International University
School of Electrical Engineering

Introduction to Computers for Engineers

Dr. Hien Ta

Lecturely Topics

Lecture 1 - Basics – variables, arrays, matrices
Lecture 2 - Basics – matrices, operators, strings, cells
Lecture 3 - Functions & Plotting
Lecture 4 - User-defined Functions
Lecture 5 - Relational & logical operators, if, switch statements
Lecture 6 - For-loops, while-loops
Lecture 7 - Review on Midterm Exam
Lecture 8 - Solving Equations & Equation System (Matrix algebra)
Lecture 9 - Data Fitting & Integral Computation
Lecture 10 - Representing Signal and System
Lecture 11 - Random variables & Wireless System
Lecture 12 - Review on Final Exam

References: H. Moore, *MATLAB for Engineers*, 4/e, Prentice Hall, 2014
G. Recktenwald, *Numerical Methods with MATLAB*, Prentice Hall, 2000
A. Gilat, *MATLAB, An Introduction with Applications*, 4/e, Wiley, 2011

Topics

Program flow control with loops

`for` - loops

`while` - loops

`break, continue`

Examples: series calculations, square-root algorithm

Program Flow Control

Program flow is controlled by the following control structures:

1. for . . . end `% loops`
2. while . . . end
3. break, continue
4. if . . . end `% conditional`
5. if . . . else . . .end
6. if . . . elseif . . . else . . . end
7. switch . . . case . . . otherwise. . .end
8. return

for-loops and conditional ifs are by far the most commonly used control stuctures

for - loops

```
for variable = expression
    statements ...
end
```

row vector or matrix

```
for k = [1,2,3,4,5]
   x = 3.0 + 0.1*k
end

x =
    3.1000
x =
    3.2000
x =
    3.3000
x =
    3.4000
x =
    3.5000
```

```
for k = 1:5
   x = 3.0 + 0.1*k
end

x =
    3.1000
x =
    3.2000
x =
    3.3000
x =
    3.4000
x =
    3.5000
```

common types of for-loops

integer limits
a<=k<=b

```
for k = a:b
    ...
end
```

step increment

```
for k = a:s:b
    ...
end
```

val = any row vector

```
for k = val
    ...
end
```

val = can also be a matrix, see next page

```
for k = val
    ...
end
```

```
k1 = [1; 0; -2];
k2 = [0; 3; 1];

for k = k1
   x = 3.0 + 0.1*k
end

x =
    3.1000
    3.0000
    2.8000
```

```
k1 = [1; 0; -2];
k2 = [0; 3; 1];

for k = [k1,k2]
   x = 3.0 + 0.1*k
end

x =
    3.1000
    3.0000
    2.8000
x =
    3.0000
    3.3000
    3.1000
```

matrix

```
[k1,k2]

ans =
     1     0
     0     3
    -2     1
```

illustrating dynamic allocation & pre-allocation

```
clear x;
for k=[3,7,10]              % k runs successively through
   x(k) = 3 + 0.1*k;        % the values of [3,7,10]
   disp(x);                 % diplay current vector x
end
```

```
    0.0   0.0   3.3
    0.0   0.0   3.3   0.0   0.0   0.0   3.7
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   4.0
```

```
x = zeros(1,10);            % pre-allocate x to length 10
for k=[3,7,10]
   x(k) = 3 + 0.1*k;
   disp(x);
end
```

```
    0.0   0.0   3.3   0.0   0.0   0.0   0.0   0.0   0.0   0.0
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   0.0
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   4.0
```

`for-loops` can contain **`if`** statements

```
g = [92, 45, 90, 80, 94, 75];
count = 0;

for k = 1:length(g)
  if g(k) >= 90               % or, more simply, replace
    count = count + 1;        % if-end statements by
  end                         % count=count + (g(k)>=90);
end

disp(count)
     3
```

```
count = sum(g>=90);   % vectorized version

disp(count)
     3
```

computation of sums with for-loops, or while-loops

$$S = \sum_{k=1}^{N} \frac{1}{k^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{N^2}$$

```
N=1000; S=0;
for k=1:N,
    S = S + 1/k^2;      % update partial sums
end
```

```
>> S                    % tends to pi^2/6 as N -> inf
S =
    1.6439

k = 1:N; S = sum(1./k.^2)    % vectorized version
S =
    1.6439
```

if statements can contain **for-loops**

```
type = 'odd';
N = 1000; S = 0;

if strcmp(type,'even')
   for k=2:2:N                    % sum over even k's
      S = S + 1/k^2;
   end
elseif strcmp(type,'odd')
   for k=1:2:N                    % sum over odd k's
      S = S + 1/k^2;
   end
else
   disp('type must be ''odd'' or ''even''');
end
```

```
>> S                % odd case tends to pi^2/8
S =                 % even case tends to pi^2/24
    1.2332
```

nested `for-loops`

```
% double-loop example

N = 4; M = 3;

for i=1:N
   for j=1:M
      A(i,j) = i+j;
   end
end

>> A
A =
     2     3     4
     3     4     5
     4     5     6
     5     6     7
```

nested **for-loops**

```
% partially vectorized
% row-wise version

N=4; M=3; j=1:M;

for i=1:N
   A(i,:) = i+j;
end
```

```
% partially vectorized
% column-wise version

N=4; M=3; i=1:N;

for j=1:M
   A(:,j) = i+j;
end
```

```
% fully vectorized
% using meshgrid

N=4; M=3;
i=1:N; j=1:M;

[J,I]=meshgrid(j,i);
A = I+J;
```

why **(j,i)** instead of **(i,j)**?

while - loops

```
while condition
    statements ...
end
```

condition to continue the loop

```
N=1000; S=0; k=1;

while k<=N,
    S = S + 1/k^2;
    k = k+1;
end
```

```
>> S,k
S =
    1.6439
k =
    1001
```

carry out a few iterations by hand

$$S = 0, \quad k = 1$$

$$S = S + 1/k^2 = 0 + 1/1^2 = 1$$
$$k = k + 1 = 1 + 1 = 2$$

$$S = S + 1/k^2 = 1 + 1/2^2$$
$$k = k + 1 = 2 + 1 = 3$$

$$S = S + 1/k^2 = 1 + 1/2^2 + 1/3^2$$
$$k = k + 1 = 3 + 1 = 4$$

$$S = \sum_{k=1}^{N} \frac{1}{k^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{N^2}$$

as $N \to \infty$, sum converges to `π^2/6 = 1.6449...`

forever while - loops

```
while 1
    statements ...
    if condition
        break;
    end
    statements ...
end
```

condition to break out of the loop

```
N=1000; S=0; k=1;

while 1,
   S = S + 1/k^2;
   if k>N,
      break;
   end
   k = k+1;
end
```

```
>> S,k
S =            k =
    1.6439         1000
```

Note: the continuation condition of the conventional loop, and the break condition of the equivalent forever while loop are logical complements of each other

break
continue

`break`

terminates execution of a loop, and continues after the **`end`** of the loop

terminates out of a nested loop only

`continue`

stops present pass through a loop, but continues with next pass

Example: Series calculations

$$\pi = 2\sqrt{3}\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)3^k} = 2\sqrt{3}\lim_{n\to\infty}\sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)3^k}$$

$$S_n = \sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)3^k} = \sum_{k=0}^{n-1}\frac{(-1)^k}{(2k+1)3^k} + \frac{(-1)^n}{(2n+1)3^n}$$

$$S_n = S_{n-1} + \frac{(-1)^n}{(2n+1)3^n}, \quad n \geq 1, \quad S_0 = 1$$

$$S_n = S_{n-1} + \frac{(-1)^n}{(2n+1)3^n}, \quad n \geq 1, \quad S_0 = 1$$

$$\boxed{T_n = \frac{(-1)^n}{(2n+1)3^n}}$$

$$\boxed{S_n = S_{n-1} + T_n, \quad n \geq 1, \quad S_0 = 1}$$

Recursion can be implemented with a for-loop or a while-loop

$$\boxed{\text{relative error} = r = \frac{|S_n - S_{n-1}|}{|S_{n-1}|} = \frac{|T_n|}{|S_{n-1}|}}$$

```
N = 10000; S = 1;                      % initialize

tol = 1e-14;                           % relative error
                                       % try r = eps
for n=1:N,
   T = (-1)^n /(2*n+1)/3^n;            % n-th term
   if abs(T) < tol                     % break out of
      break;                           % the for-loop
   end                                 % if T is small
   S = S + T;                          % update sum
end

n, [pi; 2*sqrt(3)*S]                   % compare with pi

n =                                    % actual number
    26                                 % of iterations
ans =                                  % T = 7.4229e-015
   3.14159265358979
   3.14159265358977
```

for-loop

```
S = 0; T = 1; n = 0;

tol = 1e-14;

while abs(T) > tol
    S = S + T;
    n = n+1;
    T =  (-1)^n / (2*n+1) / 3^n;
end

n, [pi; 2*sqrt(3)*S]              % compare with pi

n =
    26
ans =                             % T = 7.4229e-015
   3.141592653589793
   3.141592653589774
```

while-loop

Example: Vectorized Taylor series calculations

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \lim_{n \to \infty} \sum_{k=0}^{n} \frac{x^k}{k!}$$

$$S_n = \sum_{k=0}^{n} \frac{x^k}{k!} = \sum_{k=0}^{n-1} \frac{x^k}{k!} + \frac{x^n}{n!}$$

$$T_n = \frac{x^n}{n!} = \frac{x\,x^{n-1}}{n\,(n-1)!} = \frac{x}{n} T_{n-1}, \quad n \geq 1$$

$$\boxed{\begin{array}{c} S_n = S_{n-1} + T_n, \quad n \geq 1 \\ \\ S_0 = 1, \quad T_0 = 1 \end{array}}$$

```
% version 1 - using a for-loop

x = [1 3 0 -4 10]';          % column vector

S = ones(size(x));           % inherits size of x
T = 1;
N = 10000;                   % max iterations
tol = 1e-12;                 % error tolerance

for n=1:N,
   T = T.*x/n;                  % n-th term
   if max(abs(T)) < tol         % break if |T|<tol
      break;                    % why max(abs(T))?
   end
   S = S + T;                   % update sum
end
```

$S = 1\,, \quad T = 1\,, \quad \text{(initialize)}$

always carry out some iterations by hand

$n = 1$

$T = T \cdot x/n = 1 \cdot x/1 = x$

$S = S + T = 1 + x$

$n = 2$

$T = T \cdot x/n = x \cdot x/2 = x^2/2 = x^2/2!$

$S = S + T = (1 + x) + x^2/2! = 1 + x + x^2/2!$

$n = 3$

$T = T \cdot x/n = (x^2/2!) \cdot x/3 = x^2/(2 \cdot 3) = x^3/3!$

$S = S + T = (1 + x + x^2/2!) + x^3/3! = 1 + x + x^2/2! + x^3/3!$

```
fprintf('     x         exp(x)           S\n');
fprintf('------------------------------------\n');
fprintf('% 7.2f  %12.6f  %12.6f\n', [x,exp(x),S]');
fprintf('------------------------------------\n');
fprintf(['iterations n = ',int2str(n),'\n']);
```

```
     x         exp(x)           S
------------------------------------
   1.00      2.718282      2.718282
   3.00     20.085537     20.085537
   0.00      1.000000      1.000000
  -4.00      0.018316      0.018316
  10.00  22026.465795  22026.465795
------------------------------------
iterations n = 47
```

```
% norm(S-exp(x))  % equals 7.2760e-012
```

```
% version 2 - using a while-loop

x = [1 3 0 -4 10]';          % column vector

S = ones(size(x));           % inherits size of x
T = 1; n=1;                  % initialize
tol = 1e-12;                 % error tolerance

while max(abs(T)) > tol        % can also use
   T = x.*T/n;                 % norm(T)>tol
   S = S+T;
   n = n+1;
end
```

$$S = 1\,, \quad T = 1\,, \quad n = 1\,, \quad \text{(initialize)}$$

$$T = T \cdot x/n = 1 \cdot x/1 = x$$

$$S = S + T = 1 + x$$

$$n = 2$$

$$T = T \cdot x/n = x \cdot x/2 = x^2/2$$

$$S = S + T = 1 + x + x^2/2!$$

$$n = 3$$

$$T = T \cdot x/n = (x^2/2) \cdot x/3 = x^3/3!$$

$$S = S + T = 1 + x + x^2/2! + x^3/3!$$

$$n = 4$$

---

always carry out some iterations by hand

```
fprintf('      x           exp(x)             S\n');
fprintf('-----------------------------------\n');
fprintf('% 7.2f  %12.6f  %12.6f\n', [x,exp(x),S]');
fprintf('-----------------------------------\n');
fprintf(['iterations n = ',int2str(n-1),'\n']);
```

why **n-1**?

```
     x          exp(x)              S
-----------------------------------
   1.00      2.718282      2.718282
   3.00     20.085537     20.085537
   0.00      1.000000      1.000000
  -4.00      0.018316      0.018316
  10.00  22026.465795  22026.465795
-----------------------------------
iterations n = 47
```

```
% version 3 - using a forever while-loop

x = [1 3 0 -4 10]';        % column vector

S = ones(size(x));         % inherits size of x
T = 1; n=1;                % initialize
tol = 1e-12;               % error tolerance

while 1                    % forever loop
   T = x.*T/n;
   if max(abs(T)) < tol
      break;
   end
   S = S+T;
   n = n+1;
end
```

```
fprintf('     x           exp(x)                 S\n');
fprintf('------------------------------------\n');
fprintf('% 7.2f  %12.6f  %12.6f\n', [x,exp(x),S]');
fprintf('------------------------------------\n');
fprintf(['iterations n = ',int2str(n),'\n']);
```

```
     x           exp(x)              S
------------------------------------
   1.00       2.718282       2.718282
   3.00      20.085537      20.085537
   0.00       1.000000       1.000000
  -4.00       0.018316       0.018316
  10.00   22026.465795   22026.465795
------------------------------------
iterations n = 47
```

Example: Square-root algorithm

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \quad n = 0, 1, 2, \ldots$$

$$x_n \to \sqrt{a}$$

```
a = 20;            % sqrt(a) = 4.47213595499958O
N = 10;
x(1) = 8;          % arbitrary initial value

for n=1:N-1,
   x(n+1) = (x(n) + a/x(n))/2;
end
```

```
fprintf(' n                    x               \n');
fprintf('-----------------------------------\n');
fprintf('%3.0f   %17.15f\n', [1:N; x]);
```

```
 n           x
----------------------
 1   8.000000000000000
 2   5.250000000000000
 3   4.529761904761905
 4   4.472502502972279
 5   4.472135970019965
 6   4.472135954999580
 7   4.472135954999580
 8   4.472135954999580
 9   4.472135954999580
10   4.472135954999580
```

converged in 6 iterations

```
a = 20; N = 10; x(1) = 8;       % initialize
tol = 1e-12;                    % our choice

fprintf(' n          x(n)        \n');
fprintf('----------------------\n');

for n=1:N-1,
   fprintf('%2.0f   %17.15f\n', n,x(n));
   if abs(x(n)^2-a) <= tol
      break;
   end
   x(n+1) = (x(n) + a/x(n))/2;
end
```

break out of the loop if converged to within the error tolerance, **`tol`**

(output in next page)

```
 n              x(n)
---------------------------
 1    8.00000000000000
 2    5.25000000000000
 3    4.52976190476190
 4    4.47250250297228
 5    4.47213597001997
 6    4.47213595499958
```

converged in 6 iterations

exactly the same output is obtained using a while-loop in the next page

```
a = 20; n = 1; x = 8; tol = 1e-12;

fprintf(' n                x                error\n');
fprintf('---------------------------------\n');

while abs(x^2-a) > tol
   x = (x + a/x)/2;
   n = n+1;
   E = abs(x^2-a);
  fprintf(' %1d   %17.15f   %8.2e\n', n,x,E);
end
```

conventional while-loop

```
 n          x             error
--------------------------------
 2   5.250000000000000   7.56e+00
 3   4.529761904761905   5.19e-01
 4   4.472502502972279   3.28e-03
 5   4.472135970019965   1.34e-07
 6   4.472135954999580   3.55e-15
```

final error
$E = |x^2 - a|$
is smaller than
tol

```
a = 20; x = 8; n = 1; tol = 1e-12;

fprintf(' n                x                  error\n');
fprintf('--------------------------------------\n');

while 1
   if abs(x^2-a) <= tol, break; end
   x = (x + a/x)/2;
   n = n+1;
   E = abs(x^2-a);
   fprintf(' %1d   %17.15f   %8.2e\n', n,x,E);
end
```

forever while-loop

```
 n           x                 error
------------------------------------
 2   5.250000000000000   7.56e+00
 3   4.529761904761905   5.19e-01
 4   4.472502502972279   3.28e-03
 5   4.472135970019965   1.34e-07
 6   4.472135954999580   3.55e-15
```

Example: Calculating Products

$$\frac{\sin x}{x} = \prod_{k=1}^{\infty} \cos\left(\frac{x}{2^k}\right) = \cos\left(\frac{x}{2^1}\right) \cos\left(\frac{x}{2^2}\right) \cos\left(\frac{x}{2^3}\right) \cdots$$

$$\boxed{S_k = S_{k-1} \cdot \cos\left(\frac{x}{2^k}\right), \quad k = 1, 2, 3, \ldots, \quad S_0 = 1}$$

$$\text{relative error} = r = \frac{|S_k - S_{k-1}|}{|S_{k-1}|}$$

```
x = [0.1, 0.2, 1, 4, 8]';        % column vector

S = ones(size(x));               % inherits size of x
k=1;

r = 1e-10;                       % relative error

while 1                          % forever loop
  F = cos(x/2^k);                % k-th factor
  S1 = S.*F;
  if norm(S1-S) < r*norm(S)
    break;
  end
  S = S1;
  k = k+1;
end
```

use vector norm to measure the distance between **S** and **S1**

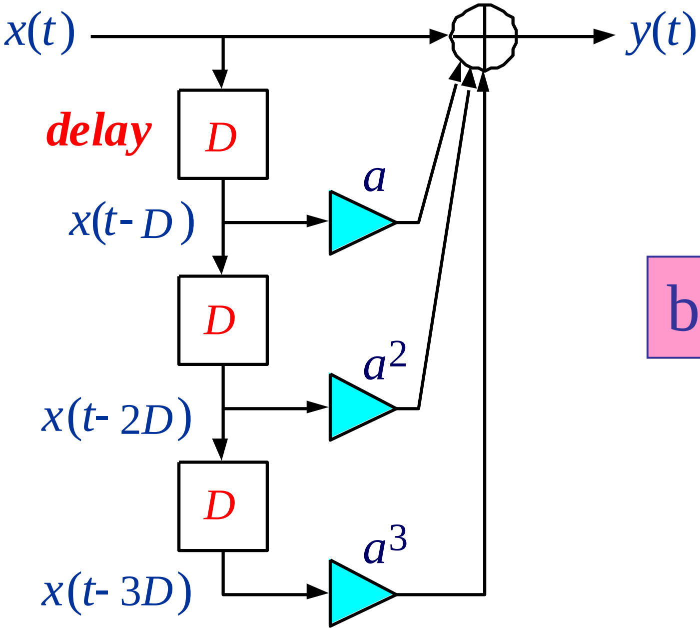
```
fprintf('   x         sin(x)/x          S\n');
fprintf('----------------------------------\n');
fprintf('%6.2f  %12.6f  %12.6f\n', [x, sin(x)./x,S]');
fprintf('----------------------------------\n');
fprintf(['iterations k = ',int2str(k),'\n']);
```

```
    x         sin(x)/x          S
----------------------------------
  0.10       0.998334      0.998334
  0.20       0.993347      0.993347
  1.00       0.841471      0.841471
  4.00      -0.189201     -0.189201
  8.00       0.123670      0.123670
----------------------------------
iterations k = 18
```

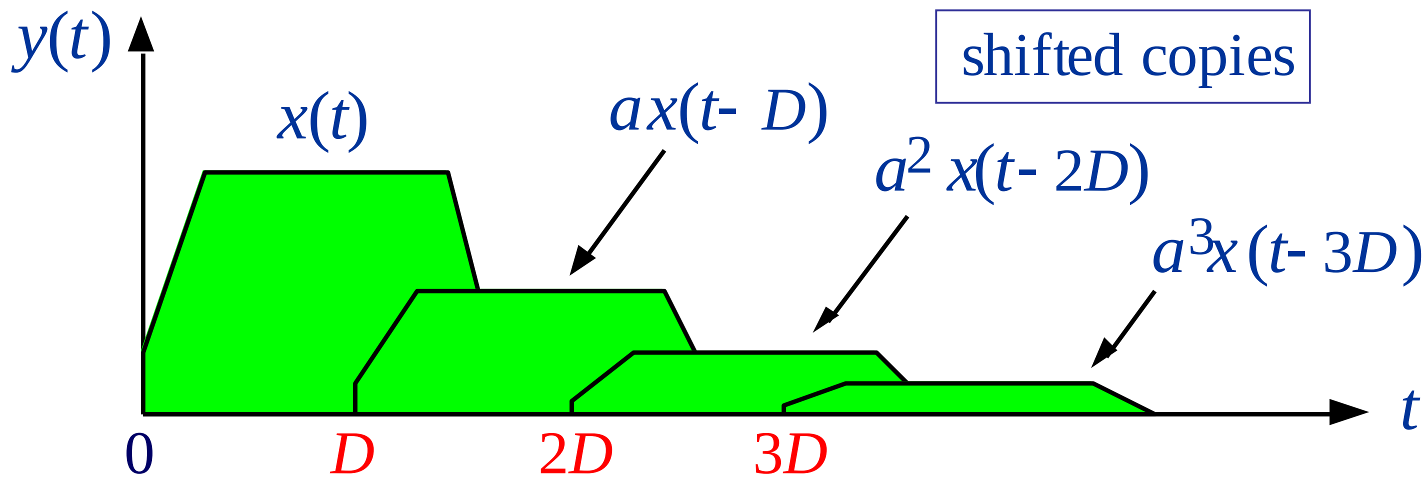
Example: Overlapping Echoes

- a simple example of a Digital Audio Effect
- reads a wave file and plays a 20-sec portion of it
- then, adds three overlapping copies of itself and plays the result
- illustrates the use of for-loops, if-statements, and pre-allocation to speed up processing

complete program, **`echoes.m`**, and supporting wave files are in the zip file, **`echoes.zip`**, (under week-2 and week-7 resources on sakai)



block-diagram realization



$$y(t) = x(t) + ax(t-D) + a^2x(t-2D) + a^3x(t-3D)$$

```
% echoes.m - listening to overlapping echoes

clear all;

[x,Fs] = wavread('dsummer.wav');   % read wave file and Fs

N = min(round(20*Fs),length(x));   % play no more than 20 sec
x = x(1:N);                        % truncate x to length N

sound(x,Fs);                       % play x

T = 1/2; D = round(T*Fs);    % echo delay in sec and in samples

Fs, N, D                  % here, Fs=44100, N=839242, D=22050

a = 0.5;                  % multiplier coefficient

y = zeros(size(x));       % pre-allocation speeds up processing
```

```
tic                        % tic-toc - execution time

for n=1:length(x),         % construct overlapped signal y
   if n<=D,
      y(n) = x(n);
   elseif n<=2*D,
      y(n) = x(n) + a * x(n-D);
   elseif n<=3*D,
      y(n) = x(n) + a * x(n-D) + a^2 * x(n-2*D);
   else,
      y(n) = x(n) + a * x(n-D) + a^2 * x(n-2*D) +...
             a^3 * x(n-3*D);
   end
end

toc

pause; sound(y,Fs);            % play overlapped signal y
```

if-elseif statements within a for-loop

pre-allocation results

wave file	Fs	N	with	without
JB.wav	16000	71472	0.02 sec	0.06 sec
nodelay.wav	22050	266758	0.10 sec	0.30 sec
dsummer.wav	44100	839242	0.30 sec	0.90 sec