MATLAB Programs

ENGR 151, Lecture 22: 26 Nov 14

Announcements

- Project 7 due tonight 11pm
- Project 8 out soon

Which is True about Function Files?

A. May contain only one function definition
B. May contain any number, but only the first can be called from command window
C. May contain any number, but the second one can only be called by the first
D. May contain any number, any can be called from anywhere
E. None of the above

Subfunctions

- A function (.m) file may contain multiple function definitions
  - The first is the primary function and should correspond to the name of the file
  - Subsequent functions are called subfunctions
- Primary and subfunctions may call each other
  - regardless of order of appearance in file
  - only primary function may be called from main scope
- Each (sub)function has its own scope for local variables

Nested Function Definitions

- can also define a function *within* another function definition

```
function [a, b] = SomeFunction (x, y)
…
    function c = NestOne(z)
    …
    end
…
end
```

NestOne is a nested function of SomeFunction
SomeFunction is the nesting function of NestOne

Example: Recursive Helper

```
function b = IsPalindrome( s )

  PalindromeAux(1,length(s));

  function PalindromeAux(low,high)
    if low >= high
      b = true;
    elseif s(low) ~= s(high)
      b = false;
    else PalindromeAux(low+1,high-1);
    end
  end

end
```

What if PalindromeAux were a Subfunction?

```
function b = IsPalindrome( s )
  PalindromeAux(1,length(s));
end
function PalindromeAux(low,high)
  if low >= high
    b = true;
  elseif s(low) ~= s(high)
    b = false;
  else
    PalindromeAux(low+1,high-1);
  end
end
```

- A It would work fine.
- B It would run, but produce no result.
- C Error: b undefined
- D Error: s undefined
- E None of the above

Nested Functions: Scope

- Unlike subfunctions, nested functions share the scope of their nesting function
  - can access and modify variables in nesting function's scope
  - variables introduced in nested function are accessible within nesting function's scope
  - formal parameters (input and output) of nested function *not* accessible to nesting function
- A function can call its nested functions
- A nested function can call its nesting function (recursion), as well as any functions its nesting function can call

Multiple Nested Functions (Same Level)

```
function [a, b] = SomeFunction (x, y)
…
    function c = NestOne(z)
    …
    end
…
    function d = NestTwo(w)
    …
    end
…
end
```

NestOne and NestTwo are nested functions of SomeFunction

SomeFunction is the nesting function of NestOne and NestTwo

Multilevel Function Nesting

```
function y = A (a1,a2)
…
   function z = B(b1,b2)
   …
      function w = C(c1,c2)
      …
      end
   end
   function u = D(d1,d2)
   …
      function h = E(e1,e2)
      …
      end
   end
…
end
```

function scope

- What can A call?
  - B, D (and itself, of course)
- What can B call?
  - C, A, D, B
- What can't C call?
- What can't D call?

Multilevel Function Nesting

```
function y = A (a1,a2)
…
    function z = B(b1,b2)
    …
        function w = C(c1,c2)
        …
        end
    end
    function u = D(d1,d2)
    …
        function h = E(e1,e2)
        …
        end
    end
…
end
```

variable scope

- Which fns have access to a1, a2, and y?
  - all of them
- other formal parameters?
- other variables (e.g., if B introduces b3)?
  - Accessible to A and C, and also D and E but only if A refers to b3.

Example: Newton's Method

- Solve for *x*:

$$f(x) = 0$$

- Approach:
  - Guess a value, g
  - If f(g) close enough to zero, return g
  - Otherwise, generate an improved guess, repeat…
- Guess improvement formula:

$$g \leftarrow g - \frac{f(g)}{f'(g)}$$

```
double squareRoot(double s, double eps)
{
  double guess = 1.0;
  double residual = abs(guess * guess - s);
  while ( residual > eps ) {
    guess = newGuess(guess,s);
    residual = abs(guess * guess - s);
    }
  return guess;
}
```

Newton's Method in MATLAB

```
function root = newton(guess)
% newton help line goes here

  function y = f(x)
    y = x^3 + x - 3;
  end

  function y = df(x)
    y = 3*x^2 + 1;
  end
…
```

Nested "helper" functions, defining equation to be solved and its derivative.

(to be continued)

adapted from Hahn & Valentine, *Essential MATLAB*

Newton's Method (continued)

```
…
steps = 0;
myrel = 1;
re = 1e-8;     %  relative error threshold

while (myrel > re) & (steps < 20)
  oldguess = guess;
  guess = guess - f(guess)/df(guess);
  steps = steps + 1;
  fprintf('guess: %8.4g; f(guess): %8.4g\n',...
           guess, f(guess));
  myrel = abs((guess-oldguess)/guess);
end

end
```

Functions as Input

- Subfunction and nested function facility make it relatively easy to encapsulate helper fns with their primary function
- What we would really like, for an example like `newton`, is to supply the functions `f` and `df` as part of the *input*.
  - So far, only way to invoke a function is to call based on name associated with definition in program text
  - Passing functions as input requires a way to construct a function as a data element
- …here's where function handles come in

Function Handles

- Create a function handle by prepending `@` before a function name.
- For example, suppose we have defined:

```
function y = myfunc(x)
  y = x^3 + x - 3;
```

  - then `@myfunc` is a function handle, and:
  - `feval(@myfunc,`*expr*`)` behaves just like `myfunc(`*expr*`)`
  - if we assign `myf2 = @myfunc`, then `myf2(`*expr*`)` also behaves just like `myfunc(`*expr*`)`

Newton's Method with Function Inputs

```
function root = newton2(guess,f,df)
% newton2 help line goes here

steps = 0; myrel = 1;
re = 1e-8;    %  relative error threshold

while (myrel > re) & (steps < 20)
  oldguess = guess;
  guess = guess - f(guess)/df(guess);
  steps = steps + 1;
  fprintf('guess: %8.4g; f(guess): %8.4g\n',...
           guess, f(guess));
  myrel = abs((guess-oldguess)/guess);
end

end
```

Function that takes function inputs called a higher-order function, or sometimes a "function function"

Anonymous Functions

- Function objects *without* names
- Form:

  @ (paramList) expr

  - paramList: comma-separated list of arguments (just as in function definitions)
  - expr: a MATLAB expression
- To associate with a function handle variable:

  handlevar = @ (paramList) expr

Calling Anon Fns with Handle Variables

- Examples:

```
calcThis = @ (m) m^2 - 2*m + 1;
calcThis(5)       →    16
calcThat = @ (m,n,p) n - 2*p + m;
calcThat(5,4,3)   →    3
calcIt = @ (m) m.^2;
calcIt([3 4 5])   →    [9 16 25]
```

Exercise

- What is b after executing the following code?

```
func = @ (x,y) sqrt(x.^2-y.^2);
a = func([5 5], [3 3]);
b = 3 + func(a,a);
```

A	B	C	D	E
4 4	0 0	5 5	3 3	0 3

Using Anonymous Functions

- Example:

```
f = @(x) x.^3 + x - 3
df = @(x) 3*x.^2 + 1
```

- Use in expressions:

```
z = 0:0.05:2;
plot(z, f(z))
```

- Pass into functions:

```
newton2(1,f,df)
```

- Assigning handle to name not actually needed:

```
newton2(1, @(x) x.^3 + x - 3, ...
           @(x) 3*x.^2 + 1 )
```

Anonymous Functions: Scope

@ (paramList) expr

- Variables in paramList have scope local to anonymous function
- Body expr may also refer to any variables or functions available in scope where anonymous function defined
  - External references evaluated at *definition* time

Try this in MATLAB

```
x = 999;
z = -111;
myFunc = @(z) sqrt(x+z);
y = myFunc(601)
x = 24;
y2 = myFunc(601)
myFunc2 = @(y) 2*myFunc(y);
y3 = myFunc2(601)
myFunc = @(x) x^2;
y4 = myFunc2(601)
```

Functions with Multiple Return Values

- Suppose `fun0` is defined to return two values
- Various ways to use these values:

```
a = fun0(x)
[a] = fun0(x)
```

one return value (first)

```
[~,a] = fun0(x)
```

one return value (second)

```
[a,b] = fun0(x)
```

two return values

Built-In `size` Function

- returns the size of each dimension of input array
- various ways to use return value (for matrix M):

```
sz = size(M)
nrows = sz(1)
ncols = sz(2)
```

one return value (a vector)

```
[nrows ncols] = size(M)
```

two return values (scalars)

```
nrows = size(M, 1)
ncols = size(M, 2)
```

two-argument version

Flexible Function Parameters

- Many MATLAB functions can handle varying numbers of arguments and return values
  - Examples: `ones`, `fopen`, `fprintf`, `save`, …
  - Purpose:
    - Provide default values
    - Varying dimensions
    - Varying number of operands
- User-defined functions may also offer this flexibility

Supporting Varying Number of Parameters

- MATLAB exposes two key variables:
  - `nargin`: number of input arguments passed to the function
  - `nargout`: number of return values requested by the caller
- Idea: condition function behavior on these values

```
function myFunc (optArg)

  if nargin == 0
    optArg = defaultVal
  end
…
```

Example with Optional Default

```
function val = simulate(time, startval)
  val = zeros(time);
  if nargin<2 | isempty(startval)
    val(1) = 0;
  else
    val(1) = startval;
  end
  for t = 2:time
    val(t) = SomeFunction(val(t-1));
  end
end
```

Example with Optional Return Value

```
function [m,v]=MeanVar(X)
 % MeanVar computes mean and variance

 n = size(X,1);    % #rows
 m = mean(X);
 if nargout>1
     temp = X - ones(n,1)*m;
     v = sum(temp.*temp)/(n-1);
 end
```