L10: Recursion

When functions call themselves

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

February 08, 2017

Version: release

Announcements

Lab 03 is due on February 10 at 12 pm (noon)

The following bCourses Pages were published:

- Required functions
- Useful (but optional) functions
- Common error messages and possible causes

Today:

- Recursion

Friday:

- Discussion and practice questions (array operations versus loops)

Example of a recursive function in maths

Consider the factorial function in maths (defined over $\mathbb{N}$):

$$n \mapsto n! = \begin{cases} 1 & \text{if } n = 0 \\ 1 \times 2 \times 3 \times \cdots \times (n-1) \times n & \text{if } n > 0 \end{cases}$$

We can also define this function as:

$$n \mapsto n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

This last definition is recursive

- *i.e.* factorial is defined as a function of a factorial

Recursive functions in programming

A recursive function is a function that calls itself

A recursive function must **always** feature:

- **One or more recursive calls** *i.e.* the function calling itself
  - Otherwise the function is not recursive
- **A "base case"**, which is resolved without the function calling itself
  - Otherwise the function would keep calling itself indefinitely

A recursive implementation of factorial in Matlab

```
function [result] = my_factorial(n)

% Returns the value of n! (n should be an
% integer of class double).
%
% This implementation uses recursion.

% The base case is 0!, which is equal to 1
if n == 0

    result = 1;

else

    % Here comes the recursive call...
    result = n * my_factorial(n-1);

end

end
```

Practice question on recursion

Consider the function:

```
function [y] = my_recursion(x)
if x > 5
    y = x + my_recursion(x/2);
else
    y = 2;
end
end
```

What would the value of the variable "var" be after executing the following commands?

```
>> number = 32;
>> var = my_recursion(number);
```

(A) 58

(B) 32

(C) 2

(D) 48

Practice question on recursion: explained

my_recursion(32) = ? = 58

my_recursion(32) = 32 + my_recursion(16) = 32 + 26 = 58

my_recursion(16) = 16 + my_recursion(8) = 16 + 10 = 26

my_recursion(8) = 8 + my_recursion(4) = 8 + 2 = 10

my_recursion(4) = 2

```
1   function [y] = my_recursion(x)
2 -   if x > 5
3 -       y = x + my_recursion(x/2);
4 -   else
5 -       y = 2;
6 -   end
7 -   end
```

→ **The last recursive call is the first one to be fully resolved**

More examples of recursive functions

```
function [result] = my_power(x, n)

% Returns the value of x.^n, where x is an array of
% class double, and n is an integer of class double.
% n should be positive or zero.
%
% This implementation uses recursion.

% The base case is x.^0
if n == 0
    result = ones(size(x));
else
    % Here comes the recursive call...
    result = x .* my_power(x, n-1);
end

end
```

→ For a more complex example, see `my_give_change.m`

Another recursive algorithm: Sierpiński triangles

Algorithm (see `my_sierpinski_triangles.m`):

1. Start with an equilateral triangle that points upward
2. Divide the triangle into 4 equilateral triangles of equal surface area
   - Color the new triangle that points downward with a different color
   - Repeat step 2 on the three new triangles that point upward