Lecture 05

Functions

Erdal Yılmaz

Cornell University

July 8, 2013

Before we begin

- **HW2** Released Tomorrow
- **FL** Vote for Final

Outline

- Functions
  - M-files
  - Subfunctions
  - Anonymous functions
- Examples
  - Factorial Function
  - Approximating Sine function
  - Sieve of Eratosthenes

Functions

Syntax

```
function [y1,..,yN] = func_name(x1,..,xM)
% Help text written here and it will be
% shown until the first non-comment line

% Do stuff

end % optional
```

Factorial Function

factorial

$$n! = 1 * 2 * ... * n \qquad 0! = 1 \qquad 1! = 1$$

Code

```
function f = factorial (n)
% Computes n! = 1*2*...*n
   f = 1;
   for j = 1:n
      f = f * j;
   end
end
```

M-files

Function Files: define functions

```
function z = fname (x,y)
% This file has to be named fname.m
  z = x + y;
end
```

Script Files: collection of statements

```
% This file can have any valid filename
a = input('Enter x: ');
b = input('Enter y: ');
c = fname(a,b);
disp(c)
```

Variable Scope

Function Scope

```
function z = fname (x,y)
% This file has to be named fname.m
  z = x + y;
end
```

Global Scope

```
x = 5; disp(x);
a = input('Enter x: ');
b = input('Enter y: ');
c = fname(a,b);
disp(c); disp(x);
```

Subfunctions

Functions within functions

```
function y = myfunc(x)

 y =  sub1(sqrt(x)) + sub2(x)

  function t = sub1(x)
     t = log(x);
  end

  function r = sub2(p)
     r = 2*p;
  end

end
```

Anonymous Functions

not stored in a file

```
myfunc = @(x) (x^2);
y = myfunc(3);
```

Approximating Sine Function

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$
$$= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

What's wrong with the code?

```
function s = approx_sin (x, k)
n = 0; s = 0;
while n < (2k+1);
   s = s + (-1)^n + x^n  /factorial(n);
   n = n + 1;
end
```

Approximating Sine Function

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$
$$= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Correct Version

```
function s = approx_sin (x, k)
n = 0; s = 0;
while (2*n+1) < k
    s = s + (-1)^n*x^(2*n+1)/factorial(2*n+1);
    n = n + 1;
end
```

Primes Function

Question

What are all prime numbers $\leq N$?

Using what we know

```
function p = primes1 (N)
  p = [];  % creates an empty array
  for j = 1:N
     if isprime(j)  % built-in isprime
        p = [p, j]; % expands the array
     end
  end
end
```

A Better Primes Function

Add knowledge

All prime numbers, except 2, are odd numbers.

Updated code

```
function p = primes2 (N)
  if N>1, p = [2]; else p = []; end
  % check only odd numbers
  for j = 3:2:N
     if isprime(j)
        p = [p, j];
     end
  end
end
```

Measuring Performance

tic/toc

tic starts the timer, toc returns the elapsed time.

Comparing primes functions

```
N = input('Enter N: ');

tic                  % Start timer
p0 = primes(N);      % Call built-in primes
t0 = toc;            % Stop timer and
                     % store elapsed time

% Let's also measure our functions
tic; p1 = primes1(N); t1 = toc;
tic; p2 = primes2(N); t2 = toc;
```

Why is it slow?

- We check isprime for 3,5,7,9,11,13,15,..
- Why do we check 9? 15? 21? 25? ..

Current Version

```
function p = primes2 (N)
  if N>1, p = [2]; else p = []; end
  % there are unnecessary checks
  for j = 3:2:N
      if isprime(j)
        p = [p, j];
     end
  end
end
```

Sieve of Eratosthenes

Prime Sieve

- Idea: Eliminate the multiples of a number ahead of time, so that we don't need to check it.

Algorithm

```
% Create an array X of all 1's of length N
% Set X(1) to 0
% Find position k of next 1 in the X array
% If k is less than or equal to sqrt(N)
%    Set X(2*k), X(3*k), X(4*k) ... to zero
%    Go back to finding k
% Else
%    Find the indices of all 1's in X array
% These indices are prime numbers
```

Sieve of Eratosthenes

```
function p = primes3 (N)
   X = ones(1,N);       % An array of N 1's
   X(1) = 0;            % 1 is not a prime number
   m = floor(sqrt(N)); % The maximum number upto
                        % which we have to work
   k = 2; % The next available 1 in X array
          % if X(2) exists :)

   while k <= m

      % Set X(2*k) X(3*k) etc to zero
      for j = 2*k:k:N
         X(j) = 0;
      end

      % Find the next 1 in X array
      k = k + 1;
      while X(k) ~= 1
         k = k + 1;
      end

   end

   p = find(X == 1); % Find all indices of elements
                     % which are equal to 1 in X array
end
```