Creating and Executing Script Files

- We can save a collection of MatLab commands in a file, called a **script file** (the term “script” signifies that MatLab simply reads from the script found in the file) or **M-files** (the term “M-file” means the script filenames must end with the extension “.m”).
- We can write a script file, `L03circle.m`, with contents:

```
% CIRCLE - a script file to draw a unit circle
% -----------------------------------------
% generate 100 linearly spaced elements
% from 0 to 2pi and save in an array theta
theta=linspace(0,2*pi,100);
```

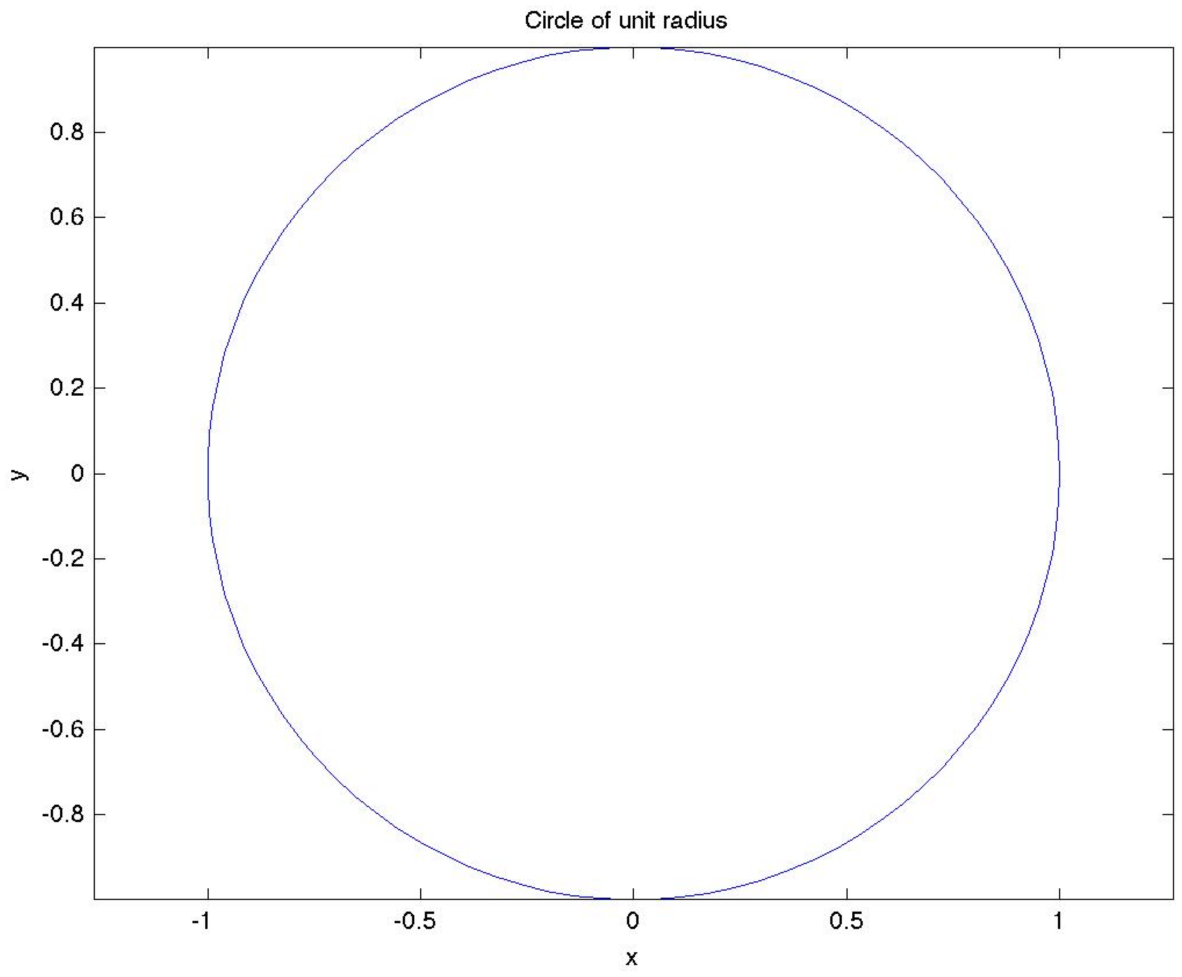
```
% x is a 100 element vector of cos(theta) terms
x=cos(theta);
% y is a 100 element vector of sin(theta) terms
y=sin(theta);
% plot x versus y elements (a circle)
plot(x,y);
% set the length scales of the 2 axes the same
axis('equal');
% label the x axis with x
xlabel('x');
% label the y axis with y
ylabel('y');
```

```
% put title on graph
title('Circle of unit radius');
print('-djpeg',circle.jpg);
```

- In MatLab, type `help L03circle` and the comment at the beginning of the `L03circle.m` file will print:

  ```
  % CIRCLE - a script file to draw a unit circle
  % -----------------------------------------------
  ```

- At the MatLab prompt `>>` type `L03circle`. Now an image of this graph appears in the figure window.


Circle of unit radius
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
y
-1
-0.5
0
0.5
1
x

Creating and Executing a Function File

- We can modify `L03circle.m` to be a function `L03circle_fct.m` as:

```
function [x,y] = L03circle_fct(r,filename);
% CIRCLE Draw a circle of radius r
% Syntax: [x,y]=L03circle_fct(r,filename);
% or L03circle_fct(r,filename);
% Input: r, the radius and
% filename, the file to save the jpg image to
% Output [x,y] - the x and y coordinates
% of the circle points
```

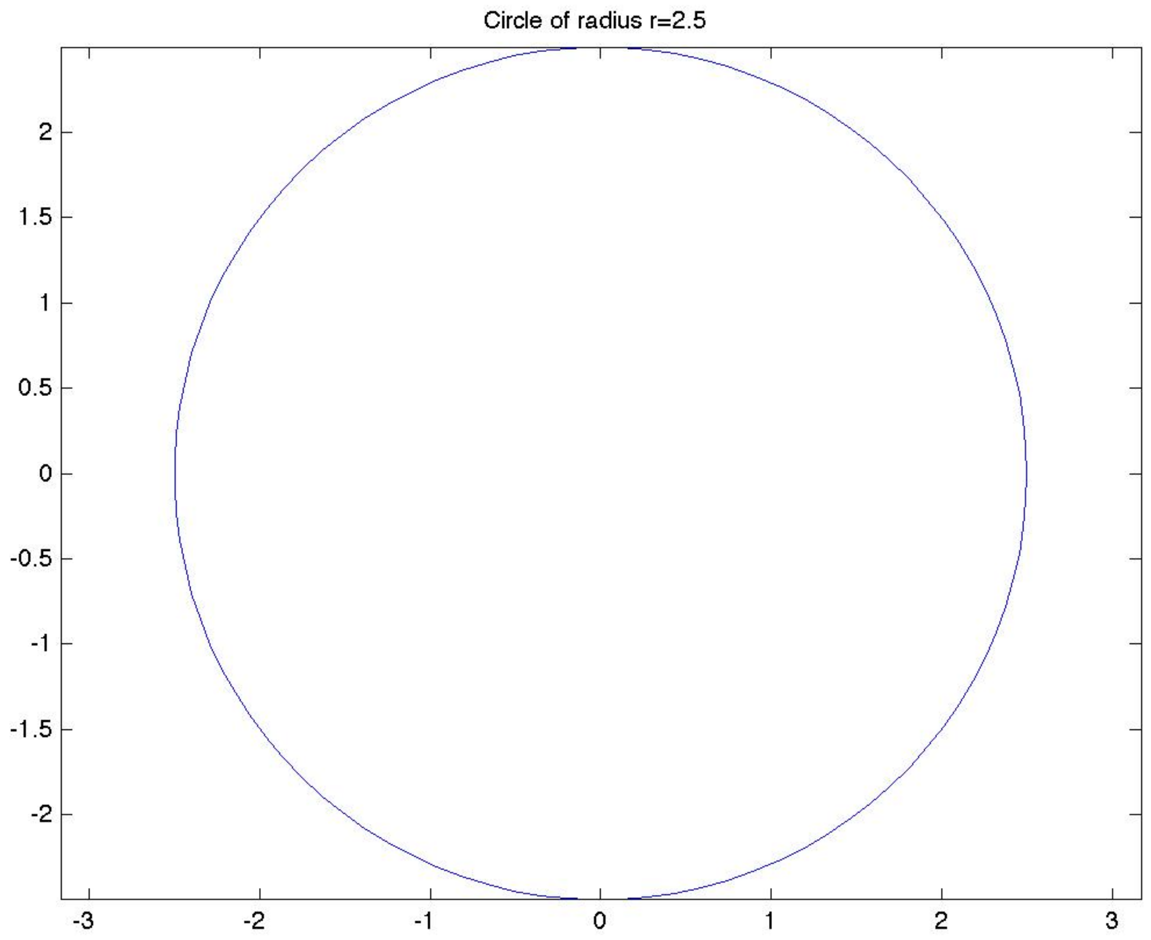
```
%
theta=linspace(0,2*pi,100);  % create vector theta
x=r*cos(theta);              % generate x coordinates
y=r*sin(theta);              % generate y coordinates
plot(x,y);                   % plot the circle
axis('equal');               % set equal scale on axes
% put title on graph
title(['Circle of radius r=',num2str(r)])
print('-djpeg',filename);
end % L03circle_fct
```

- We can execute `L03circle_fct` in a number of ways:

```
R=5;
```

```
filename='my_circle_fct.jpg';
% execute circlefn with input R and
% explicit output list
[x,y]=L03circle_fct(R,filename);
% input radius specified directly
[cx,cy]=L03circle_fct(2.5,filename);
% you can ignore the output
L03circle_fct(1,filename);
% the argument can be an expression
L03circle_fct(R^2/(R+5*sin(R),filename);
```

- The graph for `L03circle_fct(2.5,'my_circle_fct.jpg');` is as shown. Note the title and the x and y coordinates are different.



- Note that `num2str(r)` converts a number in `r` to a character string. The square brackets, `[` and `]`, concatenate the two strings between them.

Recursive Functions in MatLab

- A recursive function is a function defined in terms of itself. The recursive formula has to have a condition that stops the recursion.
- Why recursion: sometimes it is natural to write a function recursively: for example, if you are given the recursive definition of a function it is natural (and easy) to code it that way.
- Consider the recursive function, **L03factorial(n)** [$n!$]. Note the recursive calculation $n! = n\,(n-1)!$ and the stopping criteria $1! = 0! = 0$.

```
function factn=L03factorial(n);
% FACTORIAL: function to compute factorial
% Call Syntax: factn=L03factorial(n)
if(n < 0)  % If n is negative, factorial is undefined
error('Error: n is negative - factorial undefined');
else
if(n==0 || n==1)
factn=1; % 0! is defined to be 1
else % n > 1
fprintf('%d ',n);
factn=n*L03factorial(n-1);
end % if
end % L03factorial
```

- `fprintf` prints the value of `n` to the MatLab command window using format `%d` for integers. Another format is `%f` which is the format for single or double percision floating point numbers. Format `%s` is for strings and format `%c` is for characters. You can control the appearance of the output values. For example, `%4d` will use 4 characters to print an integer while `%8.3f` will print a floating point number with an overall width of 8 characters and 3 characters to the right of the decimal point (which occupies one space) and 4 characters to the left of the decimal point. For example, `fprintf('%8.3f\n',pi)` will print π as ___`3.142`. Note the 3 blank characters (we use underscore to represent them) before the 3. Also note that rounding occurs for the last digit of the number (because 3.1415 changes to 3.142 as the last digit is rounded).

- Note that when n is negative factorial is undefined and an error message is printed. Also note that when n is 0 or 1 no recursive call to factorial is made.

- Some runs:

```
L03factorial(-2)

Error using L03factorial (line 5)

n is negative - factorial undefined

L03factorial(0)

ans =

     1

L03factorial(6)

ans =

     720
```

- You need to be aware that MatLab restricts the recursion depth to 500. If you attempt to than more than 500 recursive calls (without returning, so all there are 500 instances on the stack), you will get an error message like:

```
??? Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N)
to change the limit. Be aware that exceeding your available stack space
can crash MATLAB and/or your computer.
```

- MatLab (Mathworks) says:

```
"The recursion limit is there specifically to avoid these types of crashes.
Recursive functions have high memory requirements. Each recursion doubles
the amount of memory that is allocated by the function and a stack is used
to store the data. A stack consists of a limited amount of memory and when
this limit is reached, the program cannot keep running.

This is not a MATLAB limitation, but a general limitation of any programming
language. In general, if you see a message about reaching the recursion
limit, this should be a sign that there might be a syntax or design error
in your program.
```

- MatLab sets up separate workspaces each time a function is called, building up a stack of them, and this quite memory intensive.
- Such a message is usually an indication that you have a recusion error: infinite recursion is the main culprit. Also ask yourself: is a reursive solution really the best. Most recusive functions can be made iterative.
- Lastly, note that note that there is a fundamental difference between a function that gets called 500 times (ok) and a function that recurses 500 times (500 stack entries). Normally, the latter is quite rare if the recursion is correct!!!
- You could use `set(0,'RecursionLimit',1000)` to change this recursion limit but probably to no good effect. If you have infinite recur-

sion you will still get this error at 1000 as well. Be aware that exceeding your available stack space can crash MATLAB and/or your computer!!! Excessive recursion depth almost surely indicates a problem in the code/algorithm and not some limitation in MATLAB.

Compiling Your MatLab Code

- Suppose you have a file named `L03circle_fct.m`, which is interpreted when executed). You may want to save a compiled version of this file, made using:

  ```
  pcode L03circle_fct
  ```

  which generates compiled code for this function in a file named

`L03circle_fct.p`, The next time `L03circle_fct` is called `L03circle_fct.p` is used.

- Unless your function is large, it probably make no difference (speed-wise) whether you use a `m` or a `p` file. The best use of `p` files is to protect your propriety rights to your code (other people can't see it but can use it), provided they are running your code on the same architecture it was compiled on.

Global Variables

- Syntax: `global` X defines X as global in scope.
- Typically, each MatLab function has its own local variables, which are different from those of other functions (even if they have the same names) and from the variables in your workspace.
- However, if several functions, and possibly the base workspace, all declare a particular name as global, they all share a single copy of that variable. Any assignment to that variable, in any function, is available to all the functions declaring it global.
- If a global variable does not exist, then the first time you use it (without initialization) that variable is initialized to the empty matrix.

- If a variable with the same name as the global variable already exists in the current workspace, MatLab prints a warning and changes the value of that variable to match the global variable's value.

- All parameters of a MatLab function are "pass by value", including arrays. That means large arrays have to be copied by the function and is any changes are made the array must be passed back to the calling routine, for example, `A=fct(A)` where `A` is an array. `global` variables allow this to be avoided (saves space and time) by having `global A` in function `fct`.