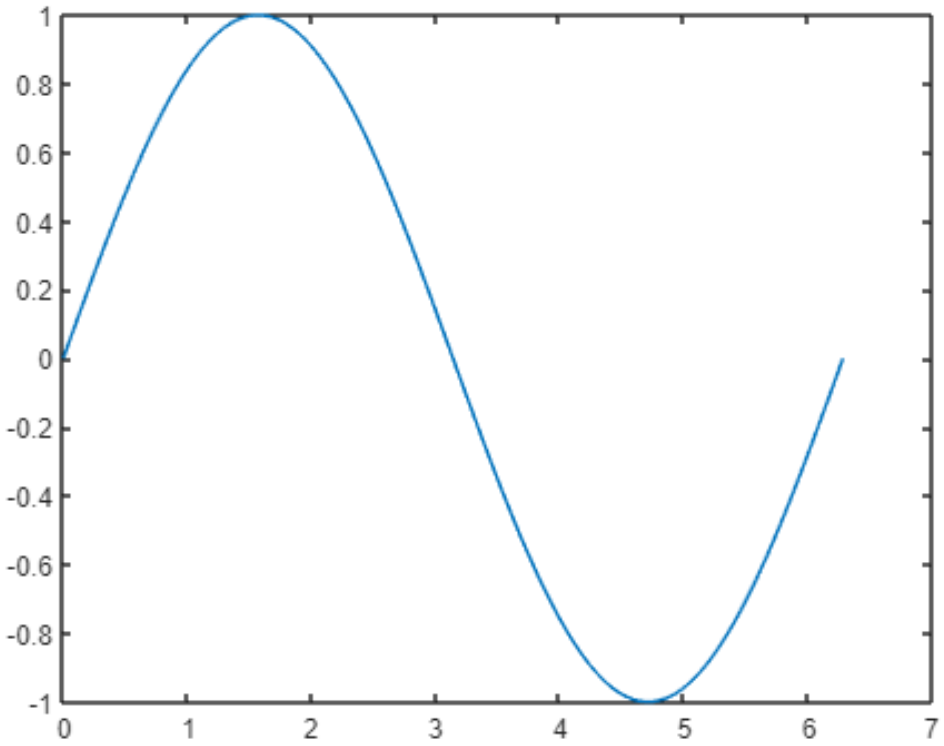
- Graphical representation of data
- Drawing multiple plots
- Formatting plot graphs
- Recursive functions
- Solving various questions

TOOLS

Graphical Representation of Data

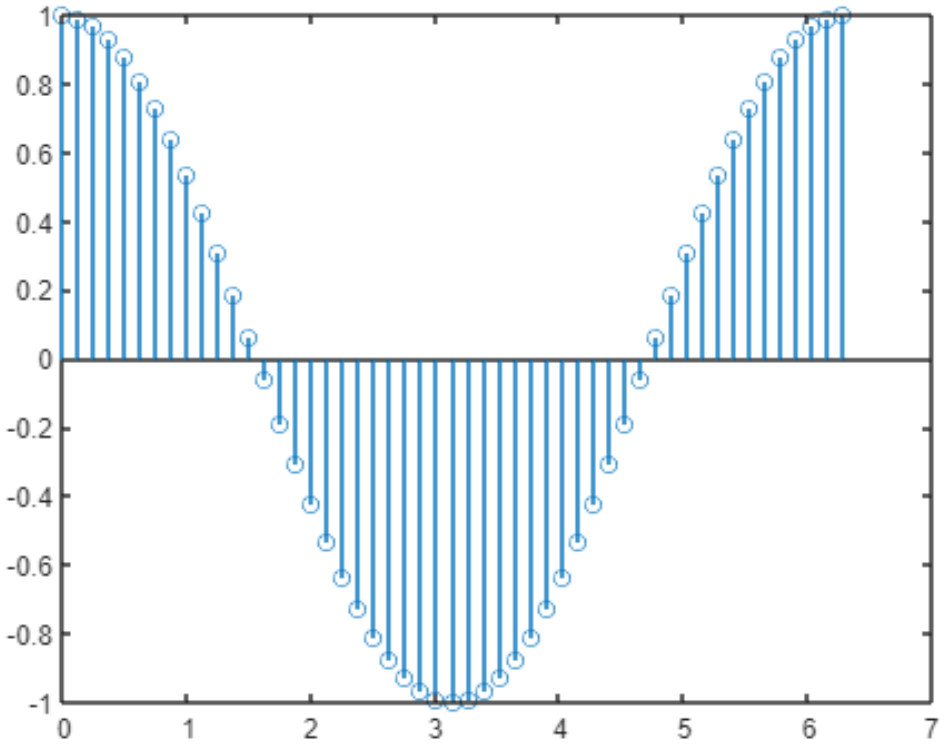
plot(): plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X. The X and Y inputs must be vectors or matrices of the same size.

```
x = 0:pi/100:2*pi;
y = sin(x);

plot(x,y)
```



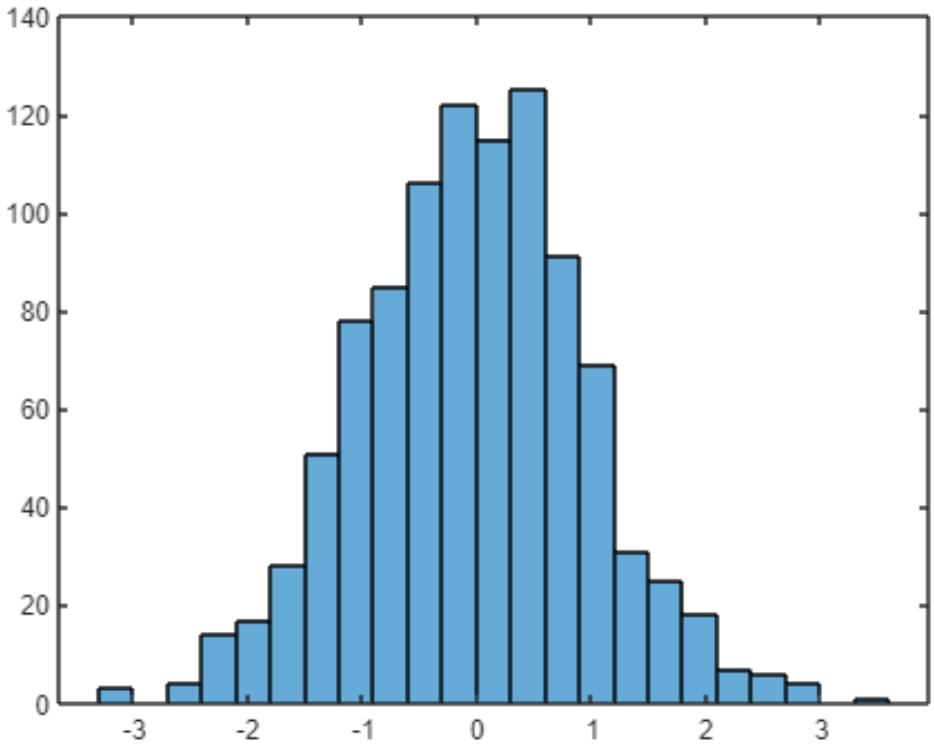
stem(): stem(X,Y) plots teh data sequence Y at values specified by X. The X and Y inputs must be vectors or matrices of the same size.

```
x = 0:pi/25:2*pi;
y = cos(x);

stem(x,y)
```



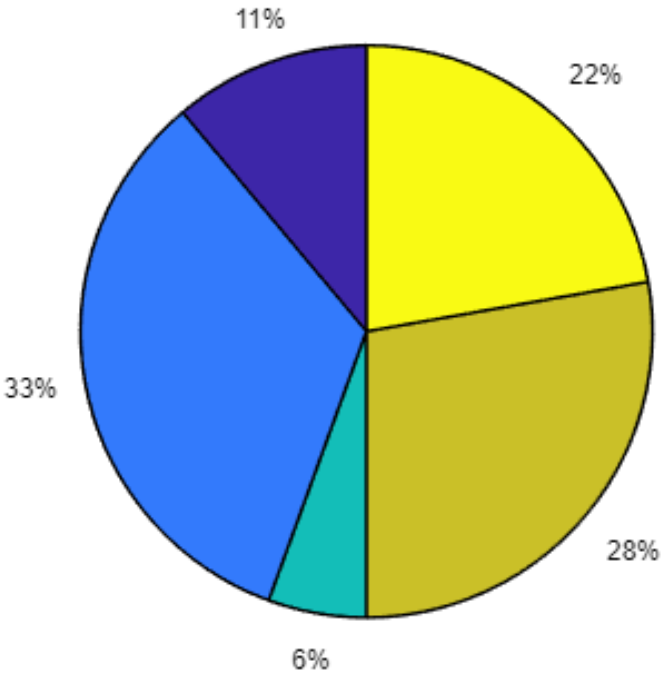
histogram(): histogram(X) creates a histogram bar chart of the elements in vector x.

```
x = randn(1000,1);

histogram(x)
```

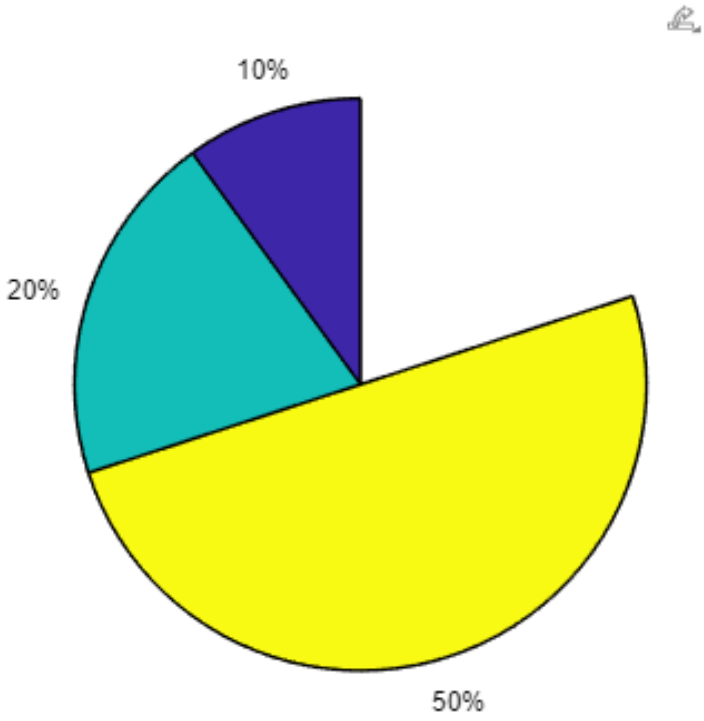


pie(): pie(X) draws a pie chart using the data in X. Each slice of the pie chart represents an element in X. If sum(X)≤1, then the values in X directly specify the areas of the pie slices. pie() draws only a partial pie if sum(X)<1. If sum(X)>1, then pie() normalizes the values by X/sum(X) to determine the area of each slice of the pie.

```
X = [1 3 0.5 2.5 2]; %sum(X)>1

pie(X)
```

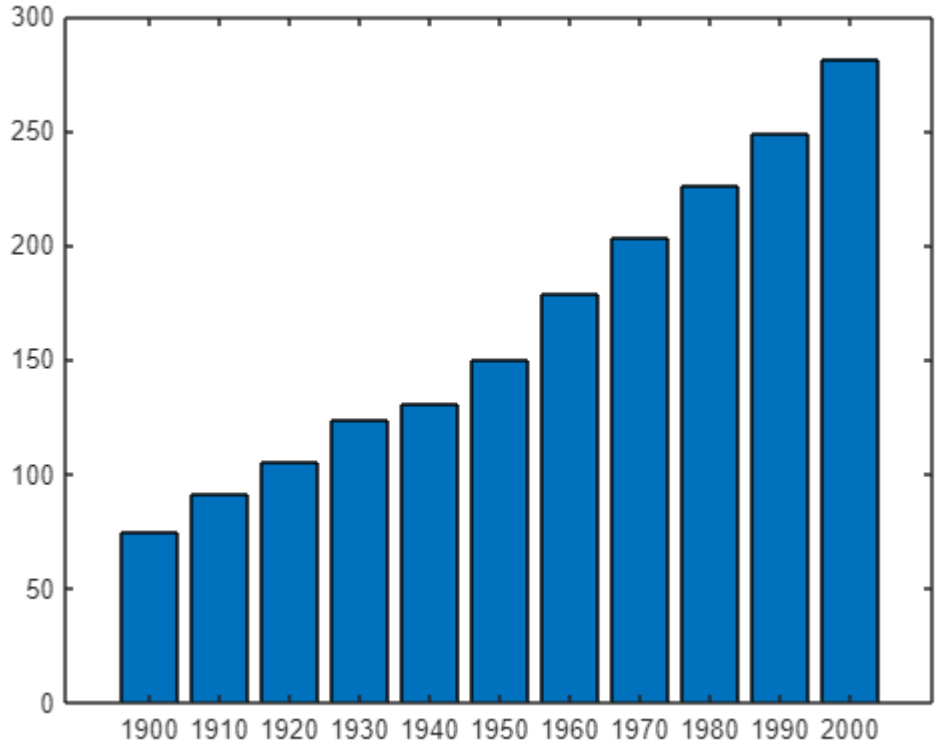



```
X = [0.1 0.2 0.5]; %sum(X)<1

pie(X)
```

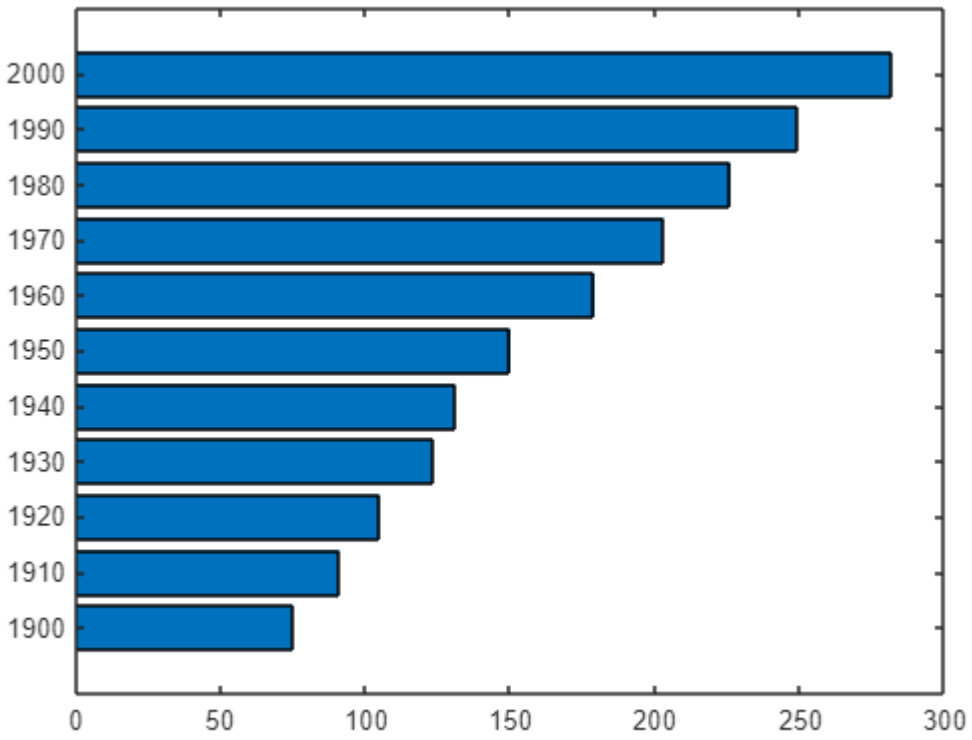


bar(): bar(X,Y) creates a bar graph with one bar for each element in y at the locations specified by x.

```
x = 1900:10:2000;
y = [75 91 105 123.5 131 150 179 203 226 249 281.5];

bar(x,y)
```

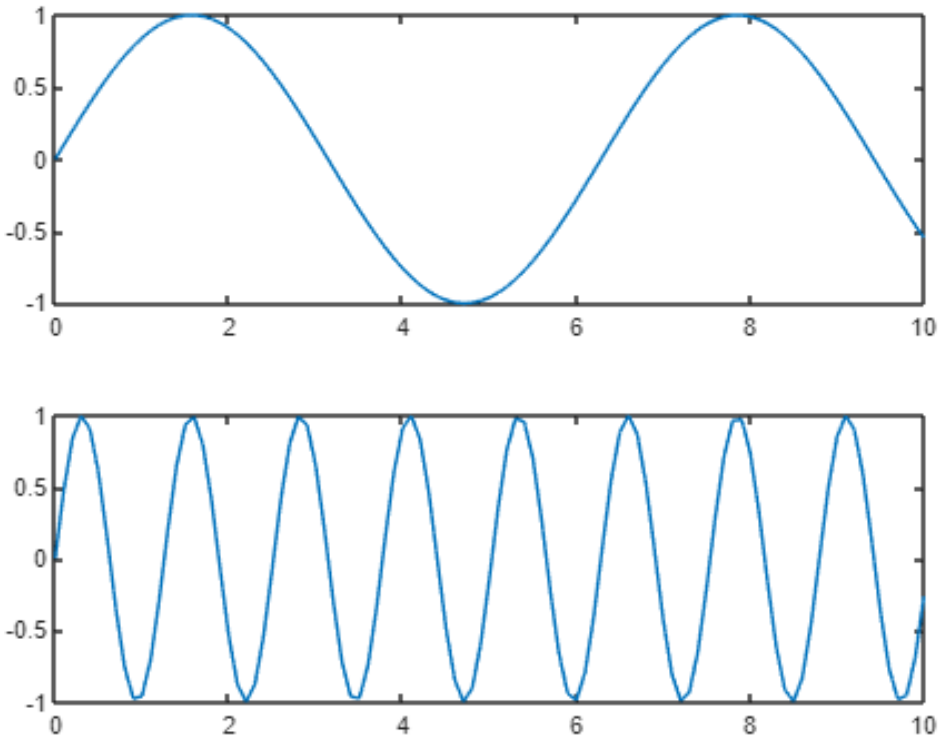

300
250
200
150
100
50
0
1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000

```
barh(x,y)
```


2000
1990
1980
1970
1960
1950
1940
1930
1920
1910
1900
0 50 100 150 200 250 300

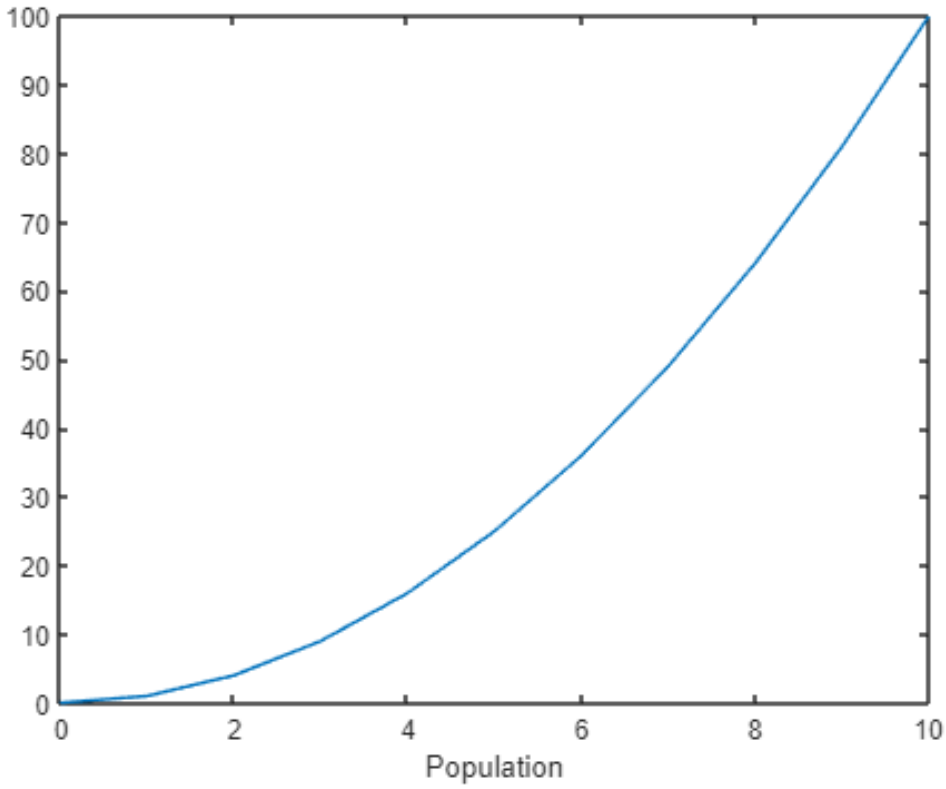
subplot(): subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB numbers the subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on.

```
x = 0:0.1:10;

y1 = sin(x);

subplot(2,1,1);
plot(x,y1)

y2 = sin(5*x);

subplot(2,1,2);
plot(x,y2)
```



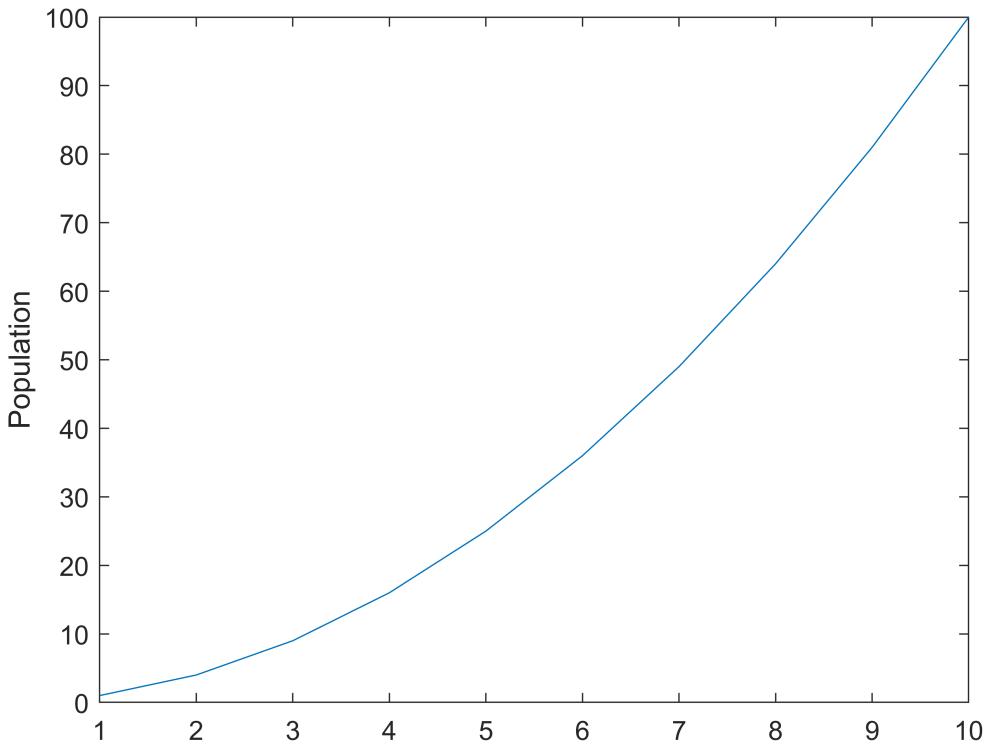
xlabel(): xlabel('txt') labels the x-axis of the current axes or chart.

```
x = 0:10;

figure; % Creates a new figure for plotting
plot(x, x.^2)
xlabel('Population')
```



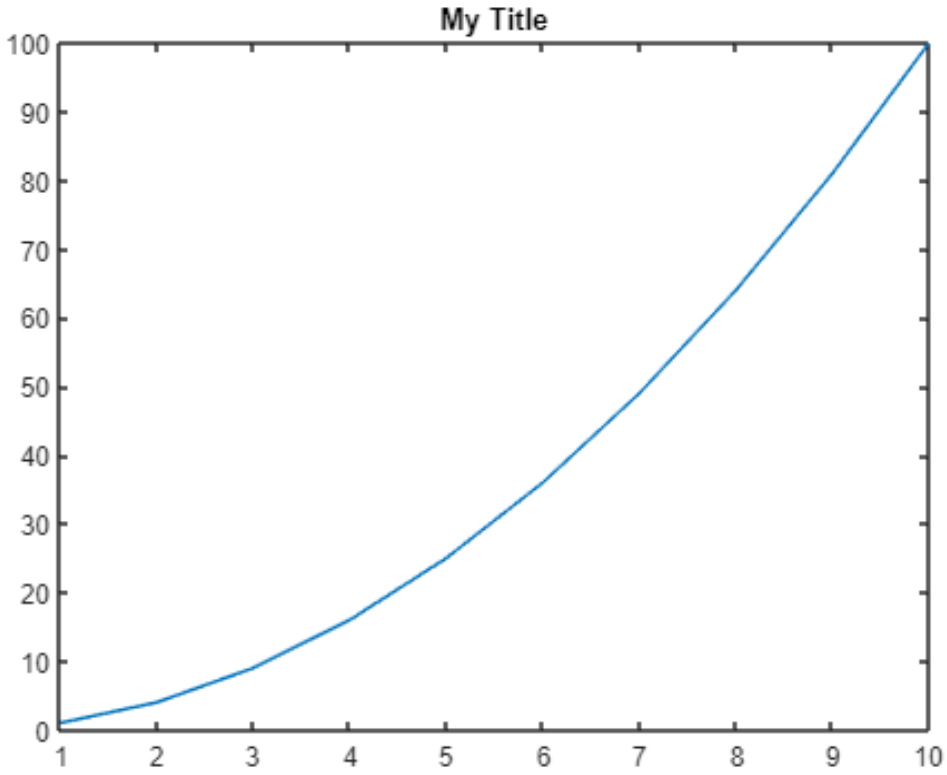
ylabel(): ylabel('txt') labels the y-axis of the current axes or chart.

```
x = 1:10;

figure;
plot(x, x.^2)
ylabel('Population')
```



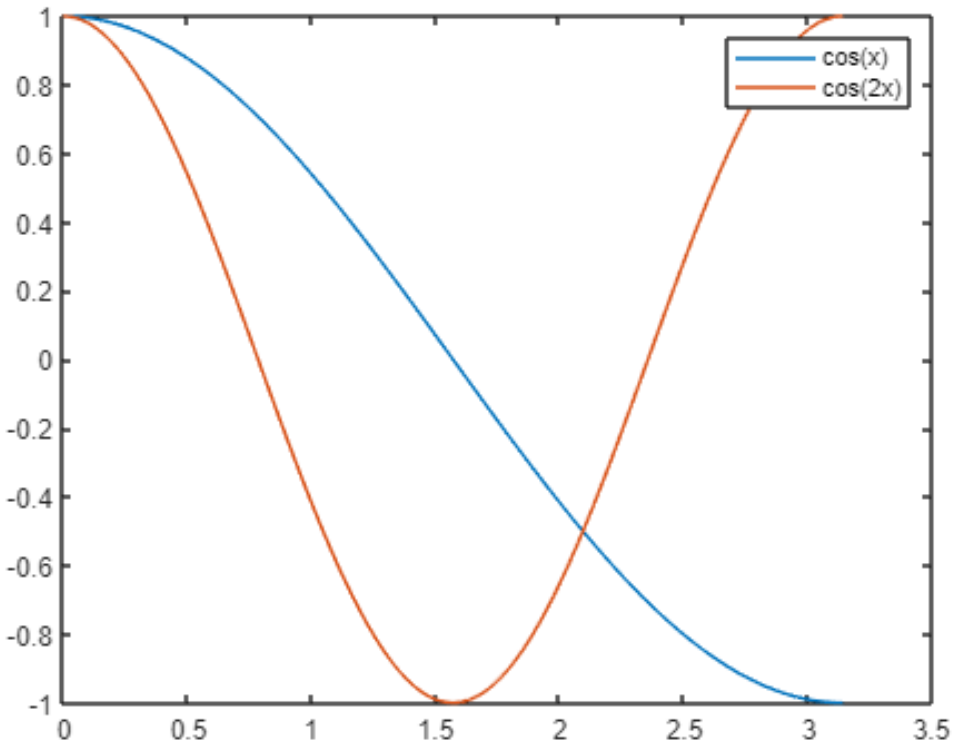
title(): title('txt') adds the specified title to the current axes or chart.

```
x = 1:10;

figure;
plot(x, x.^2)
title('My Title')
```



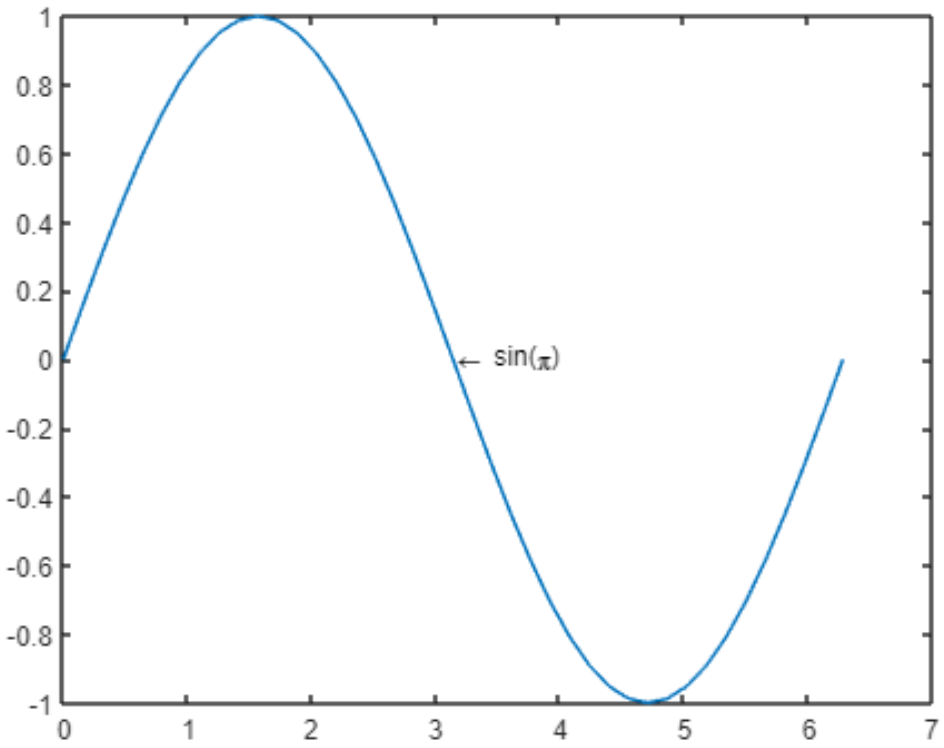
legend(): legend('label1',...,'labelN') creates a legend with descriptive labels for each plotted data series. For the labels, the legend uses the text from the DisplayName properties of the data series. If the DisplayName property is emptyi then the legend uses a label of the form 'dataN'. The legend automaticallt updates when you add or delete data series from the axes. This command creates a legend for the current axes or chart.

```
x = 0:pi/100:pi;

y1 = cos(x);
y2 = cos(2*x);

plot(x,y1)
hold on
plot(x,y2)
hold off

legend('cos(x)', 'cos(2x)')
```



text(): text(x,y,'txt') adds a text description to one or more data points in the current axes using the text specified by 'txt'. To add text to one point, specify x and y as scalars. To add text to multiple points, specify x and y as vectors with equal length.

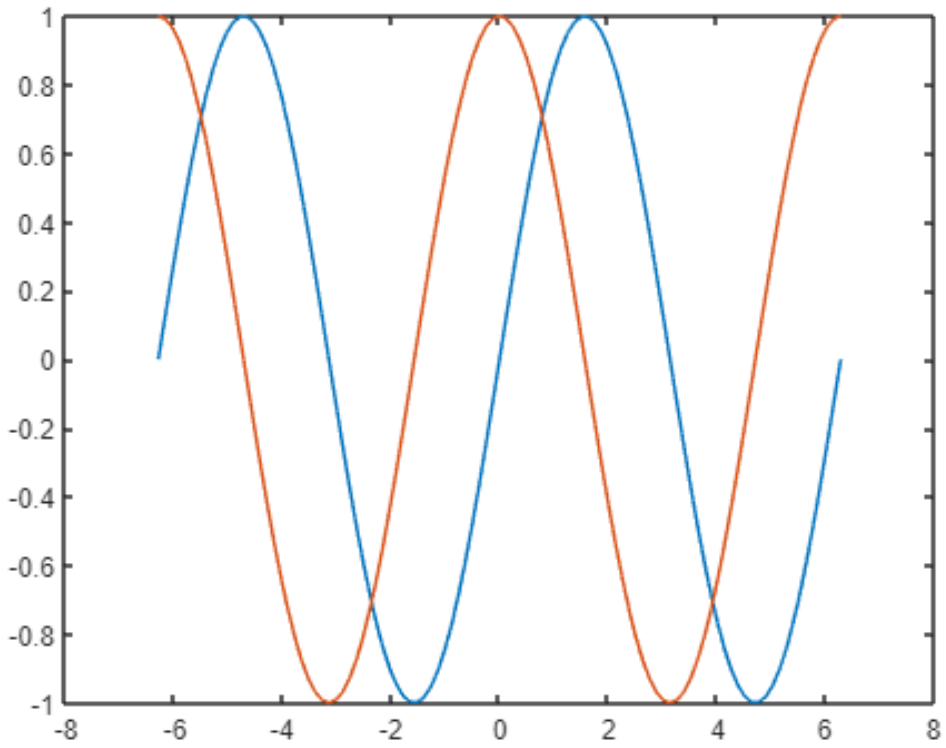
```
x = 0:pi/20:2*pi;
y= sin(x);

plot(x,y)
text(pi,0,'\leftarrow sin(\pi)')
```



Drawing Multiple Plots

Using plot(): plot(X1,Y1,...,Xn,Yn) plots multiple X, Y pairs using the same axes for all lines.

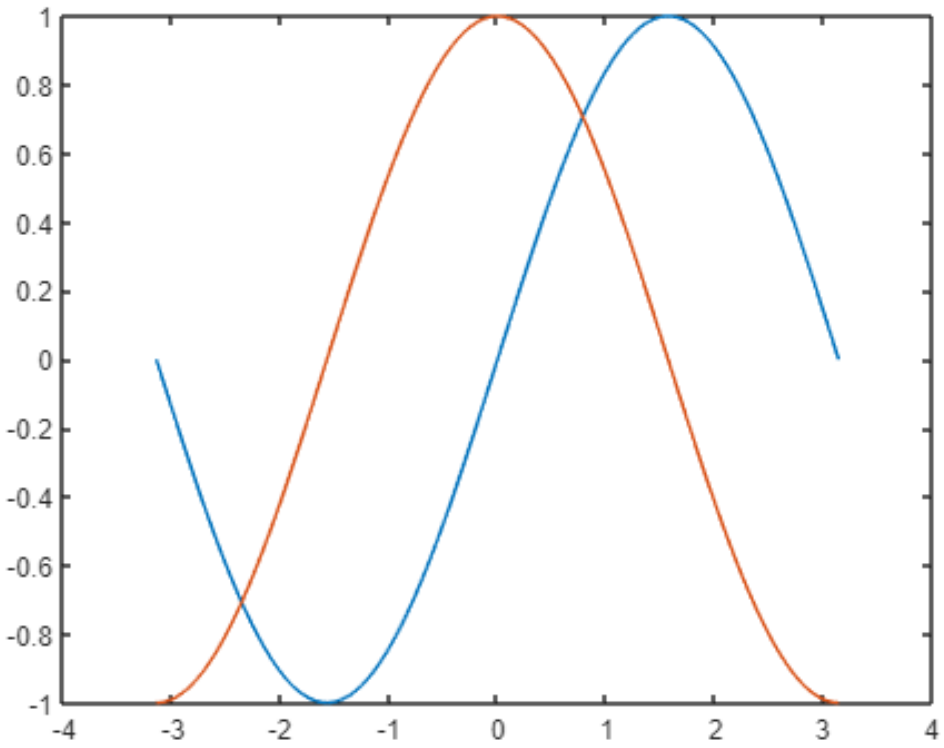
```
x = -2*pi:pi/25:2*pi;

y1 = sin(x);
y2 = cos(x);

figure;
plot(x,y1,x,y2)
```



Using hold __:

- hold on retains the plots in current axes so that new plots addes to the axes do not delete existing plots.
- hold off sets the hold state to off so that new plots added to the axes cleas existing plots and resets all axes properties.

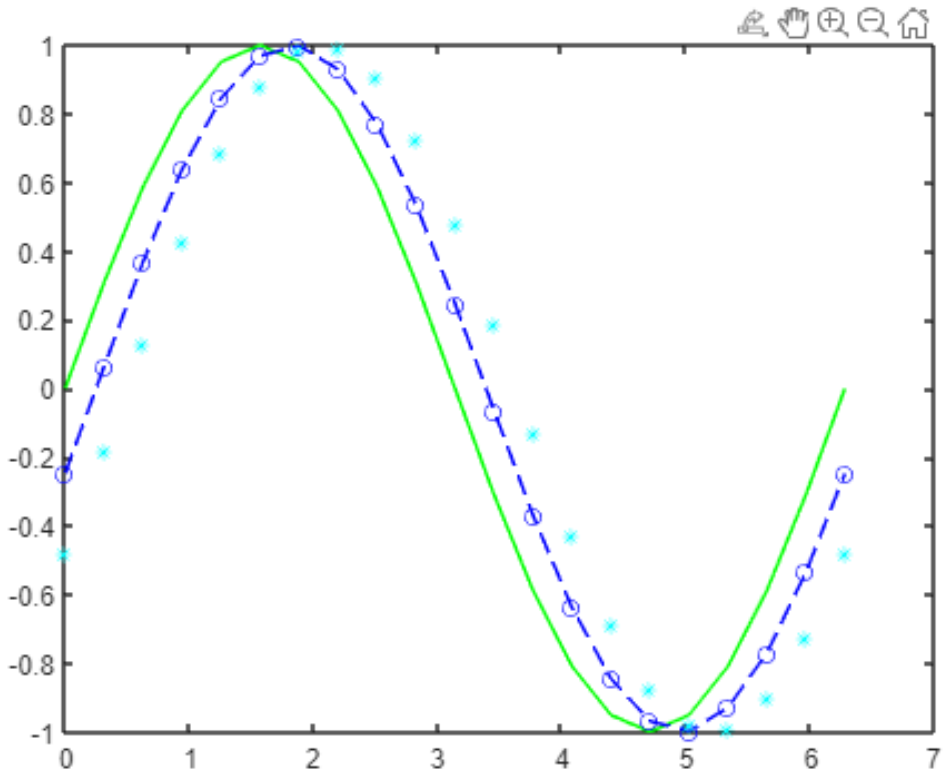
```
x = -pi:pi/50:pi;

y1 = sin(x);
y2 = cos(x);

plot(x,y1)
hold on
plot(x,y2)
hold off
```



Formatting Plot Graphs

plot(X1, Y1, LineSpec1,..., Xn, Yn, LineSpecN) sets the line style, marker type, and color for each line. You can mix X, Y, LineSpec triplets with X, Y pairs. For example, plot(X1, Y1, X2, Y2, LineSpec2, X3, Y3).

```
x = 0:pi/10:2*pi;

y1 = sin(x);
y2 = sin(x-0.25);
y3 = sin(x-0.5);

figure;
plot(x,y1,'g',x,y2,'b--o',x,y3,'c*')
```



Line style, marker, and color, are specified as a character vector or string containing symbols. You do not need to specify all three characteristics. For example, if you omit the line style and specify the marker, then the plot shows only the marker and no line.

Tables containing detailed line stylei marker and color information can be found here.

Recursive functions (1, 2)

Recursive functions are functions that call themselves directly or indirectly within their execution processes. Each recursive function requires a base condition for termination.

Example: Write a recursive function called "matryoshka()" which accepts a single positive integer for how many matryoshka dolls (nesting dolls, stacking dolls) are nested within. Then splits each doll starting from outside (from greatest number or number of nested dolls) towards inside (smallest number or 1) while displaying number for each doll.

```
function matryoshka(n)
fprintf('Doll number %d\n', n)

if n>1
    matryoshka(n-1)
end
end
```

```
matryoshka(5)
```

```
Doll number 5
Doll number 4
Doll number 3
Doll number 2
Doll number 1
```

PROBLEMS

1. Using load('data.mat') function load data saved into “data.mat” file. Then follow the steps below. Draw each graph in the specified positions in the 3 by 3 subplot area. Name all axes and give titles to each graph accordingly.

a. Create a plot graph for scores for all students in subplot position 1.

b. Create a stem graph for scores for all students in subplot position 2.

c. Create a histogram graph for all scores in subplot position 3.

d. Which graph above (a, b, c) gives meaningful information?

e. Find numbers of male and female students.

f. Create a pie graph for numbers of male and female students with legend in subplot position 4.

g. Create a bar graph for numbers of male and female students in subplot position 5. (Note: use categorize({'Male', 'Female'}) function for x axis.)

h. Find average scores of male and female students.

i. Create a bar graph for average scores of male and female students in subplot position 6. (Note: use categorize({'Male', 'Female'}) function for x axis.)

j. Which graph above (f, g, i) gives what kind of information?

k. Find numbers of students who enrolled in 2018 and 2019.

l. Create a pie graph for numbers of students who enrolled in 2018 and 2019 with legend in subplot position 7.

m. Create a bar graph for numbers numbers of students who enrolled in 2018 and 2019 in subplot position 8.

n. Find average scores of students who enrolled in 2018 and 2019.

o. Create a bar graph for average scores of students who enrolled in 2018 and 2019 in subplot position 9.

p. Which graph above (l, m, o) gives what kind of information?

2. Create a recursive function called “fact()” which accepts a single positive integer as input and calculates its factorial, then gives the result as output.

BME1901 - Introductory Computer Sciences

Laboratory Handout - 9

OBJECTIVES

Learn about;

- Cells and cell arrays
- Structures and structure arrays
- Opening, closing and reading files
- Built-in functions [isempty(), ischar(), strcmp(), input()]
- Solvinf various questions

TOOLS

Cells and Cell Arrays (1, 2, 3, 4)

A cell array is a data type with indexed data containers called cells, where each cell can contain any type of data. Cell arrays commonly contain either lists of character vectors of different lengths, or mixes of strings and numbers, or numeric arrays of different sizes. Refer to sets of cells by enclosing indices in smooth parantheses, (). Access the contents of cells by indexing with curly braces, {}.

Data can be put into a cell array using the cell array construction operator, {}.

```
C = {1, 2, 3; 'text', rand(3,2), {11;22;33}}
```

```
C = 2×3 cell
```

	1	2	3
1	1	2	3
2	'text'	[0.4898,...	3×1 cell

There are two ways to refer to elements of a cell array. Enclose indices in smooth parantheses, (), to refer to sets of cells -- for example, to define a subset of the cell array. Enclose indices in curly braces, {}, to refer to the text, numbers, or other data within individual cells.

```
D = C(1:2, 1:2)
```

```
D = 2×2 cell
```

	1	2
1	1	2
2	'text'	[0.4898,...

```
E = C(2,2)
```

```
E = 1×1 cell array
    {3×2 double}
```

```
F = C{2,2}
```

```
F = 3×2
    0.4898    0.0991
    0.1932    0.0442
    0.8959    0.5573
```

Structures and Structure Arrays (1, 2, 3, 4, 5, 6)

A structure array is a data type that groups related data using data containers called fields. Each field can contain any type of data. Access data in a structure using dot notation of the form "structName.fieldName".

Data can be out into a new structure by creating the structure using dot notation to name its fields one at a time.

Data can be accessed from a structure using the same dot notation.

```
st.aa = [1 2 3; 4 5 6];
st.bb = 'Hello World';
st.cc = 3;
st
```

```
st = struct with fields:
    aa: [2×3 double]
    bb: 'Hello World'
    cc: 3
```

```
A = st.aa
```

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

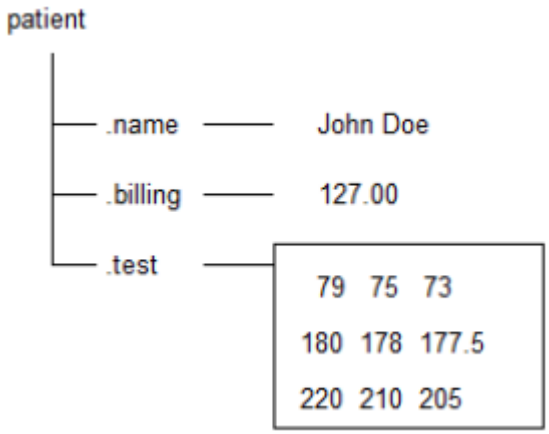
```
disp(st.bb)
```

```
Hello World
```

```
B = 2 + st.cc
```

```
B = 5
```

Example: How to store a patient record in a scalar structure with fields name, billing, and test.

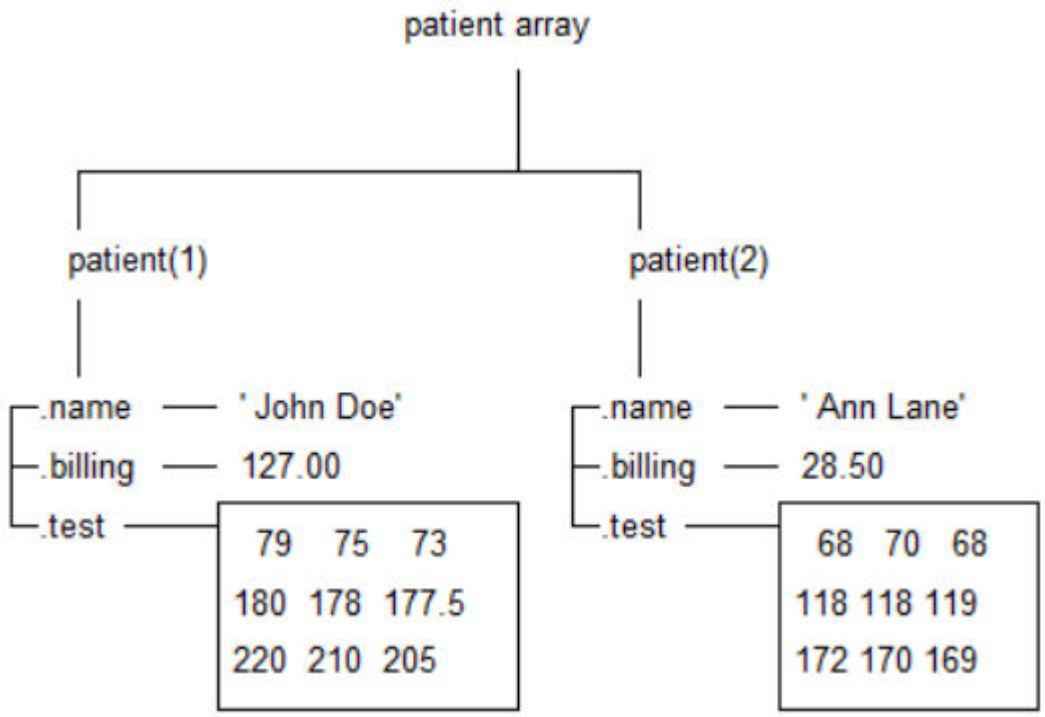


```
patient.name = 'John Doe';
patient.billing = 127.00;
patient.test = [79 75 73; 180 178 177.5; 220 210 205];
patient
```

```
patient = struct with fields:
       name: 'John Doe'
    billing: 127
       test: [3×3 double]
```

Example: How to add new patient records to the array by including subscripts after the array name.

-




```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
patient
```

```
patient = 1×2 struct
```

Fields	name	billing	test
1	'John Doe'	127	[79,75,7...
2	'Ann Lane'	28.5000	[68,70,6...

Each patient record in the array is a structure of class struct. An array of structures is often referred to as a struct array. Like other MATLAB arrays, a struct array can have any dimensions.

A struct array has the following properties;

- All structs in the array have the same number of fields.
- All structs have the same field names.
- Fields of the same name in different structs can contain different types or sizes of data.

Example: How to find the average of the test results of first and second patients.

```
aveResultsP1 = mean(patient(1).test)
```

```
aveResultsP1 = 1×3
  159.6667  154.3333  151.8333
```

```
aveResultsP2 = mean(patient(2).test)
```

```
aveResultsP2 = 1×3
  119.3333  119.3333  118.6667
```

```
for i = 1:1:size(patient,2)
    aveResults(i,:) = mean(patient(i).test);
end
aveResults
```

```
aveResults = 2×3
  159.6667  154.3333  151.8333
  119.3333  119.3333  118.6667
```

Opening, Closing and Reading Files

fopen(): fileID=fopen(filename) opens the file, filename, for binary read access. If fopen cannot open the file, then fileID is -1.

fileID = fopen(filename, permission) opens the file with the type of access specified by permission. File access type is specified as a character vector or a string scalar. You can open a file in binary mode or in text mode. To open a file in binary mode, specify one of the following.

'r'	Open file for reading. (default)
'w'	Open or create new file for writing. Discard existing contents, if any.
'a'	Open or create new file for writing. Append data to the end of the file.
'r+'	Open file for reading and writing.
'w+'	Open or create new file for reading and writing. Discard existing contents, if any.
'a+'	Open or create new file ofr reading and writing. Append data to the end of the file.
'A'	Open file for appending without automatic flushing of the current output buffer.
'W'	Open file for writing without automatic flushing of the current output buffer.

fgetl(): tline = fgetl(fileID) returns the next line of the specified file, removing the newline characters. Consecutive calls of the fgetl() function returns consecutive lines of characters. If the file is nonempty, then fgetl returns tline as a character vector. If the file is empty and contains only the end-of-file marker, then fgetl returns tline as a numeric value -1.

fclose(): fclose(fileID) closes an open file identified with fileID.

```
Oranges and lemons,
```

```
Pineapples and tea.
Orangutans and monkeys,
Dragonflys or fleas.
```

```
fid = fopen('BadPoem.txt');
fgetl(fid)
```

```
ans =
'Oranges and lemons,'
```

```
fgetl(fid)
```

```
ans =
'Pineapples and tea.'
```

```
fgetl(fid)
```

```
ans =
'Orangutans and monkeys,'
```

```
fgetl(fid)
```

```
ans =
'Dragonflys or fleas.'
```

```
fgetl(fid)
```

```
ans = -1
```

```
fclose(fid);
```

Built-in Functions

isempty(): TF = isempty(A) returns logical 1 (true) if A is empty, and logical 0 (false) otherwise. An empty array, table, or timetable has at least one dimension with length 0, such as 0-by-0 or 0-by-5.

```
A = zeros(0,5);
TF = isempty(A)
```

```
TF = logical
   1
```

```
B = zeros(2,5);
TF = isempty(B)
```

```
TF = logical
   0
```

ischar(): tf = ischar(A) returns logical 1 (true) if A is a character array and logical 0 (false) otherwise.

```
A = 'Mary Jones';
tf = ischar(A)
```

```
tf = logical
   1
```

```
B = rand(1,3);
tf = ischar(B)
```

```
tf = logical
   0
```

strcmp(): tf = strcmp(s1, s2) compares character vectors s1 and s2 and returns 1 (true) if the two are identical, and 0 (false) otherwise.Text is considered identical if the size and content of each are the same. The return result tf is of data type logical. The input arguments can me any combination of string arrays, character vectros, and cell arrays of character vectors.

```
s1 = 'Yes';
s2 = 'No';
tf = strcmp(s1,s2)
```

```
tf = logical
   0
```

```
s1 = 'Yes';
s2 = 'Yes';
tf = strcmp(s1,s2)
```

```
tf = logical
   1
```

```
s1 = 'upon';
s2 = {'Once', 'upon', 'a', 'time'};
tf = strcmp(s1,s2)
```

```
tf = 1×4 logical array
   0   1   0   0
```

input(): str = input(prompt, 's') displays the text in prompt and waits for the user to input a value and press the Enter key. This expression returns the entered text, without evaluating the input as an expression.

```
x = input('What is your name?')
```

```
x = input('What is your name?\n', 's')
```

```
x =
'John'
```

PROBLEMS

1. You are working in a software development team for a hospital and your team is assigned with the development of a patient check-in and check-out program. You are given the base part of the program as in the box below. Using indexed structure arrays write the functions 'checkIn()', 'checkOut()', and 'printInventory()'.

```
function patients = patientHospital()

disp('Generating a structure for storing patients hospital information');
patients = [];
```

```
while(1)
    choice = input('Press I for check-In, O for Check-Out and Q to quit: ', 's');

    if choice == 'I'
        patients = checkIn(patients); % Function explained in (a)

    elseif choice == 'O'
        patient_name = input('Enter the name of the patient for check-out: ', 's');
        [patients, bill] = checkOut(patients, patient_name); % Function explained in (b)
        disp(['The bill is ', num2str(bill), ' TL']);

    elseif choice == 'Q'
        return;
    end

    printInventory(patients); % Function explained in (c)
end
```

a. For the “checkIn()” function there should be an input and an output both being the structName called “patients” (i.e.: database). Under this structName, there should be fieldNames called “name, gender, age, blood_type, days_in_hospital, mobile_phone, intensive_care, anmbulance_come. For each patient checking in, these fields should be requested as an input from the user. (Note: this function should add new patients while keeping the data from before.)

- “name” should be a character array user input.
- “gender” should be a character array user input.
- “age” should be a numeric user input.
- “blood_type” should be a character array user input.
- “days_in_hospital” should be a numeric user input.
- “mobile_phone” should be a character array user input.
- “intensive_care” should be a character array user input, asking the user to input “IC” for intensive care patients and “NIC” for non-intensive care patients.
- “ambulance_come” should be a character array user input, asking the user to input “WA” for the patients coming in with an ambulance and “NA” for the patients do not come with an ambulance.

b. For the “checkOut()” function there should be two inputs and two outputs. The inputs should be the structName “patients” and a single patient name who is checking out. The outputs should be the structName “patients” and the bill of the patient.

- The code should search for the patient checking out in the list of patients. If the pateint is not on the list, then the program should exit this function with an error.
- If the patient is present, then the program should calculate their bill.
- Each day in the hospital costs 1000 TL.
- If the patient stayed in intensive care, their bill should be doubled.
- If the patient came to the hospital with an ambulance, their bill should be halved.
- At the end that patient and their information should be deleted from the database (i.e.: structName “patients”).

c. For the function “printInventory()” the input should be the structName “patients”, and there sould be no output. The program should display the names of the patients in the database (i.e.: structName “patients”) on the screen individually.

2. Write a function file called “copyText()” that request and returns no output. The first input should be the txt file to be read and the second input should be the txt file to be written. The code should scan the first file line by line and should print the same text to the second file. Create a txt file called “Poem.txt” and put the poem below Beklenen – Necip Fazıl KISAKÜREK in it.

```
Ne hasta bekler sabahi,
Ne taze oluyu mezar.
Ne de seytan, bir gunahi,
Seni bekledigim kadar.
```

BME1901 - Introductory Computer Sciences

Laboratory Handout - 10

OBJECTIVES

Learn about;

- MATLAB Live-Scripts
- table arrays
- Importing data to workspace
- Data visualization
- Built-in functions

TOOLS

MATLAB Live-Scripts

Live scripts are program files that contain your code, output, and formatted text together in a single interactive environment called the Live Editor. In live scripts, you can write your code and view the generated output and graphics along with the code that produced it.

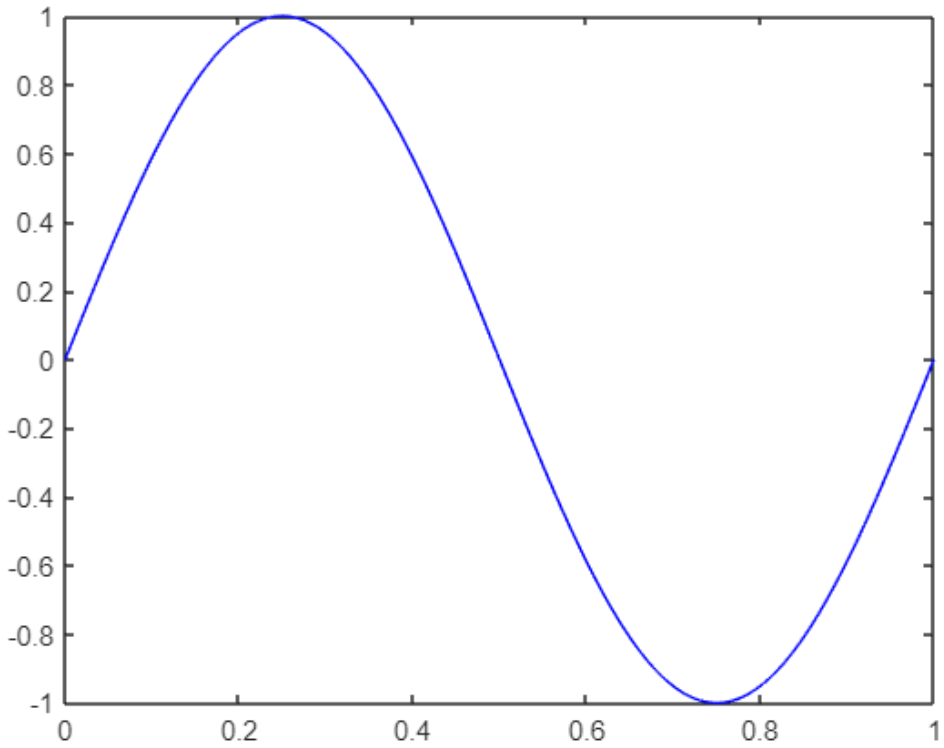
You can switch between code and text using the Live Editor section in the toolbar. Outputs generated by code will be displayed below the code cells (this can be changed using the buttons to the right of the live editor). You can use section breaks to divide the script into relevant parts.

You can add interactivity to your live-scripts using Insert->Code->Control. This menu will allow you to add interactive controls such as sliders, input fields or checkboxes to your live-script.

Example:

Below code generates a sinewave with editable frequency and duration, then displays the wave using the plot() function with selected line color.

```
f =1;
fs = 5090;
duration = 1;
color = 'b';
linestyle = '-';
marker = '';

linspec = [color, linestyle, marker];
n = 0:1:(fs*duration);
y = sin(2*pi*f*n/fs);

plot(n/fs,y,linspec)
```



Tables

table arrays store column-oriented or tabular data, such as columns from a text file or spreadsheet. Tables store each piece of column-oriented data in a *variable*. Table variables can have different data types and sizes as long as all variables have the same number of rows. Table variables have names, just as the fields of a structure have names.

You can create an empty table to using the command below. Here sz is an array containing the size of each dimension, and varTypes is an array describing the variable type of each column.

```
T = table('Size', sz, 'VariableTypes', varTypes)
```

Example: Create an empty table of size [50 5]. Columns should hold data types double, string, datetime, double, double in order.

```
varTypes = {'double','string','datetime','double','double'};
sz = [10 5];
T = table('Size', sz, 'VariableTypes', varTypes)
```

T = 10×5 table

	Var1	Var2	Var3	Var4	Var5
1	0	<missing>	NaT	0	0
2	0	<missing>	NaT	0	0
3	0	<missing>	NaT	0	0

	Var1	Var2	Var3	Var4	Var5
4	0	<missing>	NaT	0	0
5	0	<missing>	NaT	0	0
6	0	<missing>	NaT	0	0
7	0	<missing>	NaT	0	0
8	0	<missing>	NaT	0	0
9	0	<missing>	NaT	0	0
10	0	<missing>	NaT	0	0

Tables can be created from existing data using the command below. Here var1,...,varN holds column vectors of same length.

```
T = table(var1,...,varN)
```

Example: Create a table using patients.mat

```
load patients.mat
Height = round(Height * 2.54); % Convert inches to cm
Weight = round(Weight * 0.453); % Convert lbs to kg

T_patients = table(LastName,Location,Age,Gender,Height,Weight,Smoker,Systolic,Diastolic,SelfAss
```

Accessing Data in Tables

Smooth parantheses (): Returns a table that has selected rows and variables.

```
T_patients(2:4,["LastName","Age"])
```

```
ans = 3×2 table
```

	LastName	Age
1	'Johnson'	43
2	'Williams'	38
3	'Jones'	40

Curly braces {}: Returns an array concatenated from the contents of selected rows and variables. When using this notation, variables should be of the same datatype.

```
T_patients{2:4,["Systolic","Diastolic"]}
```

```
ans = 3×2
   109    77
   125    83
   117    75
```

Dot notation: Returns the contents of a variable as an array.

```
T_patients.LastName
```

```
ans = 100×1 cell
'Smith'
'Johnson'
'Williams'
'Jones'
'Brown'
'Davis'
'Miller'
'Wilson'
'Moore'
'Taylor'
    ⋮
```

Adding New Variables to a Table

You can use dot notation to add new variables to a table. The added variable need to have the same number of rows as the table. New variables can be derived from existing variables.

```
T.newVar = colVec
```

Example: Using weight and height data of each patient, calculate and create a new table variable called 'BMI'.

```
T_patients.randNew = rand(100,1);
T_patients
```

```
T_patients = 100×11 table
```

...

	LastName	Location	Age	Gender	Height	Weight
1	'Smith'	'County General Hospital'	38	'Male'	180	80
2	'Johnson'	'VA Hospital'	43	'Male'	175	74
3	'Williams'	'St. Mary's Medical Center'	38	'Female'	163	59
4	'Jones'	'VA Hospital'	40	'Female'	170	60
5	'Brown'	'County General Hospital'	49	'Female'	163	54
6	'Davis'	'St. Mary's Medical Center'	46	'Female'	173	64
7	'Miller'	'VA Hospital'	33	'Female'	163	64
8	'Wilson'	'VA Hospital'	40	'Male'	173	82
9	'Moore'	'St. Mary's Medical Center'	28	'Male'	173	83
10	'Taylor'	'County General Hospital'	31	'Female'	168	60
11	'Anderson'	'County General Hospital'	45	'Female'	173	58
12	'Thomas'	'St. Mary's Medical Center'	42	'Female'	168	62
13	'Jackson'	'VA Hospital'	25	'Male'	180	79
14	'White'	'VA Hospital'	39	'Male'	183	92

⋮

Filtering Data in a Table

Data can be filtered relational oparations using the following notation.

```
idx = (conditions);
cols = (variables)

newTable = Table(idx,cols);
```

Example: Create a new table using T_patients that contains smoker patients aged between 25-45, and contains variables 'LastName', 'Age', 'Smoker', 'BMI'.

```
idx = T_patients.Age <=45 & T_patients.Age >= 25 & T_patients.Smoker == 1;
cols = {'LastName', 'Age', 'Smoker', 'BMI'};

newTable = T_patients(idx,cols)
```

newTable = 29×4 table

	LastName	Age	Smoker	BMI
1	'Smith'	38	1	24.6914
2	'Miller'	33	1	24.0882
3	'White'	39	1	27.4717
4	'Thompson'	32	1	28.4082
5	'Garcia'	27	1	19.2653
6	'Lee'	44	1	23.3844
7	'Walker'	28	1	20.5693
8	'King'	30	1	29.0657
9	'Wright'	45	1	17.9902
10	'Baker'	44	1	26.8519
11	'Nelson'	33	1	29.0533
12	'Mitchell'	39	1	22.8395
13	'Roberts'	44	1	24.3025
14	'Turner'	37	1	27.7743

⋮

Importing Data

Import tool can be used to preview and import data from files such as spreadsheets or delimited text files. Whole or a selected portion of the data can be imported using this tool.

To use the import tool, go to HOME->VARIABLE->Import Data and select the file that you want to import. In the window that opens you can select which columns or rows that you want to import and the data types for the columns. Code that imports selected data can also be generated using this tool for repeated uses.

Plotting Using Table View

Various plots can be easily generated using the table viewer. After loading the table to the workspace, open the loaded table by double clicking on the workspace. Under the 'PLOT' tab in the opened window, you can select various plots that can be drawn with the selected data. To be able to use this tool, you need to select the columns in the correct order for the plot to be drawn. After drawing a plot, the code that creates the selected plot can be found in the command window.

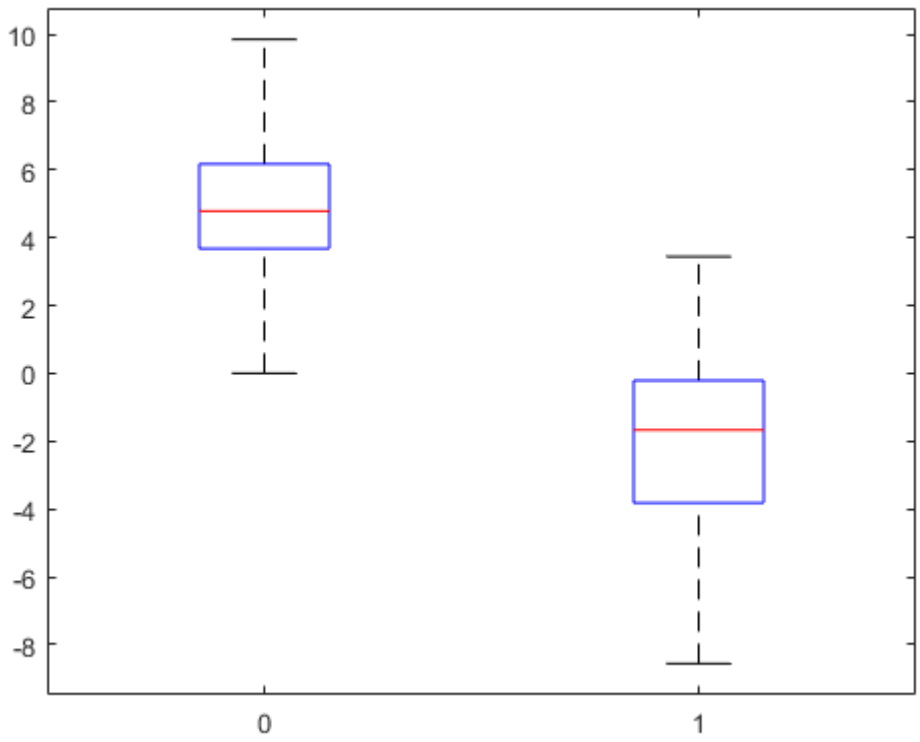
Plotting Functions

Boxplots

boxplot(x,g) creates a boxplot using one or more grouping variables contained in g. boxplot() produces a seperate box for each set of x values that share the same g value. Boxplots are useful for visualizing summary statistics, or distributions of different groups.

Example:

```
x1 = 2*randn(100,1)+5;
x2 = 3*randn(100,1)-2;
x = [x1;x2];
g = [zeros(100,1);ones(100,1)];

boxplot(x,g)
```

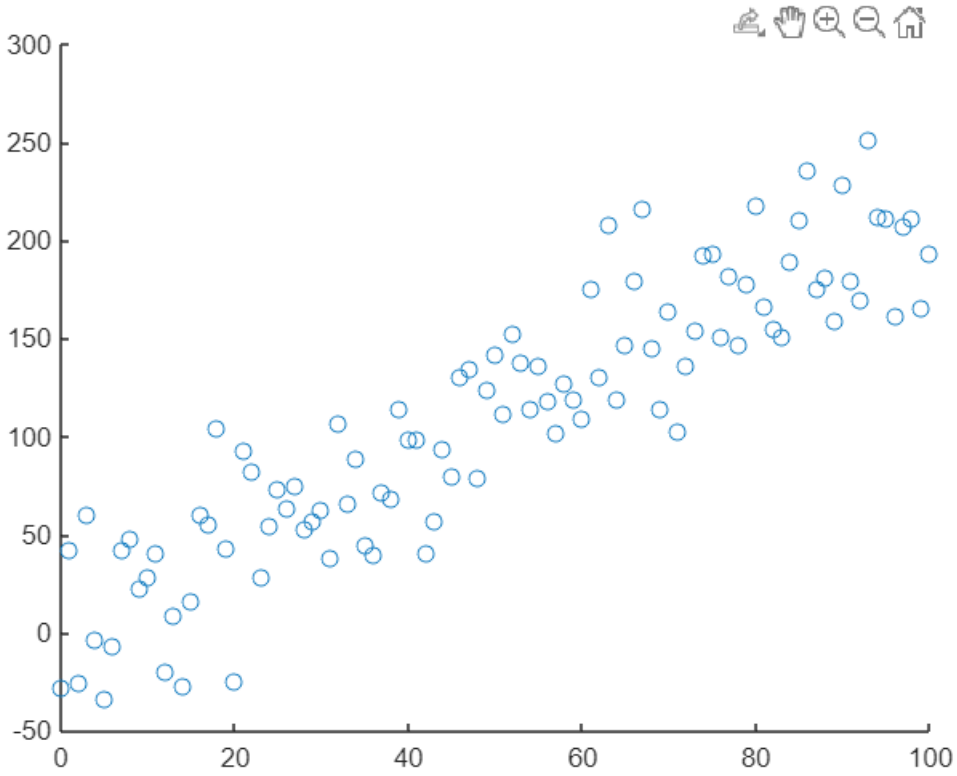


Scatter Plots

scatter(x,y) creates a scatter plot with markers at the locations specified with vectors x and y. This is useful for visualizing the relationship between two numerical variables.

Example:

```
x = 0:100;
y = 2*(x + 15*randn(1,length(x)));

scatter(x,y)
```



Built-in Functions

summary(): summary(T) prints a summary of the input table.

isnan(): TF = isnan(A) returns a logical array containing 1 where elements of A are NaN and 0 where they are not.

```
A = [NaN 5 15 NaN 2 NaN];

TF = isnan(A)
```

```
TF = 1×6 logical array
   1   0   0   1   0   1
```

ismissing(): TF = ismissing(A) returns a logical array that indicates which elements of the input data contain missing values.

```
A = [NaN 5 15 NaN 2 NaN];

TF = ismissing(A)
```

```
TF = 1×6 logical array
   1   0   0   1   0   1
```

any(): any(A,dim) tests whether at least one element of A returns logical 1 along the dimension specified by dim.

```
A = [0 0 0 0; 1 0 0 0; 0 1 1 0; 0 0 0 0]
```

```
A = 4×4
     0     0     0     0
     1     0     0     0
     0     1     1     0
     0     0     0     0
```

```
any(A,2)
```

```
ans = 4×1 logical array
   0
   1
   1
   0
```

corr(): [rho, pval] = corr(X,Y) returns a matrix of the pairwise linear correlation coefficient and p-values for testing the null hyphothesis between each pair of the columns in matrices X and Y.

```
X = randn(30,4);
Y = randn(30,4);
Y(:,4) = 2*X(:,4); % introduce correlation between columns of X and Y
Y(:,1) = -2*X(:,1);

[rho, pval] = corr(X,Y)
```

```
rho = 4×4
   -1.0000   -0.0638   -0.1247    0.0981
    0.1647    0.0799   -0.1403    0.3264
    0.1050    0.2262    0.1355    0.1294
   -0.0981    0.0605   -0.2187    1.0000
pval = 4×4
    0.0000    0.7378    0.5115    0.6062
    0.3843    0.6747    0.4598    0.0783
    0.5810    0.2293    0.4752    0.4956
    0.6062    0.7506    0.2456         0
```

mean(): M = mean(A) returns the mean of the elements of A along the first array dimension whose size does not equal 1.

std(): S = std(A) returns the standard deviation of the elements of A along the first array dimension whose size does not equal 1.

```
A = 2*(randn(100,1)) + 1;
```

```
M = mean(A)
```

```
M = 1.0159
```

```
S = std(A)
```

```
S = 2.2869
```

PROBLEMS

1) Load and print the summary of the data given in data.mat solve the problems given below. This data contains a modified version of patients.mat.

```

```

a) Find all entries with missing data in the table and display the results. Replace missing data with the mean of the appropriate variable.

```

```

b) Create a new variable called BMI using Weight and Height variables. Formula for BMI calculation is BMI = Weight / (Height^2).

```

```

c) Calculate the correlation between given variables below and visualize their relationship.

- Age vs BMI

```

```

- Height vs BMI

```

```

- Weight vs BMI

```

```

d) Compare the distributions of the variable-group pairs given below.

- BMI - Smoker

- BMI - SelfAssessedStatus

- Age - SelfAssessedStatus

- Systolic - Smoker

- Diastolic - Smoker

e) Create a new variable called BMI_norm by applying normalization (z-score) to BMI variable. Use this new variable to detect if there are any outliers the data. (|3| is considered cut-off for z-score).

Z = (xi - mean) / std