Function M-File

Contents

Function M-File

Basics

- A *function* is a piece of code which
  - performs a specific task;
  - has specific input and output arguments;
  - is encapsulated to be independent of the rest of the program.
- In MATLAB, functions are defined in .m files just as scripts.
- The name of the file and that of the function must coincide.

Why Functions?

Two Important Principles

- A piece of code should only occur once in a program. If it tries to appear more times, put it into its own function.
- Each function should do precisely **one** task well. If it has to carry out a number of tasks, separate them into their own functions.

Benefits of writing functions:

- elevated reasoning by hiding details
- facilitates top-down design
- ease of software management

Writing a Function

A function m-file must be written in a specific manner.

- When there is no input/output argument

```
function myfun()
  ....
end    % <-- optional
```

- Where there are multiple input and output arguments

```
function [out1, out2, out3] = myfun(in1, in2, in3, in4)
  ....
end    % <-- optional
```

Calling a Function

- If the function m-file `myfun.m` is saved in your current working directory[1], you can use it as you would use any other built-in functions:

```
% when no input/output argument is required
myfun
```

or

```
% multiple inputs/outputs
[out1, out2, out3] = myfun(in1, in2, in3, in4)
```

- When not all output arguments are needed:

```
out1 = myfun(in1, in2, in3, in4)          % only 1st output
[~, ~, out3] = myfun(in1, in2, in3, in4) % only 3rd output
```

Note that tilde (~) is used as a placeholder.

Practical Matters: Specification

- The comments written below `function` header statement

```
function ... = myfun( ... )
% MYFUN: this awesome function calculates ...
  ....
end
```

  can be easily accessed from the command line using `help myfun`.

- Use this feature to write the function specification such as its purpose, usage, and syntax.
- Write a clear, complete, yet concise specification.

Properties of Function M-Files

- The variable names in a function are completely isolated from the calling code. Variables, other than outputs, used inside a function are unknown outside of the function; they are *local variables*.
- The input arguments can be modified within the body of the function; no change is made on any variables in the calling code. (“ *pass by value* ” as opposed to “ *pass by reference* ”)
- Unlike script m-files, you **CANNOT** execute a function which has input arguments using the RUN icon.

Example: Understanding Local Variables

- The function on the right finds the maximum value of a vector and an index of the maximal element.
- Run the following on the Command Window. Pay attention to `m`.

```
>> m = 33;
>> x = [1 9 2 8 3 7];
>> [M, iM] = mymax(x)
>> disp(m)
```

```
function [m, el] = mymax(x)
    if isempty(x)
        el = [];
        m = [];
        return
    end
    el = 1;
    m = x(el);
    for i = 2:length(x)
        if m < x(i)
            el = i;
            m = x(el);
        end
    end
end
```

Example: Understanding Pass-By-Value

Consider the function

$$f(x) = \sin|x| \, \frac{e^{-|x|}}{1+|x|^2} + \frac{\ln(1+|x|)}{1+2|x|}$$

written as a MATLAB function:

```
function y = funky(x)
    x = abs(x);  % x is redefined to be abs(x)
    y = sin(x).*exp(-x)./(1 + x.^2)...
        + log(1 + x)./(1 + 2*x);
end
```

Confirm that the function does not affect `x` in the calling routine:

```
>> x = [-3:3]';
>> y = funky(x);
>> disp([x, y])
```

Quiz

Question

Script.

```
x = 2;
x = myfun(x);
y = 2*x
```

Function.

```
function y = myfun(x)
    x = 2*x;
    y = 2*x;
end
```

What is the output when the script on the left is run?

1. `y = 2`
2. `y = 4`
3. `y = 8`
4. `y = 16`

Example: Spiral Triangle

Description of Problem

Problem

Given m and θ (in degrees), draw m equilateral triangles so that for any two successive triangles,

- the vertices of the smaller triangle are lying on the sides of the larger;
- the angle between the two triangles is θ (degrees)

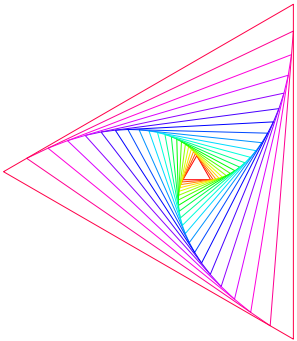


Figure 1: A spiral triangle with $m = 21$ and $\theta = 4.5°$.

Generation and Transformation of Polygons

Let $(x_j, y_j), j = 1, 2, \ldots, n$ be the coordinates of vertices of an n-gon.

- To plot the polygon:

```
x = [x1 x2 ... xn x1]; % note: x1 and y1 are repeated at
y = [y1 y2 ... yn y1]; % end to enclose the polygon.
plot(x, y)
```

- To plot the polygon obtained by scaling the original by a factor of `s`:

```
plot(s*x, s*y)
```

- To plot the polygon obtained by rotating the original by an angle `theta` (in degrees) about the origin:

```
V = [x; y];                          % all vertices
R = [cosd(theta) -sind(theta);       % 2-D rotation matrix
     sind(theta)  cosd(theta)];
Vnew = R*V;                          % rotated vertices
plot(Vnew(1,:), Vnew(2,:))
```

Special Case: Inscribed Regular Polygons

- The vertices of the *regular* n-gon inscribed in the unit circle can be found easily using trigonometry, *e.g.*,

$$(x_j, y_j) = \left(\cos \frac{2\pi(j-1)}{n}, \sin \frac{2\pi(j-1)}{n} \right), \quad j = 1, \ldots, n.$$

- Thus we can plot it by

```
theta = linspace(0, 360, n+1);
V = [cosd(theta);      % 2-by-(n+1) matrix whose cols are
     sind(theta)];     % coordinates of the vertices
plot(V(1,:), V(2,:)), axis equal
```

- To stretch and rotate the polygon:

```
Vnew = s*R*V;
plot(Vnew(1,:), Vnew(2,:)), axis equal
```

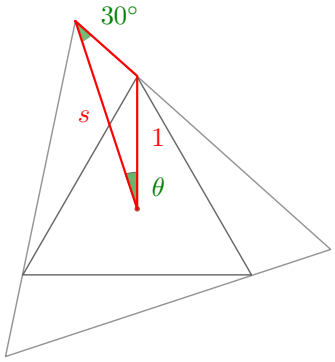
Scale to Spiral

To create the desired spiraling effect, the scaling factor must be calculated carefully.

- Useful:

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c}$$

- Compute the scaling factor s:



Put All Together

```
m = 21; d_angle = 4.5;
th = linspace(0, 360, 4) + 90;
V = [cosd(th);
     sind(th)];
C = colormap(hsv(m));
s = sind(150 - abs(d_angle))/sind(30);
R = [cosd(d_angle) -sind(d_angle);
     sind(d_angle) cosd(d_angle)];
hold off
for i = 1:m
    if i > 1
        V = s*R*V;
    end
    plot(V(1,:), V(2,:), 'Color', C(i,:))
    hold on
end
set(gcf, 'Color', 'w')
axis equal, axis off
```

Exercise: Script to Function

Question

Modify the script so that it can create a spiral n-gon consisting of m regular n-gons successively rotated by θ degrees. Then turn the script into a function m-file `spiralgon.m`.

```
function V = spiralgon(n, m, d_angle)
% SPIRALGON plots spiraling regular n-gons
% input:   n = the number of vertices
%          m = the number of regular n-gons
%          d_angle = the degree angle between successive n-gons
%          (can be positive or negative)
% output:  V = the vertices of the outermost n-gon

% Fill in the rest.
....
```

Appendix: Supplementary Notes

Local Functions

There are “general purpose” functions which may be used by a number of other functions. In this case, we put it into a separate file. However, many functions are quite specific and will only be used in one program. In such a case, we keep them in whatever file calls them.

- The first function is called the *primary* function and any function which follows the primary function is called a *local function* or a *subfunction*.

```
function <primary function>
  ....
end
function <local function #1>
  ....
end
<any number of following local functions>
```

- The primary function interact with local functions through the input and output arguments.
- Only the primary function is accessible from outside the file.

Defining/Evaluating Mathematical Functions in MATLAB

- using an anonymous function
- using a local function
- using a primary funciton
- using a nested function
- putting it *in situ*, i.e., write it out completely in place – not recommended

See `time_anon.m`.

Passing Function to Another Function

- A function can be used as an input argument to another function.
- The function must be stored as a function-handle variable in order to be passed as an argument.
- This is done by prepending " `@` " to the function names. For instance, try

```
>> class(mymax)    % WRONG
>> class(@mymax)   % CORRECT
```

When a Function Is Called

Below is what happens internally when a function is called:

- MATLAB creates a new, local workspace for the function.
- MATLAB evaluates each of the input arguments in the calling statements (if there are any). These values are then assigned to the input arguments of the function. These are now local variables in the new workspace.
- MATLAB executes the body of the function. If there are output arguments, they are assigned values before the execution is complete.
- MATLAB deletes the local workspace, saving only the values in the output arguments. These values are then assigned to the output arguments in the calling statement.
- The code continues following the calling statement.