% 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
```

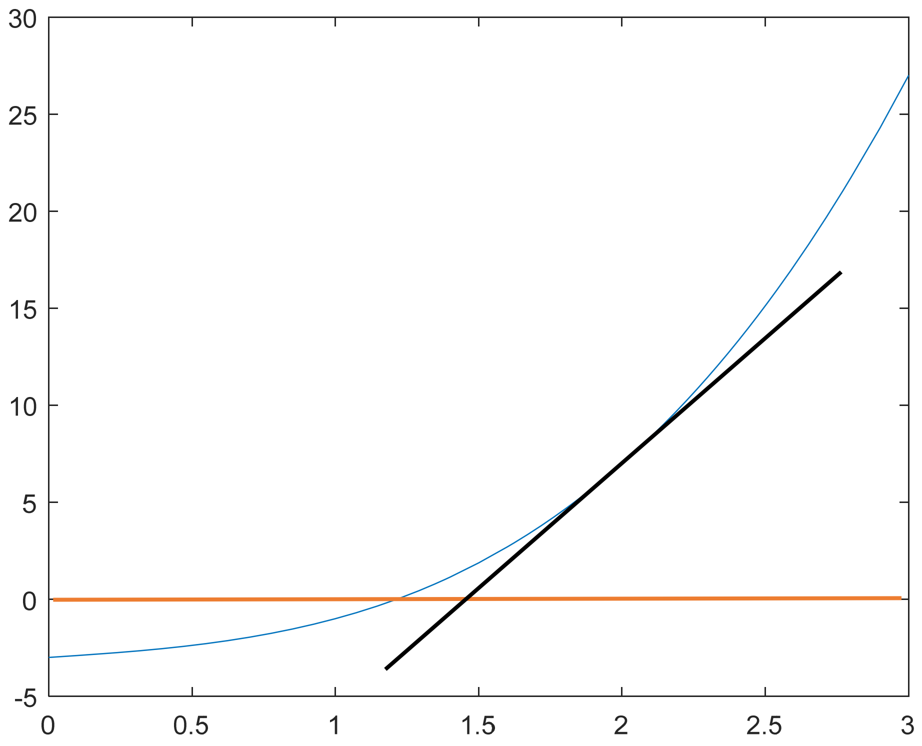
%===

%% To rewrite newton's method in book 7.1 by using (for & if break)

Ans:

```
% excludes zero roots!

steps = 0; % iteration counter
x = input( 'Initial guess: '); % estimate of root
re = 1e-8; % required relative error
myrel = 1;
for steps=1:19
    xold = x;
    x = x - f(x)/df(x);
    steps = steps + 1;
    disp( [x f(x)] )
    myrel = abs((x-xold)/x);
    if myrel <= re
        break;
    end
end
if myrel <= re
    disp( 'Zero found at' )
    disp( x )
else
    disp( 'Zero NOT found')
end
```


30
25
20
15
10
5
0
-5
0
0.5
1
1.5
2
2.5
3

```
%      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.
(6) Local variable : exist inside the function
    Here, avg, & stdev are the local variable, they are available only inside the stats subfunction.
(7) 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.

(8) Function that do not return values

```
function stars(n)
asteriks = char(abs('*')*ones(1,n));
disp( asteriks );
```

(9) Vector arguments

```
%      Vector arguments
function d = dice( n )
d = floor( 6 * rand(1, n) + 1 );      % d is a vector
```

(10) Input arguments is “Call by Value” to pass the values to the sunfunction.
(11) Checking the number of function arguments
Add a common line ‘disp(nargin)’, you can find the number of the input.
(12) Call a function with any number of input & output arguments

- Exec: Exec 7.5 in p.179 (a) write as a .m function (b) keep the input (x value)-output in the mean function and call this function as the subfunction.

```
%% Debugging a script Newton¡¦s method in general
clear all;clc;
value=1; % initial value
x=3;
re = 1e-6; % required relative error
step=1;
myerr = 1;
while (myerr > re)
    value_old=value;
    value =value+(x.^(step))./(fact(step));
    disp( [step value] )
    step=step+1;
    myerr = value-value_old;
end

disp( [x value] )
```

% % 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)

% To defined function handle
Fhandle=@ (arglist) expression
For example:

Two kinds of functions: anonymous function, and parameterized function.

(1) anonymous function, the parameters value is anonymous (hided in expression)

Fh1= @ (x) 4*x.^2-50*x+2;

Fh2= @ (x,y) sqrt(x.^2+y.^2)

Fh3=@(x) (x-1.5).^2;

(2) parameterized function: you can change parameter every time you call it

```
fh2 = @(x,c) (x-c).^2;% the value of the parameter is not defined
yet
c = 1.5;
fh3 = @ (x) fh2(x,c);  % Now the value is defined
d=fh3(3)
```

% 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.
Two function forms for the definition of the function handle:
              X = fminbnd(@(x) sin(x)+3,2,8)
                y=2:0.1:8;
             plot(y,sin(y)+3)
%----------------------------------
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);    % return the function value at 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)

- Exer

  (a) Find the minimum value for the function $y = 1 + e^{-0.2x}\sin(x+2)$, for the interval of $0 < x < 10$. (Ans: (x,y)=(2.515, 9.0). (Use fminbnd)

  (b) Use fplot to plot this function for the interval of $0 < x < 10$.

  (c) Write this function as the parametric form, that is

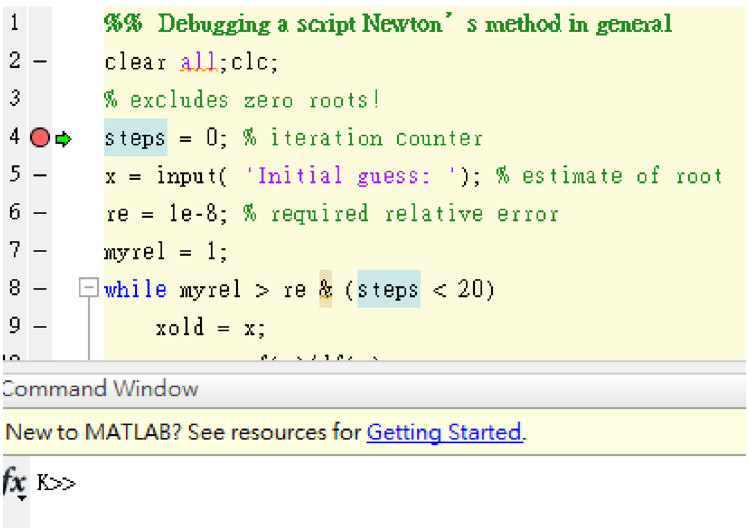
  $y = 1 + e^{-0.2x}\sin(x+c)$, where c is the parameter.

  Do the same thing as (a) & (b), by given c=2.5.

%% 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.



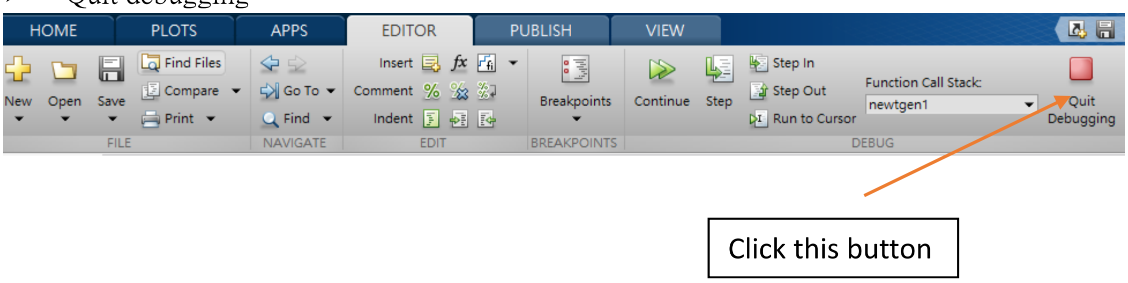
```
%%    Check the procedure in p. 175 for the debugging
% Debugging a script Newton's method in general
% (1) set breakpoint at Line 4; and Line 11
% (2) run & continuous to run the program

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



% Sec 7.7 RECURSION Call by one function itself

%   The factorial function may be written recursively in
%   an M-file fact.m like this:

n=10
call fact(n)

```
function y = fact(n)
% FACT Recursive definition of n!
if n > 1
    y = n * fact(n-1);
else
    y = 1;
end;
```

n=9

```
function y = fact(n)
% FACT Recursive
definition of n!
if n > 1
    y = n * fact(n-1);
else
    y = 1;
```

n=8

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

- Exec 7.8 in p.180 fibonacci number

```
function y = fib(n)
fib(1)=1;
for i=2:n
    if (i<3)
        fib(i)=1+fib(i-1);
    else
    fib(i)=fib(i-2)+fib(i-1);
    end
    disp(fib(i));
end
```

1. To rewrite newton's method in book 7.1 by using (for & if break)

Ans:

```
% excludes zero roots!

steps = 0; % iteration counter
x = input( 'Initial guess: '); % estimate of root
re = 1e-8; % required relative error
myrel = 1;
for steps=1:19
    xold = x;
    x = x - f(x)/df(x);
    steps = steps + 1;
    disp( [x f(x)] )
    myrel = abs((x-xold)/x);
    if myrel <= re
        break;
    end
end
if myrel <= re
    disp( 'Zero found at' )
    disp( x )
else
    disp( 'Zero NOT found')
end
```

2. Function handle: (a) Find the minimum value for the function $y = 1 + e^{-0.2x}\sin(x+2)$, for the interval of $0 < x < 10$. (Ans: (x,y)=(2.515, 9.0). (Use fminbnd)
   (b) Use fplot to plot this function for the interval of $0 < x < 10$.
   (c) Write this function as the parametric form, that is
   $y = 1 + e^{-0.2x}\sin(x+c)$, where c is the parameter.
   Do the same thing as (a) & (b), by given c=2.5.

3. Exer in textbook 7.2,7.4,7.5,7.6, 7.8,7.9.

EXERCISES

7.1 Change the function `stars` of Section 7.2 to a function `pretty` so that it will draw a line of any specified character. The character to be used must be passed as an additional input (string) argument, e.g., `pretty(6, '$')` should draw six dollar symbols.

7.2 Write a script `newquot.m` which uses the Newton quotient $[f(x+h) - f(x)]/h$ to estimate the first derivative of $f(x) = x^3$ at $x = 1$, using successively smaller values of h: 1, 10^{-1}, 10^{-2}, etc. Use a function M-file for $f(x)$.
Rewrite `newquot` as a function M-file able to take a handle for $f(x)$ as an input argument.

7.3 Write and test a function `double(x)` which doubles its input argument, i.e., the statement `x = double(x)` should double the value in `x`.

7.4 Write and test a function `swop(x, y)` which will exchange the values of its two input arguments.

7.5 Write your own MATLAB function to compute the exponential function directly from the Taylor series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

The series should end when the last term is less than 10^{-6}. Test your function against the built-in function `exp`, but be careful not to make x too large—this could cause rounding error.

7.6 If a random variable X is distributed normally with zero mean and unit standard deviation, the probability that $0 \le X \le x$ is given by the standard normal function $\Phi(x)$. This is usually looked up in tables, but it may be approximated as follows:

$$\Phi(x) = 0.5 - r(at + bt^2 + ct^3),$$

where $a = 0.4361836$, $b = -0.1201676$, $c = 0.937298$, $r = \exp(-0.5x^2)/\sqrt{2\pi}$, and $t = 1/(1 + 0.3326x)$.
Write a function to compute $\Phi(x)$, and use it in a program to write out its values for $0 \le x \le 4$ in steps of 0.1. Check: $\Phi(1) = 0.3413$.

7.7 Write a function

```
function [x1, x2, flag] = quad( a, b, c )
```

which computes the roots of the quadratic equation $ax^2 + bx + c = 0$. The input arguments `a`, `b` and `c` (which may take any values) are the coefficients of the quadratic, and `x1`, `x2` are the two roots (if they exist), which may be equal. See Figure 3.3 in Chapter 3 for the structure plan. The output argument `flag` must return the following values, according to the number and type of roots:

0: no solution ($a = b = 0$, $c \ne 0$);

1: one real root ($a=0$, $b\neq 0$, so the root is $-c/b$);
2: two real or complex roots (which could be equal if they are real);
99: any x is a solution ($a=b=c=0$).
Test your function on the data in Exercise 3.5.

7.8 The Fibonacci numbers are generated by the sequence

$$1, 1, 2, 3, 5, 8, 13, \ldots$$

Can you work out what the next term is? Write a recursive function `f(n)` to compute the Fibonacci numbers F_0 to F_{20}, using the relationship

$$F_n = F_{n-1} + F_{n-2},$$

given that $F_0 = F_1 = 1$.

7.9 The first three Legendre polynomials are $P_0(x) = 1$, $P_1(x) = x$, and $P_2(x) = (3x^2 - 1)/2$. There is a general *recurrence* formula for Legendre polynomials, by which they are defined recursively:

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0.$$

Define a recursive function `p(n,x)` to generate Legendre polynomials, given the form of P_0 and P_1. Use your function to compute `p(2,x)` for a few values of `x`, and compare your results with those using the analytic form of $P_2(x)$ given above.

APPENDIX 7.A SUPPLEMENTARY MATERIAL

Supplementary material related to this chapter can be found online at http://dx.doi.org/10.1016/B978-0-08-100877-5.00008-6.

```
%% %7.1
pretty(6, '$') %Type it in the command window 得  $$$$$$

% Function file pretty.m
function pretty(n, ch)
line = char(double(ch)*ones(1,n));%char  將 ASCII 碼轉回字串形式
disp(line)

%% %7.2

newquot(1) %Type it in the command window

% Function file f.m
function y=f(x)
y=x^3;

% Function file newquot.m
function newquot(x)
h = 1;
for i = 1 : 10
        df = (f(x + h) - f(x)) / h;
        disp( [h, df] );
        h = h / 10;
end

%Type the following three statements in the command window
fn = @f;
x = 2;
newquot_handel(fn,x)

%Using function handels to pass the function as the input parameter.
% another example
fn = @sin; % sine function
x=0.3;
newquot_handel(fn,x) % derivative of sin(0.3) = cos(0.3) = 0.9553

% Function file newquot_handel.m
function newquot_handel(fn,x)
```

```
h = 1;
for i = 1 : 10
	df = (feval(fn, x + h) - feval(fn, x)) / h;
	disp( [h, df] );
	h = h / 10;
end
```

```
%% %7.3
y = double(3)% Type it in the command window 得  y=6

% Function file double.m
function y = double(x)
y = x * 2;
```

```
%% %7.4
[xout, yout] = swop(4, 5);% Type it in the command window, [xout, yout]=[5 4]

% Function file swop.m
function [xout, yout] = swop(x, y)
xout = y;
yout = x;
```

```
%% %7.5
ex=exponential(2)
%Type it in the command window,  得    ex=6.3891
exp(2) % Matlab built-in function

% Function file exponential.m
function ex=exponential(x)
y=1; i=1; z=1;
while z>=10^-6
	z=x^i/factorial(i);%factorial 階乘
	i=i+1;
	y=y+z;
```

```
end
ex=y;

%% %7.6
% Script file
for i = 0 : 0.1 : 4
    disp( [i, phi(i)] );
end

% Function file phi.m
function y = phi(x)
a = 0.4361836;
b = -0.1201676;
c = 0.937298;
r = exp(-0.5 * x ^2) / sqrt(2 * pi);
t = 1 / (1 + 0.3326 * x);
y = 0.5 - r * (a * t + b * t ^2+ c * t ^ 3);

%% %7.7
[x1, x2, flag] = quad( 0.5, -1, 2)%  得[x1, x2, flag]=[0.5000      -1.0000        2.0000]
% Function file quad.m

function [x1, x2, flag] = quad( a, b, c )

if a==0 & b==0 & c==0
    flag = 99; x1=0; x2 = 0;
elseif a==0 & b==0
    flag = 0; x1=NaN; x2 = NaN;
elseif a==0
    flag = 1;
    x1 = -c/b;
    x2 = NaN;
else
    x1 = (-b + sqrt(b^2 - 4*a*c))/(2*a);
    x2 = (-b    - sqrt(b^2 - 4*a*c))/(2*a);
    flag = 2;
end
```

```
%% %7.8
% Type the following four statements in the command windpow
y = zeros(1,12);
for k = 1:12
y(k)=f(k);
end
display(y);

% Function file f.m
% Function file f.m
function y = f(n)
if n > 2
        y = f(n - 1) + f(n - 2);
else
        y = 1;
end
```