MATLAB – HW2 Upload Date: 6 Nov. 2020

Dorreen Rostami – 97243034
Zahra Hashemi – 97243072

2-1)

- Function *cylinder* calculates the volume of a cylinder using the 2 input arguments *height* & *radius* and returns the result in *volume*.
- Function *cylinder2* calculates the volume & area of a cylinder using the 2 input arguments *height* & *radius* and returns the results in *area* & *volume*.

Ways of defining a function in MATLAB:

- **Anonymous Functions:**
  An anonymous function is a function that is not stored in a program file, but is associated with a variable whose data type is function_handle. Anonymous functions can accept inputs and return outputs, just as standard functions do. However, they can contain only a single executable statement.
  For example, create a handle to an anonymous function that finds the square of a number:
  `sqr = @(x) x.^2;`
  Variable sqr is a function handle. The @ operator creates the handle, and the parentheses () immediately after the @ operator include the function input arguments. This anonymous function accepts a single input x, and implicitly returns a single output, an array the same size as x that contains the squared values. To find the square of a particular value (5): `a = sqr(5)`
- **Nested Functions:**
  A nested function is a function that is completely contained within a parent function.
  The primary difference between nested functions and other types of functions is that they can access and modify variables that are defined in their parent functions. As a result:
  - Nested functions can use variables that are not explicitly passed as input arguments.
  - In a parent function, you can create a handle to a nested function that contains the data necessary to run the nested function.

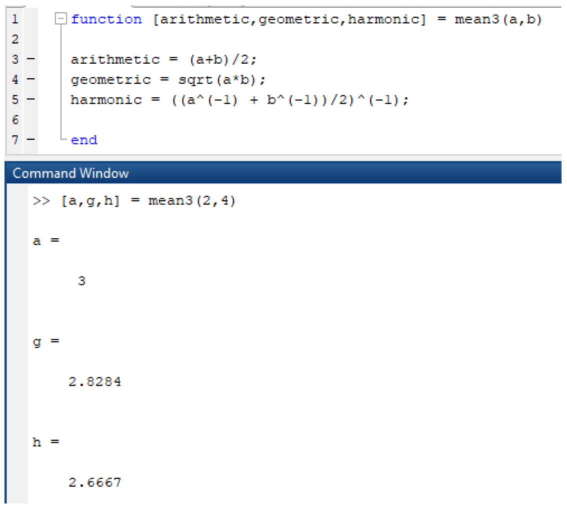
```
function parent
disp('This is the parent function')
nestedfx

    function nestedfx
        disp('This is the nested function')
    end

end
```

- **Extrinsic Functions:**
  When processing a call to a function foo, the code generator finds the definition of foo and generates code for its body. You can use `coder.extrinsic('foo')` to declare that calls to foo do not generate code and instead use the MATLAB engine for execution. In this context, foo is referred to as an extrinsic function. This functionality is available only when the MATLAB engine is available in MEX functions or during coder.const calls at compile time.

For unsupported functions other than common visualization functions (such as plot, disp, and figure), you must declare the functions to be extrinsic. Extrinsic functions are not compiled, but instead executed in MATLAB during simulation.
To declare a MATLAB function to be extrinsic, add the coder.extrinsic construct at the top of the main function or a local function:
coder.extrinsic('function_name_1', ... , 'function_name_n');

❖ Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores.

2-2)



```
function [arithmetic,geometric,harmonic] = mean3(a,b)

arithmetic = (a+b)/2;
geometric = sqrt(a*b);
harmonic = ((a^(-1) + b^(-1))/2)^(-1);

end
```

Command Window

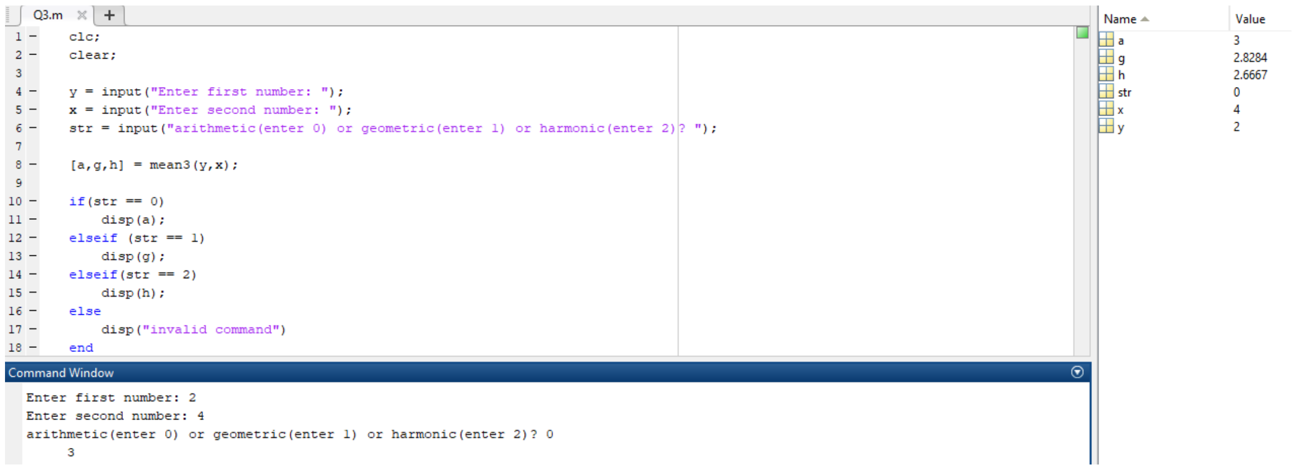
```
>> [a,g,h] = mean3(2,4)

a =

     3


g =

    2.8284


h =

    2.6667
```

a = (2+4)/2 = 3

g = sqrt(2*4) = 2.8284

h = ((1/2 + 1/4)/2)^(-1) = 8/3 = 2.6667

2-3)



Q3.m

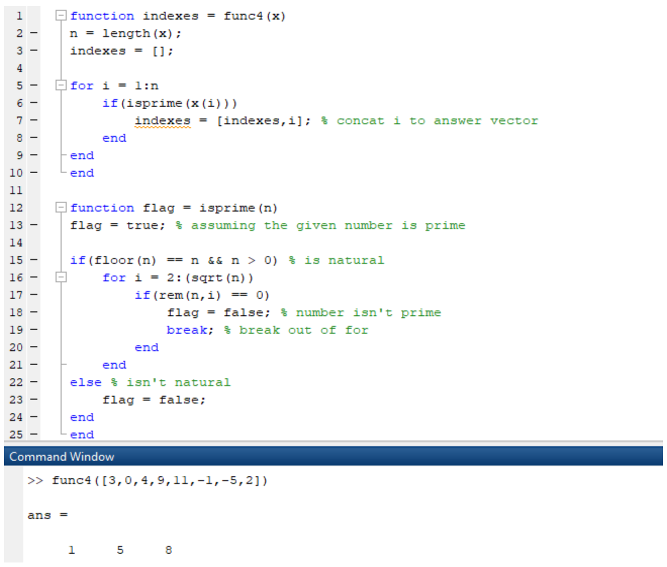
```
clc;
clear;

y = input("Enter first number: ");
x = input("Enter second number: ");
str = input("arithmetic(enter 0) or geometric(enter 1) or harmonic(enter 2)? ");

[a,g,h] = mean3(y,x);

if(str == 0)
    disp(a);
elseif (str == 1)
    disp(g);
elseif(str == 2)
    disp(h);
else
    disp("invalid command")
end
```

Command Window

```
Enter first number: 2
Enter second number: 4
arithmetic(enter 0) or geometric(enter 1) or harmonic(enter 2)? 0
     3
```

Name	Value
a	3
g	2.8284
h	2.6667
str	0
x	4
y	2

2-4)

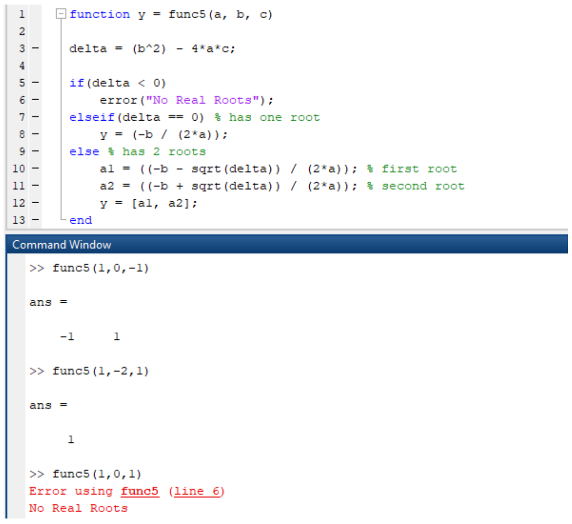


```
function indexes = func4(x)
n = length(x);
indexes = [];

for i = 1:n
    if(isprime(x(i)))
        indexes = [indexes,i]; % concat i to answer vector
    end
end
end

function flag = isprime(n)
flag = true; % assuming the given number is prime

if(floor(n) == n && n > 0) % is natural
    for i = 2:(sqrt(n))
        if(rem(n,i) == 0)
            flag = false; % number isn't prime
            break; % break out of for
        end
    end
else % isn't natural
    flag = false;
end
end
```

Command Window

```
>> func4([3,0,4,9,11,-1,-5,2])

ans =

     1     5     8
```

2-5)



```
function y = func5(a, b, c)

delta = (b^2) - 4*a*c;

if(delta < 0)
    error("No Real Roots");
elseif(delta == 0) % has one root
    y = (-b / (2*a));
else % has 2 roots
    a1 = ((-b - sqrt(delta)) / (2*a)); % first root
    a2 = ((-b + sqrt(delta)) / (2*a)); % second root
    y = [a1, a2];
end
```

Command Window

```
>> func5(1,0,-1)

ans =

    -1     1

>> func5(1,-2,1)

ans =

     1

>> func5(1,0,1)
Error using func5 (line 6)
No Real Roots
```

x^2=1 => 1*x^2 - 0*x - 1=0

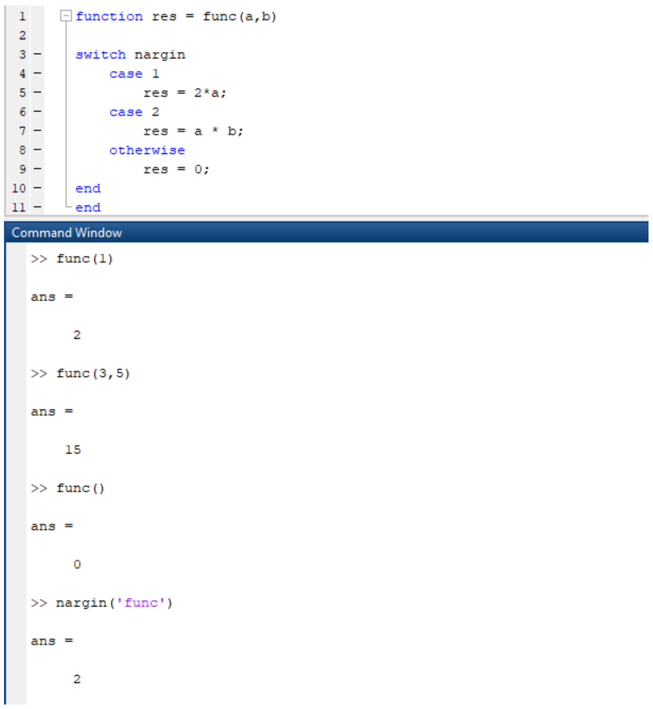
(x-1)^2=0 => 1*x^2 – 2*x + 1 = 0

x^2=-1 => 1*x^2 - 0*x + 1=0

2-6)

	Syntax	Description
nargin	nargin	nargin returns the number of function input arguments given in the call to the currently executing function. Use this syntax in the body of a function only.
	nargin(func)	nargin(fun) returns the number of input arguments that appear in the fun function definition. If the function includes varargin in its definition, then nargin returns the negative of the number of inputs.

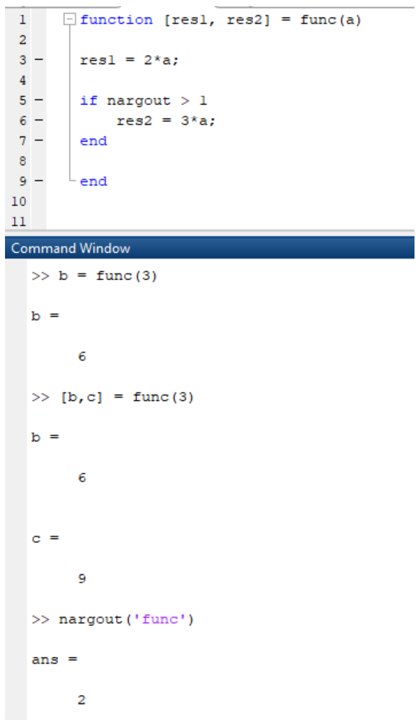
	Syntax	Description
nargout	nargout	nargout returns the number of function output arguments specified in the call to the currently executing function. Use this syntax in the body of a function only.
	nargout(func)	nargout(fun) returns the number of outputs that appear in the fun function definition. If the function includes varargout in its definition, then nargout returns the negative of the number of outputs.



```
function res = func(a,b)

switch nargin
    case 1
        res = 2*a;
    case 2
        res = a * b;
    otherwise
        res = 0;
end
end
```

Command Window

```
>> func(1)

ans =

     2

>> func(3,5)

ans =

    15

>> func()

ans =

     0

>> nargin('func')

ans =

     2
```

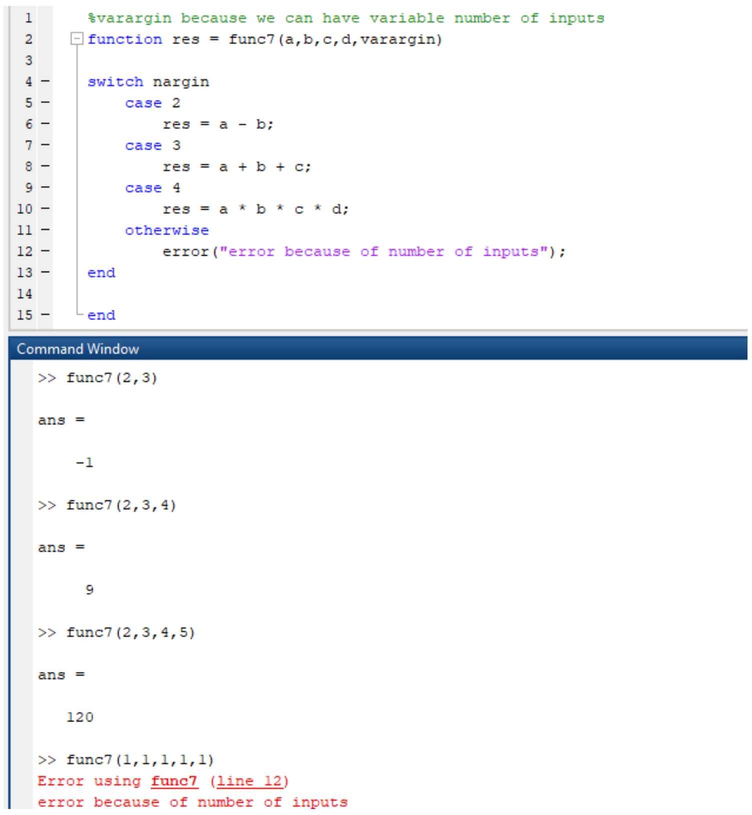


```
function [res1, res2] = func(a)

res1 = 2*a;

if nargout > 1
    res2 = 3*a;
end

end
```

Command Window

```
>> b = func(3)

b =

     6

>> [b,c] = func(3)

b =

     6


c =

     9

>> nargout('func')

ans =

     2
```

2-7)



```
%varargin because we can have variable number of inputs
function res = func7(a,b,c,d,varargin)

switch nargin
    case 2
        res = a - b;
    case 3
        res = a + b + c;
    case 4
        res = a * b * c * d;
    otherwise
        error("error because of number of inputs");
end

end
```

Command Window

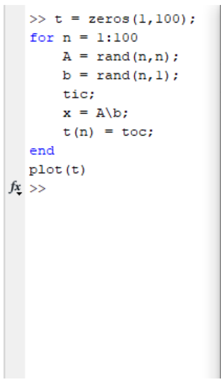
```
>> func7(2,3)

ans =

    -1

>> func7(2,3,4)

ans =

     9

>> func7(2,3,4,5)

ans =

   120

>> func7(1,1,1,1,1)
Error using func7 (line 12)
error because of number of inputs
```

2-8)

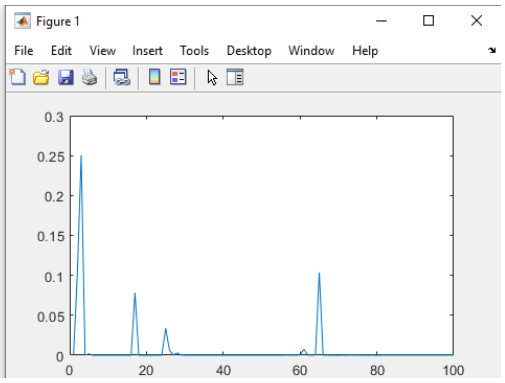
	Syntax	Description
tic	tic	tic starts a stopwatch timer to measure performance. The function records the internal time at execution of the tic command. Display the elapsed time with the toc function.
	timerVal = tic	timerVal = tic returns the value of the internal timer at the execution of the tic command, so that you can record time for simultaneous time spans.

	Syntax	Description
toc	toc	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.
	elapsedTime = toc	elapsedTime = toc returns the elapsed time in a variable.
	toc(timerVal)	toc(timerVal) displays the time elapsed since the tic command corresponding to timerVal.
	elapsedTime = toc(timerVal)	elapsedTime = toc(timerVal) returns the elapsed time since the tic command corresponding to timerVal.

Code to measure how the time required to solve a linear system varies with the order of a matrix:



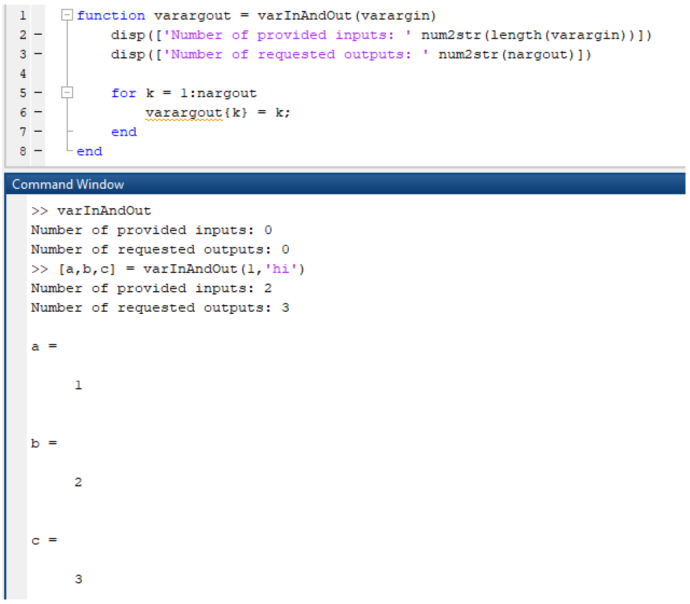
```
>> t = zeros(1,100);
for n = 1:100
    A = rand(n,n);
    b = rand(n,1);
    tic;
    x = A\b;
    t(n) = toc;
end
plot(t)
>>
```



2-9)

	Syntax	Description
varargin	varargin	varargin is an input variable in a function definition statement that enables the function to accept any number of input arguments. Specify varargin using lowercase characters, and include it as the last input argument after any explicitly declared inputs. When the function executes, varargin is a 1-by-N cell array, where N is the number of inputs that the function receives after the explicitly declared inputs. However, if the function receives no inputs after the explicitly declared inputs, then varargin is an empty cell array.

	Syntax	Description
varargout	varargout	varargout is an output variable in a function definition statement that enables the function to return any number of output arguments. Specify varargout using lowercase characters, and include it as the last output argument after any explicitly declared outputs. When the function executes, varargout is a 1-by-N cell array, where N is the number of outputs requested after the explicitly declared outputs.



```
function varargout = varInAndOut(varargin)
    disp(['Number of provided inputs: ' num2str(length(varargin))])
    disp(['Number of requested outputs: ' num2str(nargout)])

    for k = 1:nargout
        varargout{k} = k;
    end
end
```

Command Window

```
>> varInAndOut
Number of provided inputs: 0
Number of requested outputs: 0
>> [a,b,c] = varInAndOut(1,'hi')
Number of provided inputs: 2
Number of requested outputs: 3

a =

     1


b =

     2


c =

     3
```