Chapter 7
Advanced Features of User-Defined Functions
Function Functions

- **Function function** is a function whose input arguments include the names or handles of other functions.
  - Syntax: `func(handler, range);`
    - Where `func` is a function function,
    - `handler` is the input function, and
    - `range` is the range of the `handler`.
  - For example:
    - `feval`: evaluate a named function
    - `fplot`: plot a named function
    - `fzero`: find a zero of a named function
Function Functions

Three ways to define function handlers:

- Define handlers as strings, e.g.
  \[ cs = '\sin(x) - \cos(3\times x)' \];
- Define handlers as inline functions, e.g.
  \[ fh = \text{inline} ('\sin(x) - \cos(3\times x)') \];
- Define handlers as anonymous functions, e.g.
  \[ an = @(x)(\sin(x) - \cos(3\times x)) \];
Function Functions

- For example:

```matlab
>> x=0:pi/40:2*pi;
>> y=cos(x)-sin(3*x);
>> plot(x,y,:ro); grid
>> fzero('cos(x)-sin(3*x)',3.5)
ans =
    3.5343
>> fzero('cos(x)-sin(3*x)',2)
ans =
    1.9635
>> fh=inline('cos(x)-sin(3*x)');
>> feval(fh,[1,3])
ans =
    0.3992   -1.4021
```
Function Functions

- More example: to draw

\[ f(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6 \]
Function Functions

- To create a function function \texttt{myplot}.
  - Calling syntax: \texttt{myplot(handler, limits)}
  - \texttt{handler} is a function string,
  - \texttt{limits} is the range of function in \([\text{min}, \text{max}]\).
  - The key is to convert the given function string to a function handler so the build-in function \texttt{feval} can be used.
Function Functions

```matlab
function myplot(handle,limits)
    % myplot
    if nargin ~= 2
        error('nargin must be 2');
    end
    dim = size(limits);
    if dim(1) ~= 1 || dim(2) ~= 2 ...
        || limits(2) <= limits(1)
        error('wrong form of limits');
    end
    cap = class(handle);
    switch(cap)
    case 'char'
        fh = inline(handle);
        cap = handle;
        opstr = '-bx';
    case 'function_handle'
        fh = handle;
        opstr = ':ro';
    case 'inline'
        fh = handle;
        opstr = ':m.';
    otherwise
        error('wrong class of handle');
    end
    nps = 100;
    x = linspace(limits(1), limits(2), nps);
    y = zeros(size(x));
    for i = 1:length(x)
        y(i) = feval(fh, x(i));
    end
    plot(x, y, opstr)
    grid
    xlabel('x'); ylabel(cap);
end
```
Function Functions

Command Window

```matlab
>> edit myplot
>> cs='cos(x)-sin(3*x)';
>> fh=inline('sin(x)-cos(3*x)');
>> an=@(x)(cos(x)-sin(3*x));
>> subplot(3,1,1)
>> myplot(cs,[0,2*pi])
>> subplot(3,1,2)
>> myplot(fh,[0,2*pi])
>> subplot(3,1,3)
>> myplot(an,[0,2*pi])
```

---

![Figure 1](image-url)
Subfunctions

- It is possible to place more than one function in a single file.
- The topmost function is called **primary function**, while the rest are called **local functions** or **subfunctions**.
- All functions begin with their own function definition line.

```plaintext
function primaryFunc(inarg_list)
primaryFunc statements
z=funcSub(x, y);
Return

function c=subFunc (a, b)
subFunc statements
return
```
Subfunctions\(^2\)

- For example: Create a function `mystats` to compute the mean and the median of a given array. Functions `mean` and `median` are two local functions of `mystats`. 

![Diagram showing `mystats`, `mean`, and `median` functions]
Subfunctions
Private Functions

- **Private functions** are functions that reside in subdirectories with the special name `private`.
  - A private function can be called in an M-file that resides in the directory immediately above that private subdirectory.
  - For example, assume the directory `cwd` is the current working directory and it has a sub-directory called `private`. It can contain functions that only the functions in `cwd` can call.
Nested Functions

- You can define functions within the body of any function. These are said to be nested within the outer function.
- A nested function has its own workspace. It also has access to the workspaces of all functions in which it is nested.
- For example, a variable that has a value assigned to it by the primary function can be read or overwritten by a function nested at any level within the primary.
Nested Functions

An illustration of a nested function:

```matlab
function out=tstnested(in1, in2)
% test nested function
% out=in1^2+sqrt(in2)
% ^2 is done by a nested func
% sqrt is done by a sub func

key='accessible';
out=nested(in1)+sub(in2);

function p=nested(x)
p=x.^2;
key2=key;
fprintf('nested:key=%s\n',key2);
end

fprintf('%d^2+sqrt(%d)=%g\n',..., in1, in2, out);
end

function q=sub(y)
q=sqrt(y);
try
    key2=key;
catch
    key2='undefined';
end
fprintf('sub: key=%s\n',key2);
end
```

```plaintext
>> edit tstnested
>> tstnested(4,9)
nested:key=accessible
sub: key=undefined
4^2+sqrt(9)=19.00
ans = 19
```

```plaintext
fx >> |
```
## Order of Function Evaluation

<table>
<thead>
<tr>
<th>Order</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Checks to see if there is a nested function with the specified name. If so, it is executed.</td>
</tr>
<tr>
<td>2</td>
<td>Checks to see if there is a subfunction with the specified name. If so, it is executed.</td>
</tr>
<tr>
<td>3</td>
<td>Checks to see if there is a private function with the specified name. If so, it is executed.</td>
</tr>
<tr>
<td>4</td>
<td>Checks to see if there is a function with the specified name in the current directory. If so, it is executed.</td>
</tr>
<tr>
<td>5</td>
<td>Checks to see if there is a function with the specified name on the MATLAB path. MATLAB will execute the first file with the right name found on the path.</td>
</tr>
</tbody>
</table>
Recursive Functions

- Recursion is a devious construction which allows a function to call itself.
- The function may call itself forever; so a terminating statement is necessary.
- An individual workspace will be created each time a function call itself. Therefore, it needs extra memory space to accommodate the overhead of the recursion.
- For example:

  \[ n! = n \times (n-1)! \]
  \[ \text{factorial}(n) = n \times \text{factorial}(n-1) \]
Recursive Functions

- So, each loop can be rewritten as a recursive function. For example:
  \[ n! = n \times (n-1)! \]

  It can be represented as:
  \[ \text{factorial}(n) = n \times \text{factorial}(n-1) \]

% n-factorial use a loop
res = 1;
for i = n:-1:2
  res = res * i;
end

% recursive n-factorial
function res = factorial(n)
  if n < 2
    res = 1;
  else
    res = n * factorial(n-1);
  end
More Plotting Functions

- `ezplot`
- `fplot`
Homework Assignment #14

- 7.9 Exercises
  - Page 251: 7.4, 7.5, 7.12, 7.18
- This homework is for your reference.