% 7.1 EXAMPLE: NEWTON’S METHOD AGAIN

% Use the Editor to create and save (in the current MATLAB directory) the function

% file f.m as follows:

function y = f(x)

y = x^3 + x - 3;

%----------------------------------

% Then create and save another function file df.m:

function y = df(x)

y = 3\*x^2 + 1;

%----------------------------------

% Now write a separate script file, newtgen.m (in the same directory), which will

% stop either when the relative error in x is less than 10?8, or after 20 steps, say:

% Newton’s method in general

% excludes zero roots!

steps = 0; % iteration counter

x = input( 'Initial guess: '); % estimate of root

re = 1e-8; % required relative error

myrel = 1;

while myrel > re & (steps < 20)

xold = x;

x = x - f(x)/df(x);

steps = steps + 1;

disp( [x f(x)] )

myrel = abs((x-xold)/x);

end

if myrel <= re

disp( 'Zero found at' )

disp( x )

else

disp( 'Zero NOT found')

end

%================================================================================================

% 7.2 BASIC RULES : p. 165

% Write a function file stats.m:

function [avg, stdev] = stats( x )

% function definition line

% STATS Mean and standard deviation % H1 line

% Returns mean (avg) and standard % Help text

% deviation (stdev) of the data in the

% vector x, using Matlab functions

avg = mean(x); % function body

stdev = std(x);

%----------------------------------

% Now test it in the Command Window with some random numbers, e.g.,

r = rand(100,1);

[a, s] = stats(r);

%----------------------------------

1. Start with the function keyword.
2. Input & output arguments.
3. Multiple output arguments.
4. Function naming. (same as variable)
5. Help text (start with ‘%’ )

Using help function\_name display Help text.

1. Local variable : exist inside the function

Here, avg, & stdev are the local variable, they are available only inside the stats subfunction.

1. Global variable.

Two rules of using global variables:

* Before define every workspace, to declare global variable.
* Using ‘capital letters’ for Global variable. Make different from the local variable.

‘whos global’ : to check the global variable in the workspace.

‘clear global X’ : clear global variable X to all workspace.

1. Function that do not return values

function stars(n)

asteriks = char(abs('\*')\*ones(1,n));

disp( asteriks );

1. Vector arguments

% Vector arguments

function d = dice( n )

d = floor( 6 \* rand(1, n) + 1 ); % d is a vector

1. Input arguments is “Call by Value” to pass the values to the sunfunction.
2. Checking the number of function arguments

Add a common line ‘disp(nargin)’, you can find the number of the input.

1. Call a function with any number of input & output arguments

% % Sec 7.2.3 : p-code

* + In general, M – function is the script file. Source code can be seen.
  + Let someone can run your function, however the source code cannot be seen. Translate it to p-code (Pseudo-Code).
  + For example

pcode stats

can generate a ‘stats.p’ p-code.

* + P-code can save computation time.

%% Sec. 7.3 FUNCTION HANDLES

% FUNHANDLE = @FUNCTION\_NAME returns a handle to the named function,

% FUNCTION\_NAME. A function handle is a MATLAB value that provides a

% means of calling a function indirectly. You can pass function

% handles in calls to other functions (which are often called function

% functions).

% Try the following on the command line:

fhandle = @sqrt;

feval(fhandle, 9)

feval(fhandle, 25)

% Two build-in function to pass the funtion name as the argument

% feval & fminbnd

% feval(F,x1,...,xn) evaluates the function specified by a function

% handle or function name, F, at the given arguments, x1,...,xn.

fhandle = @sqrt;

feval(fhandle, 9)

feval(fhandle, 25)

f = @(x,y,c) (x-c).^2+y.^2; % The parameterized function.

c = 1.5; % The parameter.

% IN BUILD-IN FUNCTION feval call the function that you define

b = feval(@(x,y) f(x,y,c),0.6,0.5)

% fminbnd Single-variable bounded nonlinear function minimization.

% X = fminbnd(FUN,x1,x2) attempts to find a local minimizer X of the function

% FUN in the interval x1 < X < x2. FUN is a function handle. FUN accepts

% scalar input X and returns a scalar function value F evaluated at X.

% FUN can be an anonymous function:

X = fminbnd(@(x) sin(x)+3,2,8)

y=2:0.1:8;

plot(y,sin(y)+3)

%----------------------------------

% FUN can be a parameterized function. Use an anonymous function to

% capture the problem-dependent parameters:

clear all;clc;

f = @(x,c) (x-c).^2; % The parameterized function.

c = 1.5; % The parameter.

X = fminbnd(@(x) f(x,c),0,2)

% use help to find the fplot

fplot(@(x) f(x,c),[0,2])

% fplot Plot 2-D function

% fplot(FUN) plots the function FUN between the limits of the current

% axes, with a default of [-5 5].

%

% fplot(FUN,LIMS) plots the function FUN between the x-axis limits

% specified by LIMS = [XMIN XMAX].

% Example: fplot(@(x) x.^2.\*sin(1./x),[-1,1])

% As an example, we would like to

% rewrite our newtgen script as a function newtfun, to be called as follows:

function y = f(x)

y = x^3 + x - 3;

%----------------------------------

% Then create and save another function file df.m:

function y = df(x)

y = 3\*x^2 + 1;

% [x f conv] = newtfun( fh, dfh, x0 )

% The complete M-file newtfun.m is as follows:

function [x, f1, conv] = newtfun(fh, dfh, x0)

% NEWTON Uses Newton’s method to solve f(x) = 0.

% fh is handle to f(x), dfh is handle to f’(x).

% Initial guess is x0.

% Returns final value of x, f(x), and

% conv (1 = convergence, 0 = divergence)

steps = 0; % iteration counter

x = x0;

re = 1e-8; % required relative error

myrel = 1;

while myrel > re & (steps < 20)

xold = x;

x = x - feval(fh, x)/feval(dfh, x);

steps = steps + 1;

disp( [x feval(fh, x)] )

myrel = abs((x-xold)/x);

end;

if myrel <= re

conv = 1;

else

conv = 0;

end;

f1 = feval(fh, x);

%----After previous 3 function defined we can use the following command to run newtfun

% ---------------------

% Method 1: define two function handle

clear all;clc;

fhand = @f;

dfhand = @df;

[x,f1,con] = newtfun(fhand, dfhand, 2)

% Method 2

clear all;clc;

fhand = @(x) x^3 + x - 3; % The parameterized function.

dfhand= @(x) 3\*x^2 + 1;

[x,f1,con] = newtfun(fhand, dfhand, 2)

%% Sec 7.5: function name resolution

% if you having the same name in a variable and a function then the

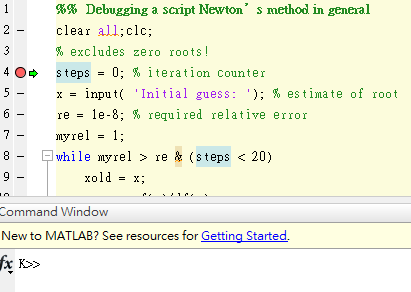
% priority of the MATLAB (check p. 174):

% variable ---> subfunction ---> private function ---> dictionary

%% Sec. 7.6 Debugging a script: please open the subfunction 'newtgen1.m' and run it in a debugging mode

% Newton’s method in general

1. In command Window ‘>>k’ means that you are in the Debug mode.
2. Green arrow means the run ending at this point.



%% Check the procedure in p. 175 for the debugging

% Debugging a script Newton¡¦s method in general

% (1) set breakpoint at Line 4; and Line 11

% (2) run & continuous to run the program

clear all;clc;

% excludes zero roots!

* steps = 0; % iteration counter

x = input( 'Initial guess: '); % estimate of root

re = 1e-8; % required relative error

myrel = 1;

while myrel > re & (steps < 20)

xold = x;

* x = x - f(x)/df(x);

steps = steps + 1;

disp( [x f(x)] )

myrel = abs((x-xold)/x);

end

if myrel <= re

disp( 'Zero found at' )

disp( x )

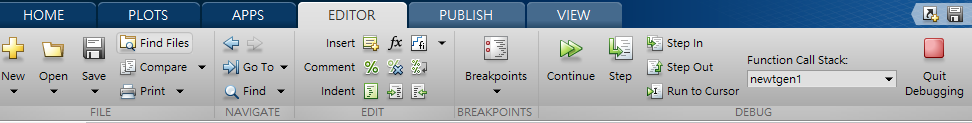
else

disp( 'Zero NOT found')

end

%================================================================================================

* Quit debugging



Click this button

% 7.7 RECURSION

% The factorial function may be written recursively in

% an M-file fact.m like this:

function y = fact(n)

% FACT Recursive definition of n!

if n > 1

y = n \* fact(n-1);

else

y = 1;

end;

%================================================================================================