Lecture 7 - Functions (subprograms)

Functions (subprograms) - Motivation

- In the previous sort examples, we wrote algorithms that sorted 1-D arrays, and another that sorted 2-D arrays.
- What if I had to sort several sets of data in a single program? Each time I want data sorted, I need to copy and paste the same algorithm over again. Not very efficient.

e.g. I collect three pieces of information from 6 students, each is saved into a different vector

1) exam grade  \( \text{grd} = [68 \ 95 \ 85 \ 91 \ 78 \ 81] \)
2) height in inches \( \text{ht} = [72 \ 65 \ 70 \ 76 \ 61 \ 68] \)
3) shoe size \( \text{sz} = [11 \ 6 \ 9 \ 13 \ 4 \ 10] \)

For some reason I wish to sort each of these vectors from low to high

From what we know so far, what are our options?

1) write three successive sort structures, each one changing the variable of interest to either \( \text{grd}, \text{ht}, \text{or sz} \)

2) create a 2-D array out of the 3 1-D arrays

\[
\text{studentinfo} = [\text{grd}' \ \text{ht}' \ \text{sz}'];
\]

the ‘ after each vector changes horizontal (row) to vertical (column)
The result is:

\[
\text{studentinfo} = \begin{bmatrix}
68 & 72 & 11 \\
95 & 65 & 6 \\
85 & 70 & 9 \\
91 & 76 & 13 \\
78 & 61 & 4 \\
81 & 68 & 10
\end{bmatrix}
\]

Now use the 2-D sorter to sort each column

Suppose I only wanted to sort the second column in the 2-D array?
I’d have to write another special algorithm to handle that...

Is there an easier way to handle sorting? Can’t I just write a group of statements that can sort whatever it is given as input? Let this group of statements reside in a separate small program. Whenever we want to sort any information, just send it off to this program, let it do the work, and send the results (sorted data) back to me.
Functions (subprograms) - Motivation (cont.)

3) How about some way to have the sorting algorithm self-contained somewhere else, and your main program just accesses it when needed.

- Generically this is often called a subprogram or subroutine
- In MATLAB it's called a function

Two important properties of a good function:

a) The function should be able to handle any form of the data:
   - a scalar, a vector, a column in a 2-D array, a row in a 2-D array, etc.

b) The function should not be dependent on the specific name of the array being used in the main program. Incoming information will be renamed to something non-specific

example: Instead of re-writing the sorting algorithm three times, wouldn’t it be great if we could just do this? Let’s say that grd, ht, and sz are vectors with unsorted data in them. From the main program, we can just send these vectors to a function called ‘sorter’, and save the sorted results in new vector names.

```matlab
>> grd_sort = sorter(grd);
>> ht_sort = sorter(ht);
>> sz_sort = sorter(sz);
```

That’s it!, no loops, all the MATLAB commands needed to sort are in the function called ‘sorter’

the command ‘sorter’ will do all the work, and return the results

Even if the data is a 2-D array, we can use the 1-D sorter by just sending in one column of data at a time:

```matlab
>> grd_sorted = sorter(studentinfo(:,1))   %%sort column 1
>> ht_sorted = sorter(studentinfo(:,2))   %%sort column 2
>> sz_sorted = sorter(studentinfo(:,3))   %%sort column 3
```

or we can get fancier

```matlab
>> for i=1:3
   >> sortedinfo(:,i) = sorter(studentinfo(:,i));
>> end
```

Many functions are already built into MATLAB

```matlab
>> numgrades = length(grd)
```

We can create a library of our own functions that can be accessed just like any of the built in functions
Functions - schematic
The figure below demonstrates that a main program can simply be used to input data, then delegate tasks to a group of smaller functions. The main program is much more organized and cleaner this way.

Setting up Functions - the rules:
- A Function is created within a text editor like any other .m file
- A Function is a separate program accessed from any MATLAB code or line `>>` command
- A function is different from a main program in its first line. It must start with the word `function`
- list output variable(s) `[out1, out2, out3,...]` within brackets `[ ]`
- `funcname` must be same as the function program name
- list input variable(s) `(in1, in2, in3,...)` within parenthesis `()`
- assign results to output variable names
- when function ends, main program continues where it left off
- changes to input variables within a function do not change their value in the main program
- DO NOT use the `clear` command in a function, or it will wipe out what’s goin’ on in the main program.

general format:

```
funcname.m

function [out1, out2]=funcname(in1, in2,...)
%
%these comments will show up when you
%type help funcname in MATLAB, so explain
%your function here

use in1, in2, in3... to perform calculations
assign results to out1, out2, ...
```

Main Program
```
>> load grades
>> big = max(grade)
>> inorder = sorter(grade)
>> numgrade = length(grade)
>> average = mean(grade)
>> deviation = std(grade)
```
Using a function

Accessing a function works by using the same structure as the first line of the function itself from a main program or command line:

```matlab
>> [output variables list] = function_name ( input variables list )
```

This can be done from within a program or from a command line `>>`

When MATLAB executes the above statement it does the following:

1) Locates the function_name if it is in the MATLAB path list
2) Sends the input list of variables to the function
3) Executes the statements inside the function
4) Passes results through the output list of variables
5) Continues executing the main program

**Simple example: write a function to find a cube root**

\[
out = \frac{3}{n}
\]

```matlab
function [out] = cubert(in)
%function out = cubert(in)
%This function takes the input ‘in’,
%and calculates its cube root
%in can be any size matrix, and cubert will
%operate on each element individually
%
out = in.^(1/3);
```

A main program that uses cubert

```matlab
clear
%
% a program that uses the cubert function
%
z=peaks(50);
x=cubert(abs(z)+1);   % note the variable names used here are not
% same as in the function
figure(1);
surf(abs(z));
a=axis;
figure(2)
surf(x);
axis(a);
```

lecture7_1.m
Functions - local memory, local workspace

A function is set up and executed in a separate workspace from the main MATLAB workspace where your program is running. This affects how and where data is stored, and which programs can see the data.

What does this mean to you?

- Variables that exist in one workspace are not seen or recognized by the other workspace.
- The only information shared between workspaces is through the input and output lists.

1) Contents of input variables are copied into the function under the local function name. Changes to the variable within the function do not change the variable in the main program.
2) Contents of the function’s local output variables are copied into the main program to the variable(s) left of the =
   - Think of the input/output variable lists as one way mirrors between main workspace and function workspace.
   - Any other variables within the function that are not in the input or output lists are never shared between programs.

Let’s revisit the cube root function example:

```
lecture7_1.m
clear
% comments
% z=peaks(50)
x=cubert(abs(z)+1);
figure(1);
surf(abs(data));
a=axis;
figure(2);
surf(x);
axis(a);
```

```
cubert.m
function out=cubert(in)
% descriptive comments
%belong here
% out = in.^((1/3));
a=length(in);
disp('number of')
disp('elements is = ')
disp(a);
in = 0; <==what will this do??
```

<table>
<thead>
<tr>
<th>main prog. variables</th>
<th>function variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(z)+1</td>
<td>in</td>
</tr>
<tr>
<td>x</td>
<td>out</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

- ‘abs(z)+1’ is copied into the function as ‘in’
- ‘out’ is copied from the function to the main program as ‘x’
- The line ‘in=0’ does not change anything in main program
- Both the main program and the function use the variable ‘a’. These two ‘a’ variables are not the same, they are completely separated in different work spaces. Changing one will not change the other.
Functions: An example with multiple inputs and outputs

Goal: write a function called ‘minmax’ that will find either the minimum or the maximum of a given vector, and return the value, and its location in the vector.

Pseudo-code:
1) Send a vector to the minmax function
2) Send an indicator of which one to find (min or max)
3) Assume first value in the input vector is the min/max (pointer = 1)
4) Compare with the next value, if next is new min/max, re-assign the pointer to that place
5) Continue the process in a loop through the entire vector
6) Use the pointer to identify the actual min/max value
7) Return value and pointer to main program

The next page has the function, a program that uses this function, and diagrams the workspace useage
function [val, ptr] = minmax(in, indicator)

%function [val, ptr] = minmax(in, indicator)
%find either minimum or maximum of ‘in’
%if indicator < 0, find min
%if indicator >= 0, find max
lg = length(in);
ptr = 1
for i = 2:lg
    if (indicator < 0)
        if (in(i) < in(ptr))
            ptr = i;
        end
    else
        if (in(i) > in(ptr))
            ptr = i;
        end
    end
end
val = in(ptr);

function minmax.m

main program workspace minmax workspace

clear
%program to find minimum / maximum in a list
% 1/31/00    K. Gurley
%
load grades.inp -ascii
ind = -1;
[smallest, location] = minmax(grades, ind);
disp(['Low grade = ', num2str(smallest)]);
disp(['student # ', num2str(location)]);

[highest, location] = minmax(grades, 1);
disp(['High grade = ', num2str(highest)]);
disp(['student # ', num2str(location)]);

main program
another function example

Recall the homework assignment where the deflection of a beam was calculated. We’ll revisit that example and use a function to perform most of the calculations.

Main program
The main program below will collect the necessary input, then send it to the function ‘beam’ to do the calculations. The variables in the input / output must appear in the same order when calling the function as they appear in the first line of the function. The main program will then display results.

```
%Main program will prompt user for material properties of a beam and
%applied load. A function called 'beam' will be called to calculate deflection
%and classify deflection as safe, danger, or failure
clear
% INPUT SECTION
Length=input('Enter beam length =>  ');
Emod=29000;
Inertia=400;
Load=input('Enter tip point load =>  ');
nump=input('Enter number of places to calc. defl. =>  ');
% CALL BEAM FUNCTION TO PERFORM CALCULATIONS
[defl,x,classification]=beam(Length,Emod,Inertia,Load,nump);
% PLOT RESULTING DEFLECTION
figure(1);clf;
plot(x,defl,[0 max(x)],[0 0])
% ADJUST AXIS
xmin= -0.2*Length;
xmax= 1.2*Length;
if (max(defl) <= 0)
ymin= 1.2*min(defl);
ymax= 0.8*abs(min(defl));
axis([xmin xmax ymin ymax]);
else
ymin= -0.8*max(defl);
ymax= 1.2*max(defl);
axis([xmin xmax ymin ymax])
end

% PUT CLASSIFICATION ON GRAPH
disp('put cursor on the graph')
gtext(['classification = ' classification])
```
**Function ‘beam.m’** - The function called in the previous program

- Note the detailed use of comments to spell out how the function works
- Each of the variables in the output list are assigned to values at some point inside the function
- Names in the input / output list don’t have to be same as those in the main program

```matlab
function [out,x,class]=beam(L,E,I,P,np)
%function [out,x,class]=beam(L,E,I,P,np)
%
%function calculates the deflection of cantilevered
%beam with enf point load
%at several points and classifies as safe,
danger, or fail
%
%INPUT:
%L: length of beam
%E: young's modulus
%I: moment of inertia
%P: tip point load
%np: number of evenly spaced points along beam to calc.
%
%OUTPUT:
%out: vector of deflection values
%x: locations along beam at which 'out' is calculated
%class: classification of beam deflection
%
%last modified 9/13/00
x=   0 : (L/(np-1)) : L ;
out= P/(6*E*I)*(x.^3 - 3*L*x.^2);
if (max(abs(out)) < L/180)
class = 'safe';
elseif(max(abs(out)) < L/60)
class = 'danger';
else
class = 'failure';
end
```
Diagram of the main and function workspaces

**main program workspace**  **minmax workspace**

<table>
<thead>
<tr>
<th>defl</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>classification</td>
<td>class</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emod</td>
<td>E</td>
</tr>
<tr>
<td>Inertia</td>
<td>I</td>
</tr>
<tr>
<td>Load</td>
<td>P</td>
</tr>
<tr>
<td>nump</td>
<td>np</td>
</tr>
</tbody>
</table>

ymin  
ymax  
xmin  
xmax

Remember: Any variables above without an arrow are not seen in the other workspace.