Rutgers University
School of Engineering

Fall 2011

14:440:127 - Introduction to Computers for Engineers

Sophocles J. Orfanidis
ECE Department
orfanidi@ece.rutgers.edu

week 7

Weekly Topics

Week 1 - Basics – variables, arrays, matrices, plotting (ch. 2 & 3)
Week 2 - Basics – operators, functions, program flow (ch. 2 & 3)
Week 3 - Matrices (ch. 4)
Week 4 - Plotting – 2D and 3D plots (ch. 5)
Week 5 - User-defined functions (ch. 6)
Week 6 - Input-output processing (ch. 7)
Week 7 - Program flow control & relational operators (ch. 8)
Week 8 - Matrix algebra – solving linear equations (ch. 9)
Week 9 - Structures & cell arrays (ch. 10)
Week 10 - Symbolic math (ch. 11)
Week 11 - Numerical methods – data fitting (ch. 12)
Week 12 – Selected topics

Textbook: H. Moore, *MATLAB for Engineers*, 2nd ed., Prentice Hall, 2009

Topics

Relational and logical operators
Logical indexing
`find` function
Program flow control
`for` - loops
`while` - loops
`if` – statements
`switch` – statements
`break, continue`
Examples: series calculations, square-root algorithm, piece-wise functions, unit-step function, indicator functions, sinc function, echoes

Relational and Logical Operators

Relational and logical functions

```
find, logical, true, false, any, all

ischar, isequal, isfinite, isinf, isinteger
islogical, isnan, isreal
```

```
>> doc is*            % list of all 'is' functions
>> help logical       % convert to logical
>> help true          % logical 1
>> help false         % logical 0
>> help relop         % relational operators
>> help ops           % same as help /
>> help find          % indices of non-zero elements
```

```
>> help precedence
```

Relational Operators

```
==   equal
~=   not equal
 <   less than
 >   greater than
<=   less than or equal
>=   greater than or equal
```

>> help relop

Logical Operators

```
&    logical AND,     e.g., A&B, A,B=expressions
&&   logical AND for scalars w/ short-circuiting
|    logical OR,      e.g., A|B, or A||B
||   logical OR for scalars w/ short-circuiting
~    logical NOT,     e.g., ~A
xor  exclusive OR,    e.g., xor(A,B)
any  true if any elements are non-zero
all  true if all elements are non-zero
```

```
>> a = [1 2 0 -3 7];
>> b = [3 2 4 -1 7];
>> a == b
ans =
     0    1    0    0    1
>> a == -3
ans =
     0    0    0    1    0
>> find(a==-3)             % otherwise, empty
ans =
     4
>> find(a), find(a>=2), find(a<=0)
ans =
     1    2    4    5
ans =
     2    5
ans =
     3    4
```

```
>> a>=2
ans =
     0   1   0   0   1
```

```
>> a = [1 2 0 -3 7];
>> b = [3 2 4 -1 7];

>> a < b
ans =
     1    0    1    1    0

>> a>=2, b<=2
ans =
     0    1    0    0    1
ans =
     0    1    0    1    0

>> (a>=2) & (b<=2)                % logical AND
ans =
     0    1    0    0    0

>> (a>=2) | (b<=2)                % logical OR
ans =
     0    1    0    1    1
```

```
>> a = [1 3 4 -3 7];
>> k = (a>=2), m = find(a>=2)
k =
     0     1     1     0     1
m =
     2     3     5

>> a(m), a(k)
ans =
     3     4     7
ans =
     3     4     7

>> i = [0 1 1 0 1]
>> a(i)
??? Subscript indices must either be real
positive integers or logicals.

% but note, a(logical(i)) works
```

logical indexing

`class(k)` is logical

logical indexing

`a(a>=2)`

`class(i)` is double, but `i==k` is `true`

```
>> A = [3 4 nan; -5 inf 2]
A =
     3     4   NaN
    -5   Inf     2

>> k = isfinite(A)
k =
     1     1     0
     1     0     1

>> A(k)        % listed column-wise
ans =
     3
    -5
     4
     2

>> A(~k)=0     % set non-finite entries to zero
A =
     3     4     0
    -5     0     2
```

more on
logical indexing

```
>> find(k)
ans =
     1
     2
     3
     6
```

```
>> [i,j] = find(k)
```

```
A = [9  9  2        B = [7  1  7
     2  5  4             3  4  8
     9  8  9];           9  4  2];
```

```
>> A<B
ans =
     0  0  1
     1  0  1
     0  0  0
```

```
>> find(A<B)
ans =
     2
     7
     8
```

```
[i,j]=find(A<B)
i =    j =
   2      1
   1      3
   2      3
```

```
>> A==9
ans =
     1  1  0
     0  0  0
     1  0  1
```

```
>> find(A==9)
ans =
     1
     3
     4
     9
```

```
>> A(A==9)=-9
A =
    -9  -9   2
     2   5   4
    -9   8  -9
```

any
all

```
A = [9  9  2
     2  5  4
     9  8  9];
```

```
B = [7  1  7
     3  4  8
     9  4  2];
```

```
any(A==2)
ans =
     1   0   1

any(A==2,2)
ans =
     1
     1
     0
```

```
all(A>B)
ans =
     0   1   0

all(A>B,2)
ans =
     0
     0
     0
```

```
A==B
ans =
     0   0   0
     0   0   0
     1   0   0

any(A==B)
ans =
     1   0   0

any(any(A==B))
ans =
     1
```

any,all operate column-wise, or, row-wise with extra argument

```
all(all(A==B));
```

```
>> A = [36 -4 9; 16 9 -25], B=A;

A =
    36    -4     9
    16     9   -25


>> k = (B>=0)

k =
     1     0     1
     1     1     0


>> B(k) = sqrt(B(k));
>> B(~k) = -sqrt(-B(~k))

B =
     6    -2     3
     4     3    -5
```

Example:
take square-roots of the absolute values, but preserve the signs

Program Flow Control

Program flow is controlled by the following control structures:

1. for . . . end **% loops**
2. while . . . end

3. if . . . end **% conditional**
4. if . . . else . . .end
5. if . . . elseif . . . else . . . end
6. switch . . . case . . . otherwise. . .end
7. break, continue

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

for - loops

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

```
>> N=1000; S=0;
>> for n=1:N,
     S = S + 1/n^2;      % compute sum:
   end

>> S
S =
    1.6439

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

$$S = \sum_{n=1}^{N} \frac{1}{n^2}$$

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

while - loops

```
>> N=1000; S=0; n=1;
>> while n<=N,
      S = S + 1/n^2;     % compute sum:
      n = n+1;
   end

>> S
S =
    1.6439

>> pi^2/6          % note the limiting sum,
ans =              % first derived by Euler
    1.6449
```

$$S = \sum_{n=1}^{N} \frac{1}{n^2}$$

$$\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$$

if - statements

```
if condition
    statements ...
end
```

```
if condition
    statements ...
else
    statements ...
end
```

```
if condition
    statements ...
elseif condition
    statements ...
elseif condition
    statements ...
else
    statements ...
end
```

several **`elseif`** statements
may be present,

`elseif` does not need a matching **`end`**

```
>> x = 1;
>> % x = 0/0
>> % x = 1/0

>> if isinf(x),
      disp('x is infinite');
   elseif isnan(x),
      disp('x is not-a-number');
   else
      disp('x is finite number');
   end

x is finite number
% x is not-a-number
% x is infinite
```

```
switch expression
   case expression
      statements ...
   case expression
      statements ...
   otherwise
      statements ...
end
```

this expression is evaluated first, and if its value matches any of these, then the corresponding case-statements are executed

several **case** statements may be present

equivalent calculation using the built-in function **norm** :

```
x = [1, -4, 5, 3]; p = inf;
switch p
   case 1
      N = sum(abs(x));               % N = norm(x,1);
   case 2
      N = sqrt(sum(abs(x).^2));      % N = norm(x,2);
   case inf
      N = max(abs(x));               % N = norm(x,inf);
   otherwise
      N = sqrt(sum(abs(x).^2));      % N = norm(x,2);
end
```

L_1, L_2, and L_∞ norms of a vector

$$\mathbf{x} = [x_1, x_2, \ldots, x_N]$$

$$\|\mathbf{x}\|_1 = \sum_{n=1}^{N} |x_n|$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{n=1}^{N} |x_n|^2}$$

discussed further in week 8

$$\|\mathbf{x}\|_1 = \max\left(|x_1|, |x_2|, \ldots, |x_N|\right)$$

```
>> help norm        % vector and matrix norms
```

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 1: Series calculations

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

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

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

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

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

for n=1:N,
   T = (-1)^n /(2*n+1)/3^n;            % n-th term
   if abs(T) < eps                     % 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
    30                                 % of iterations
ans =
   3.141592653589793
   3.141592653589794
```

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

while abs(T) > eps
   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 =
    30
ans =
   3.141592653589793
   3.141592653589794
```

Example 2: Vectorized Taylor series calculations

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

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

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

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

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

S = ones(size(x));           % inherits size of x
T = 1;
N = 10000;                   % max iterations

for n=1:N,
   T = T.*x/n;                  % n-th term
   if max(abs(T)) < eps         % break if T<eps
      break;                    % why max(abs(T))?
   end
   S = S + T;                   % update sum
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 = 52
```

Example 3: 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
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)<=eps(a), break; end
   x(n+1) = (x(n) + a/x(n))/2;
end
```

break out of the loop if converged within the floating point limits

```
 n          x(n)
----------------------
 1   8.000000000000000
 2   5.250000000000000
 3   4.529761904761905
 4   4.472502502972279
 5   4.472135970019965
 6   4.472135954999580
```

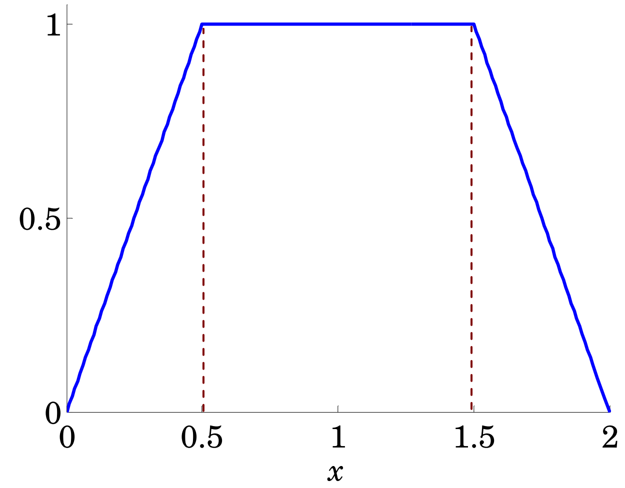
converged in 6 iterations

```
a = 20; x = 8; n = 1; X = [n, x];

while abs(x^2-a)>eps(a)    % note eps(a)
   x = (x + a/x)/2;
   n = n+1; X = [X; n, x];
end

fprintf(' n           x          \n');
fprintf('----------------------\n');
fprintf('%2.0f   %17.15f\n', X');
```

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

Example 4: Defining piece-wise functions

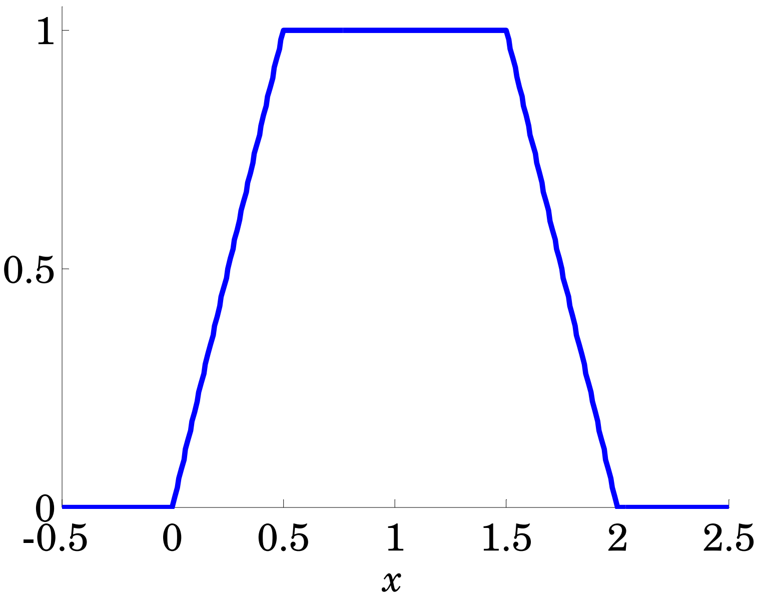
$$f(x) = \begin{cases} 2x, & 0 \le x \le 0.5 \\ 1, & 0.5 \le x \le 1.5 \\ 4 - 2x, & 1.5 \le x \le 2 \end{cases}$$



$$v(x, a, b) = \begin{cases} 1, & a \le x < b \\ 0, & \text{otherwise} \end{cases} = \text{(indicator function)}$$

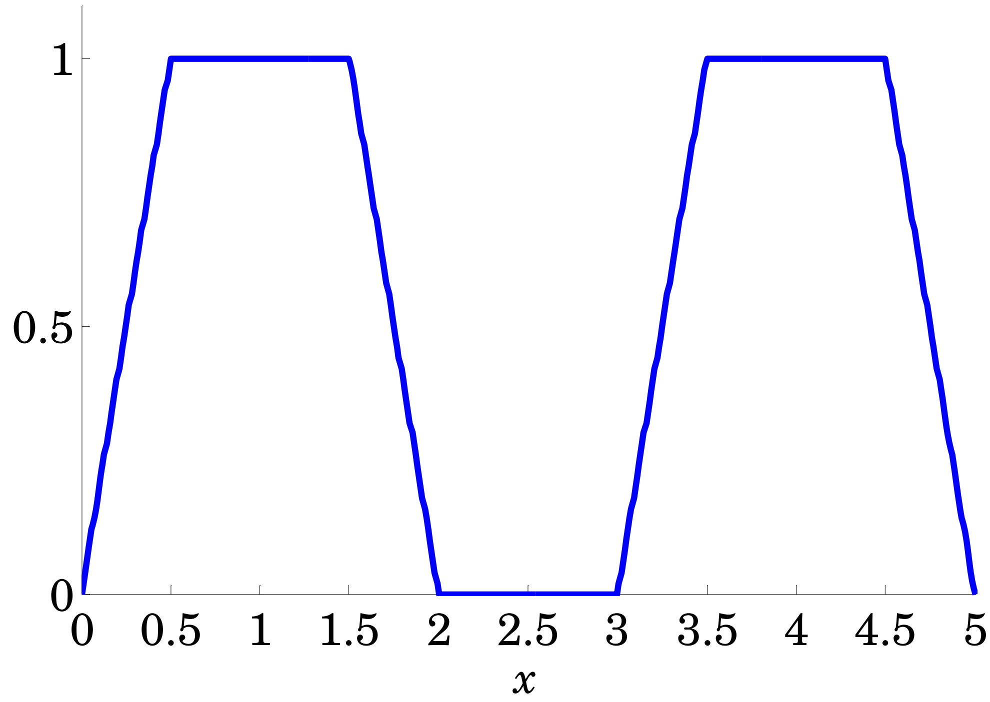
$$f(x) = 2x\,v(x,\ 0,\ 0.5) + v(x,\ 0.5,\ 1.5) + (4 - 2x)\,v(x,\ 1.5,\ 2)$$

```
v = @(x,a,b) ((x>=a) & (x<b));

f = @(x) 2*x.*v(x,0,0.5) + v(x,0.5,1.5) + ...
         (4-2*x).*v(x,1.5,2);

x = linspace(-0.5,2.5,301);

figure; plot(x,f(x), 'b-');
```



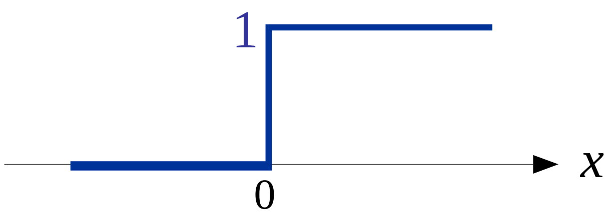
```
v = @(x,a,b) ((x>=a) & (x<b));

f = @(x) 2*x.*v(x,0,0.5) + v(x,0.5,1.5) + ...
         (4-2*x).*v(x,1.5,2);

x = linspace(0,5,501);

figure; plot(x,f(x)+f(x-3), 'b-');
```

replicating f(x)



unit-step function

$$u(x) = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

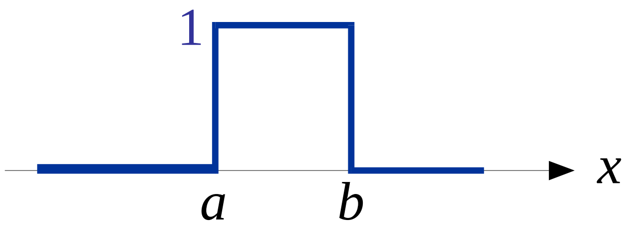



```
u = @(x) (x>=0);      % unit-step function
```

```
e.g., x =-3,-2,-1, 0, 1, 2, 3
     u(x)= 0, 0, 0, 1, 1, 1, 1
```

indicator function

$$v(x,a,b) = u(x-a) - u(x-b)$$




```
v = @(x,a,b) u(x-a)–u(x-b);   % indicator
```

Example 5: Evaluating the sinc function

```
function y = my_sinc(x)

warning off;

y = sin(pi*x)./(pi*x);

y(isinf(x)) = 0;

y(x==0) = 1;
```

generates **NaN**s for **x=inf** and **x=0**

fix **NaN** when **x=inf**

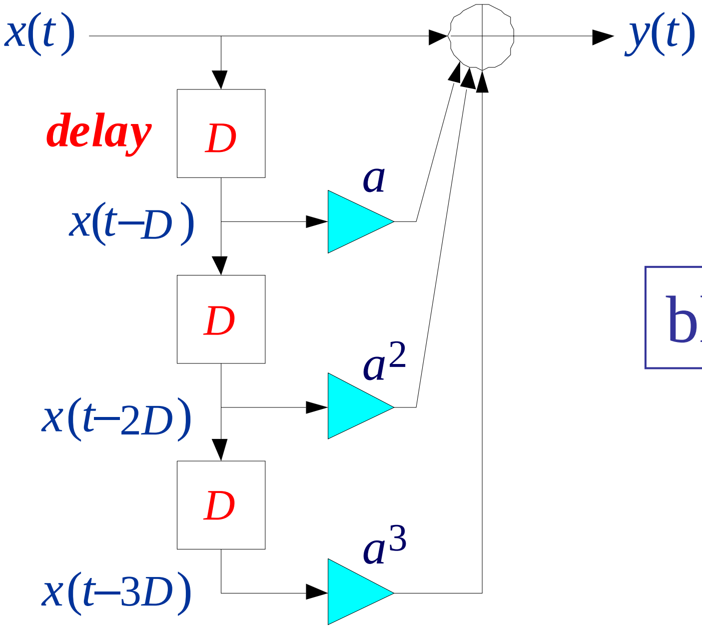
fix **NaN** when **x=0**

```
x = [0 0 inf 0 nan];
y = sin(pi*x)./(pi*x)
y =
   NaN   NaN   NaN   NaN   NaN

isinf(x)
ans =
     0     0     1     0     0
y(isinf(x)) = 0
y =
   NaN   NaN     0   NaN   NaN

x==0
ans =
     1     1     0     1     0
y(x==0) = 1
y =
     1     1     0     1   NaN
```

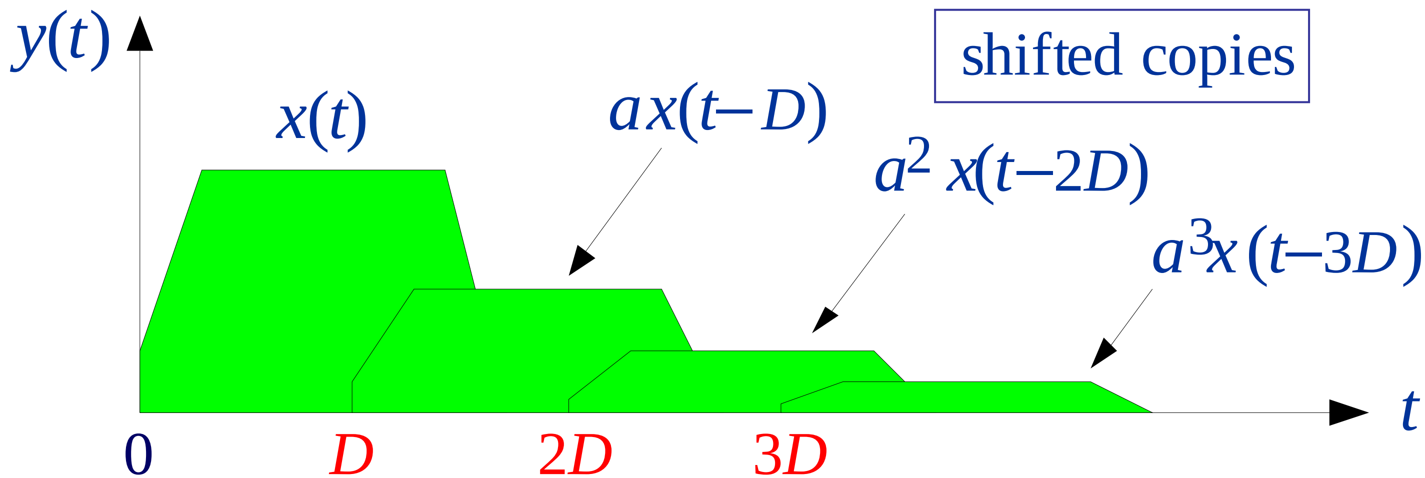
Example 5: Overlapping Echoes

- 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 resources on sakai)



block-diagram realization



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

```
% echoes.m - listen 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),          % 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 y
```

pre-allocation results

wave file	Fs	N	with	without
JB.wav	16000	71472	0.02 sec	34.44 sec
nodelay.wav	22050	266758	0.13 sec	702.33 sec
dsummer.wav	44100	839242	0.39 sec	too long