Functions and Loops

Dr. Mihail

October 30, 2018

Functions

MATLAB syntax

Functions take input, called **arguments** and produce output. There can be any number (including none) of arguments. There can also be any number (or none) of outputs. The syntax used in MATLAB to **create** a function is:

```
function [o1, o2, ...] = function_name(arg1, arg2, arg3, ...)
  body of function
end
```

The function has to be its own .m file. **The function name has to have the same name as the file.**

Functions

Use

```
function [o1, o2, ...] = function_name(arg1, arg2, arg3, ...)
  body of function
end
```

- All output variables {o1, o2, ...} have to be **declared and initialized** in the body of the function.
- All inputs {arg1, arg2, arg3, ...} have to be **used** in the body of the function.
  - Although MATLAB is forgiving (e.g., unless inputs/outputs used in the body), it will interpret with warnings, **it is extremely poor programming practice and will be penalized in this course**.

Functions

Terminology

```
function [o1, o2, ...] = function_name(arg1, arg2, arg3, ...)
  body of function
end
```

- {o1, o2, ...} are called the output variables.
- {arg1, arg2, ...} are named differently when part of the function definition and during function call:
  - Part of function definition: **parameters**
  - During function call: **arguments**

Functions

```
function [o1, o2, ...] = function_name(arg1, arg2, arg3, ...)
  body of function
end
```

Input/output behavior

- Parameters act like **variables with assigned values in the function body**
- Outputs act like **variables with unassigned values in the function body**. In other words, they don't exist until you assign them (e.g.: `o1 = arg1^2;`).
- Inputs/outputs can be scalars, vectors, matrices or other types.

Functions

Usage

Functions can be used in, or called from:

- expressions
- other scripts
- other functions
- inside the same function (recursion)

A simple function

Below is a simple function with one input (parameter) and one output. The function simply **returns** the parameter squared.

```
function [x_squared] = custom_square(x)
  x_squared = x^2;
end
```

Functions

A simple function

Below is a simple function with one input (parameter) and one output. The function simply **returns** the parameter squared.

```
function [x_squared] = custom_square(x)
  x_squared = x^2;
end
```

Usage

From the command window:

```
>> custom_square(2)

ans =

     4
```

Another function

2D function

Below is a function with two inputs (parameters) and one output.

```
function [ elevation ] = mywave(x, y)
  elevation = sqrt(sin(x).^2 + cos(y).^2) * 2;
end
% notice element-wise operations
```

Usage

From the command window:

```
>> mywave(1, 2)

ans =

    1.8775
```

Another function

2D function

```
function [ elevation ] = mywave(x, y)
  elevation = sqrt(sin(x).^2 + cos(y).^2) * 2;
end
% notice element-wise operations
```

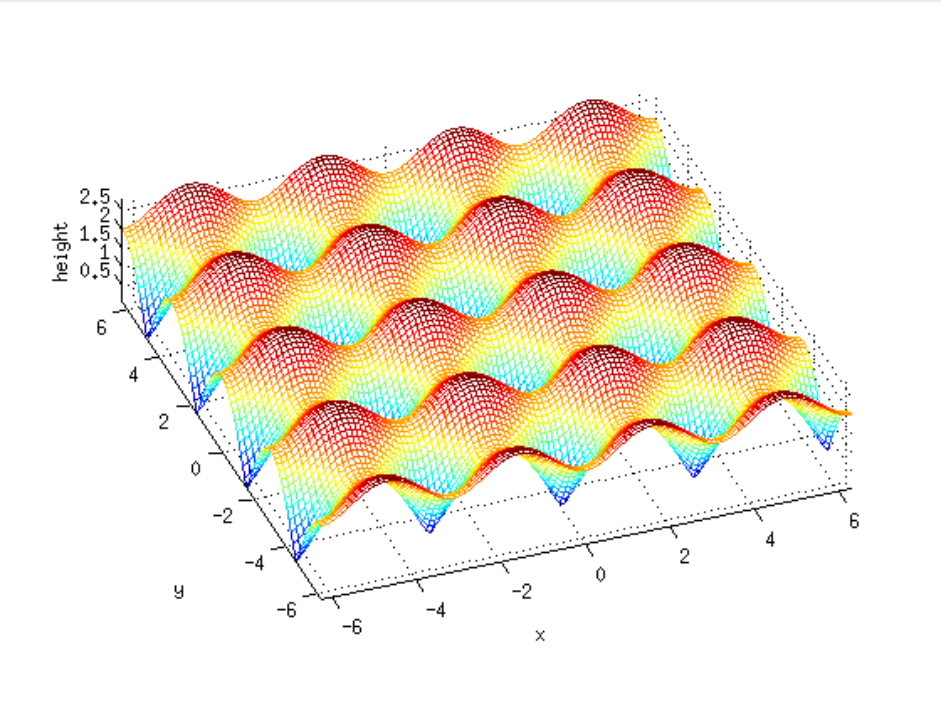
Usage

We can visualize this function by sampling $x \in [-2 * pi, 2 * pi]$ and $y \in [-2 * pi, 2 * pi]$:

```
[x, y] = meshgrid(linspace(-2*pi, 2*pi, 100), ...
    linspace(-2*pi, 2*pi, 100)); % samples x's and y's
height_map = mywave(x, y);
mesh(x, y, height_map); % plotting function
axis equal; % scales axes equally
```

Visualization

$$height = \sqrt{sin(x)^2 + cos(y)^2} * 2$$



Documenting functions

Always document your functions. In the beginning of the function body, include the following:

- Purpose: what your function does
- Pre-conditions: document the parameters, i.e., what each of them represents and their shape (scalars, vectors, matrices, etc.)
- Post-conditions: document the outputs, i.e., what each of them represents and their shape

Example

```
% purpose: wave function with fixed amplitude
% pre-conditions: x and y are matrices of the same size
% post-conditions: elevation is a matrix of the same
%     size as x and y
function [ elevation ] = mywave(x, y)
  elevation = sqrt(sin(x).^2 + cos(y).^2) * 2;
end
```

Loops

Big Idea

Loops execute statements a specified number of times. The first loop construct we will look at is the for loop.

MATLAB for loop sytnax

```
for index = values
  program statements
           :
end
```

For loop

MATLAB for loop sytnax

```
for index = values
  body
end
```

Terminology

- `body` is called the loop body. It consists of statements to be executed repeatedly.
  - one execution of the `loop body` is called an iteration.
- `index` is called the loop index. It acts like a variable that changes value every iteration.
- `values` is a **row vector**. Each element of values will be assigned, in order, to the loop index.

Counting

Let's count

```
values = 1:10; % row vector
for index = values
   fprintf('In this iteration, index=%d\n', index);
end
```

Counting

Let's count

```
values = 1:10; % row vector
for index = values
   fprintf('In this iteration, index=%d\n', index);
end
```

Square each index

```
values = 1:10; % row vector
for index = values
   fprintf('%d^2 = %d\n', index, index^2);
end
```

Nesting

Loop within a loop

You can **nest** loops, by having a for loop in another for loop's body. Below is an example to compute multiplication table up to 10:

```
for x = 1:10
    for y = 1:10
        fprintf('%d times %d = %d\n', x, y, x*y)
    end
end
```

You can nest as many loops as you need to.