Topic 7

Introduction to User-Defined Function

Function Types

Creating UDF

Creating Anonymous Function

Function Type

Two Types:

1) **Built-in function**

Functions that come together with the software. Example of built-in functions;

```
abs,sin,atand,fix,sgn,mean
```

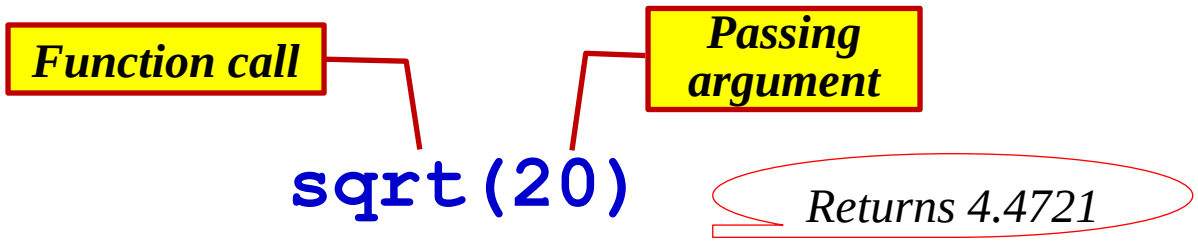
2) **User defined function (UDF)**

New functions that the **programmer defines**, and **then uses** (call) in either the Command window or script.

UDF is saved as .m-file.

Function - Call and Argument

- A function is implicitly called when an expression that contains the name of the function is evaluated.
- The passed argument is processed in the function routine and return none or a single or several values.
- After a function has finished execution, execution resumes at the point just after the call.



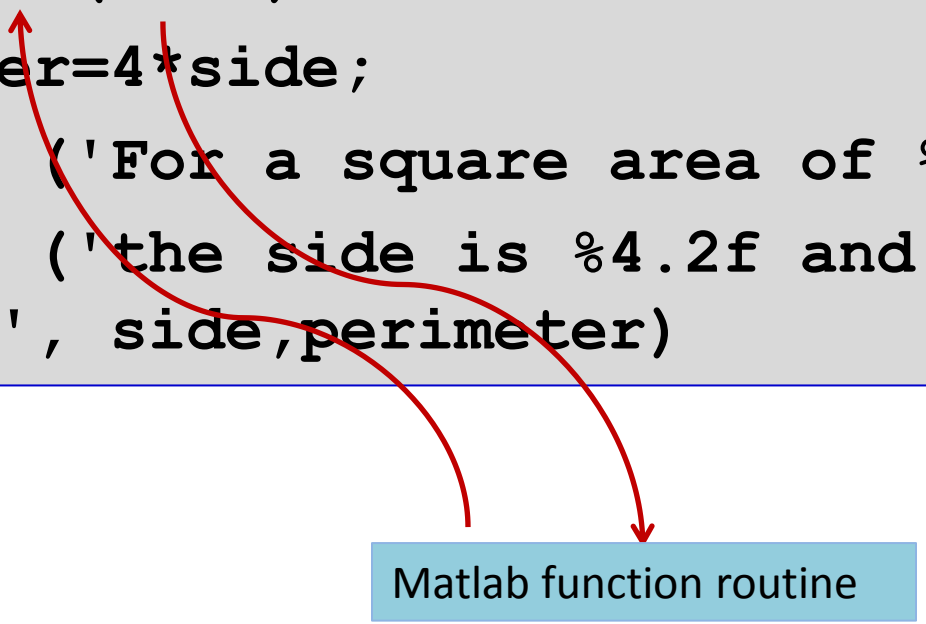
Function - Control transfer

Calling a function in script file

Script file

```
% Get Area of a square find Sides Perimeter


area=input('Area of square :');
side=sqrt(area);
perimeter=4*side;
fprintf ('For a square area of %5.1f\n', area)
fprintf ('the side is %4.2f and perimeter is
%4.2f\n', side,perimeter)
```



User Defined Function

Creation Format & Calling UDF

Vector argument

Matlab Command & Function

User-Defined-Function

- For **built-in** functions, the **programmer only need to call** the function with the require argument, and it will return a value. The function **routine is already** in the Matlab.

- But for UDF the function, it **has to be created**.
  This function can then called in the script or command window.

There are several types of user-defined functions. In this section concentrates on function that returns a single result.

Other types of functions will be discussed later.

Creating UDF - format

```
function outputarg = functionname(input args)
    % comments describing the function
    Statements –
    Statements – must include putting value in the outputarg
end % end of function
```

Creating UDF example

circlearea.m

filename

Functionname

```
function area = circlearea(rad)
% circlearea calculates area of a circle
% to call this functon : circlearea(radius)
area = pi * rad ^ 2
end
```

To display m-file function in Command window, use `type` *command*

Calling UDF

(cont'd)

```
>> area=circlearea(3)
area =
    28.2743
```

Calling function in the Command window

argument

Function call

```
>> luas=circlearea(4.2)
luas =
    55.4177
```

```
>> luas=circlearea(3);
```

```
>> disp(circlearea(2))
    12.5664
>> fprintf('Area is %4.2f\n',circlearea(2.4))
Area  is 18.10
```

UDF - passing vector argument

To allow vectors to be passed to functions array operations (**.^ & .***) must be used.

Circlearea.m

```
function area = circlearea(rad)
% circlearea calculates area of a circle.
% To call this functon : circlearea(radius)
area = pi * rad .^2
end
```

Modified function to allow vector argument

```
>> rad=[2 4 3];
>> luas=circlearea(rad)
luas =
    12.5664    50.2655    28.2743
```

vector

UDF - block comment

```
>> help circlearea
```

Display block comment

```
 circlearea calculates area of a circle.
  To call this function : circlearea(radius)
```

It is a good practice to include block comment in a function, that may include:

- Name of function
- Description of what the function does
- Description of input
- Description of output
- Programmer name and date
- Information on revisions
- etc.

UDF - more than one arguments

Cone_example.m

```
%Calculates vol of cone
%input : radius and height
radius=input('Enter radius :');
height=input('Enter height :');
volume=conevol(radius,height);
fprintf('For a cone of radius %4.1f ',radius)
fprintf('and height %4.1f\n',height)
fprintf('the volume is %4.1f\n',volume)
```

Script file

Arguments must correspond one-to-one

conevol.m

```
function vol= conevol( rad,ht )
%calculates volume of cone
%  format : conevol(rad, ht)
vol=(pi/3)*rad.^2.*ht;
end
```

Function file

```
>> Cone_example
Enter radius :5
Enter height :7
For a cone of radius  5.0 and height  7.0
the volume is 183.3
```

Local variable in Function

Variables used in the **script** are also known in the **Command window** and vise versa. They use a common workspace – the **base workspace**.

Variables use in a function are local to that function only because it has **its own** workspace.

```
function outcost = cylcost(radius, height, cost)
% cylcost calculates the cost of constructing a closed
% cylinder
% Format of call: cylcost(radius, height, cost)
% Returns the total cost
% The radius and height are in inches
% The cost is per square foot
% Calculate surface area in square inches
surf_area = 2 * pi * radius .* height þ 2 * pi * radius .^ 2;
% Convert surface area in square feet and round up
surf_areasf = ceil(surf_area/144);
% Calculate cost
outcost = surf_areasf .* cost;
end
```

surf_area and *surf_areasf* are local variables.

They are only valid/visible in the function

Scope of variable

Command and Function

commands - format, type, save, load

They are actually just the shortcuts for function calls.
Function can be used as commands when:

- All arguments are strings
- Does not return any values

```
>> type script1

radius=3
circumf=2*pi*radius
```

```
>> type('script1')

radius=3
circumf=2*pi*radius
```

both produce the same result

Command and Function- cont'd

Load as a command

```
>> load Pt1.dat
>> type Pt1.dat

     3        7
```

When load is used as a command creates a variable with the same name as the file.

Load as a function

```
>>Locat1 = load('Pt1.dat')

Locat1 =
     3     7
```

When load is used as a function the content of the file can be assigned to a different variable name..

Anonymous Function

Creation format & calling

Anonymous function with &without arguments

Saving anonymous function in mat-file

Anonymous Function

- Simple **one line** function
- Does not have to stored in m-file.
- Can be created in Command window or script or UDF

Syntax

```
fnhandlevar = @ (args) functionbody
```

A way of referring to the function

Arguments correspond to the arguments that are passed to the function

Valid MATLAB expression

Anonymous Function - with arguments

Example

```
>> circlearea=@(rad) pi*rad.^2;
>> circlearea(3)
ans =
    28.2743
>> area=circlearea(2:4)
area =
    12.5664   28.2743   50.2655
```

Anonymous function

Scalar argument

Vector argument

Anonymous Functions - without arguments

Example –
print a random number

No argument

```
>> prtran = @ () fprintf('%.2f\n',rand);
>> prtran()
0.76
```

No argument in call function

Typing the function handle will display the function definition.

```
>> prtran
prtran =
    @ () fprintf('%.2f\n',rand)
```

No parentheses

Anonymous Functions - save in MAT-file

- Frequently used anonymous functions can be saved in a MAT-file and then loaded from this MAT-file in every MATLAB Command window.
- Other anonymous function could be appended to the MAT-file.

Anonymous Functions - save in MAT-file

```
>> cirarea = @ (radius) pi * radius .^ 2;
>> save anonfns cirarea
>> clear
>> load anonfns
>> who

Your variables are:

Cirarea

>> cirarea
cirarea =
    @ (radius) pi * radius .^2
```

Saving **cirarea** function in **anonfns** mat-file

Only one function in anonfns mat-file

Anonymous Functions - appending to MAT-file

```
>> circircumf =@ (rad) 2 *pi .*rad;
>> save anonfns circircumf -append
>> clear
>> load anonfns
>> who

Your variables are:

cirarea     circircumf
```

Appending anonymous function (**circircumf**) to existing mat-file (**anonfns**)

Now there two functions in anonfns mat-file

Problem Examples

Problem Example 1

A simply supported beam length *l*, carries a udl along the whole span. Determine the bending moment *at 0.2l*, 0.35*l* and *0.8l* from the left support. Use anonymous functrion to get bending moments.

$r_a = ql/2; \quad m_x = r_a x - qx^2/2$

Bmfullyudlanonyfunc.m

```
clear
q=input('Enter udl load intensity: ');
l=input('Enter beam span: ');
x1=input('Enter first point: ');
x2=input('Enter second point: ');
x3=input('Enter third point: ');
ra=q*l/2;
bm=@(x) ra*x-(q*x^2)/2;
fprintf('\nBM at %.1f is %.1f\n',x1,bm(x1))
fprintf('BM at %.1f is %.1f\n',x2,bm(x2))
fprintf('BM at %.1f is %.1f\n',x3,bm(x3))
```

Anonymous function

You can also define x's as vector. As in the earlier example

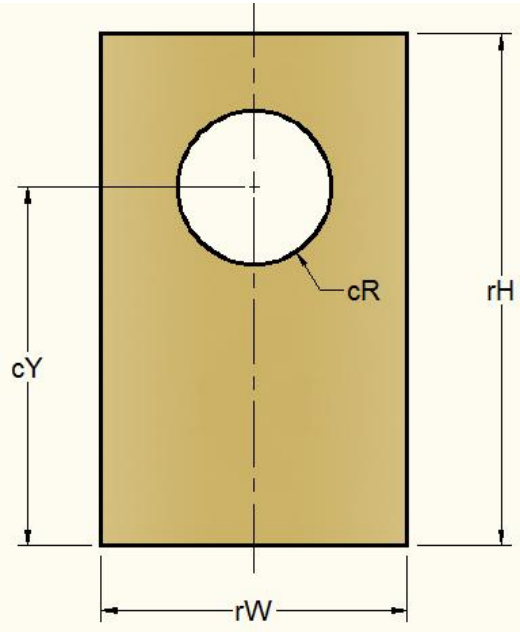
Problem Example 1 - cont'd

```
>> bmfullyudlanonyfunc
Enter udl load intensity: 15
Enter beam span: 7
Enter first point: 0.2*l
Enter second point: 0.35*l
Enter third point: 0.8*l

BM at 1.4 is 58.8
BM at 2.4 is 83.6
BM at 5.6 is 58.80
```

Problem Example 2

Determine the location of the centroid of the section and its moment of inertia about the horizontal centroidal axis.



Create 2 anonymous functions
and save them in a in mat-file

```
>> ACircle=@(r) pi*r.^2;
>> save AreaAnonFunc ACircle
>> ARect=@(b,h)  b*h;>>
>> save AreaAnonFunc Arect -append
>> load('AreaAnonFunc.mat')
```

CentroidRecHoleAnonInMatFile.m

```
rW=input('Rectangle Width : ');
rH=input('Rectangle Height : ');
cR=input('Radius of circle : ');
cY=input('Circle centroid from base : ');
rY=rH/2;
```

Problem Example 2 - cont'd

```
% calculate areas
cA=ACircle(cR);
rA=ARect(rW,rH);
% calc axes transfer distance
cD=yBar-cY;
rD=rY-yBar;
% calc moment of inertia
rI=rW*rH^3/12;
cI=pi*cR^4/4;
secI=(rI+rA*rD^2)- (cI+cA*cD^2);
fprintf('Ybar is at %f and Ix is
%.1f\n',yBar,secI)
```

Anonymous functions

```
>> CentroidRecHoleAnonInMatFile
Rectangle Width : 5
Rectangle Height : 10
Radius of circle : 2
Circle centroid from base : 4
Ybar is at 5.335697 and Ix is 387.3
```

Thank You