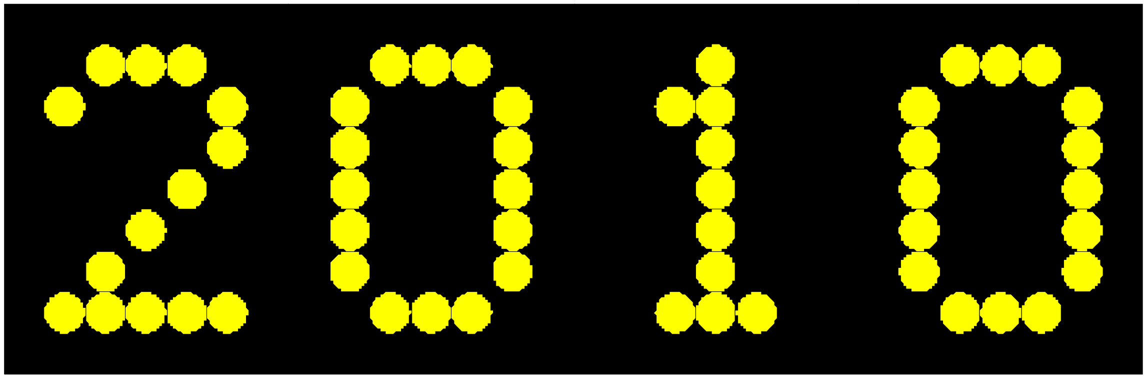
- Previous Lecture:
  - Review matrix, cell array, structure array

- Today's Lecture:
  - Working with sound files
  - Review vector, graphics, struct array, cell array

- Announcements:
  - P5 due Friday at 11pm
  - Prelim 3 Tuesday 7:30pm
    - Sound (today's topic) will NOT be on prelim 3
    - Review session Sunday 1:30-3pm, location TBA

Digital display of a whole number

- Example: `showNumber(2010)`



- Need to convert the number to a vector of digits
  - `2010` → `[2 0 1 0]`
- Then display the digits in the vector side-by-side

```
function showNumber(n)
% Digital display of integer n, n>0

hold on; axis equal off

% Convert n to a vector of digits







% Display the digits in v
D = TheDigits();  % D{k} is matrix encoding digit k
```

```
function showNumber(n)
% Digital display of integer n, n>0

hold on; axis equal off

% Convert n to a vector of digits







% Display the digits in v
D = TheDigits();  % D{k} is matrix encoding digit k
for k=1:length(v)
    index= v(k);
    if index==0
        index= 10;
    end
    drawDigit(k,1,1,D{index})
end
```

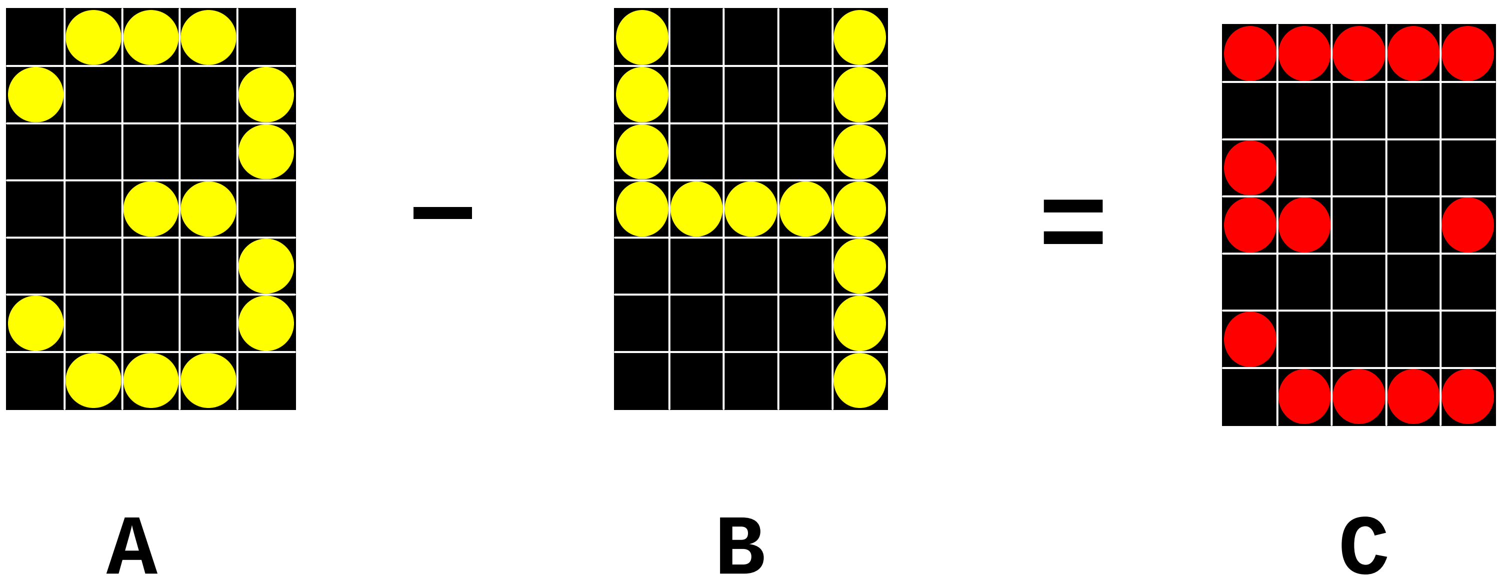
```
function showNumber(n)
% Digital display of integer n, n>0

hold on; axis equal off

% Convert n to a vector of digits
v= [];
while n>0
    v= [rem(n,10) v];
    n= floor(n/10);
end


% Display the digits in v
D = TheDigits();  % D{k} is matrix encoding digit k
for k=1:length(v)
    index= v(k);
    if index==0
        index= 10;
    end
    drawDigit(k,1,1,D{index})
end
```

How to calculate the difference between 2 bitmaps?




```
C(i,j) = abs( A(i,j) - B(i,j) )
```

```
% A and B have same size
[nr,nc]= size(A);
B= zeros(nr,nc);
for r= 1:nr
   for c= 1:nc
      C(r,c)= abs(A(r,c)-B(r,c));
   end
end
```

```
% A and B have same size
C= abs(A-B);
```

C is a 0-1 matrix where **1** indicates that **A(i,j)** and **B(i,j)** are *different*.

Reading and playing .wav files

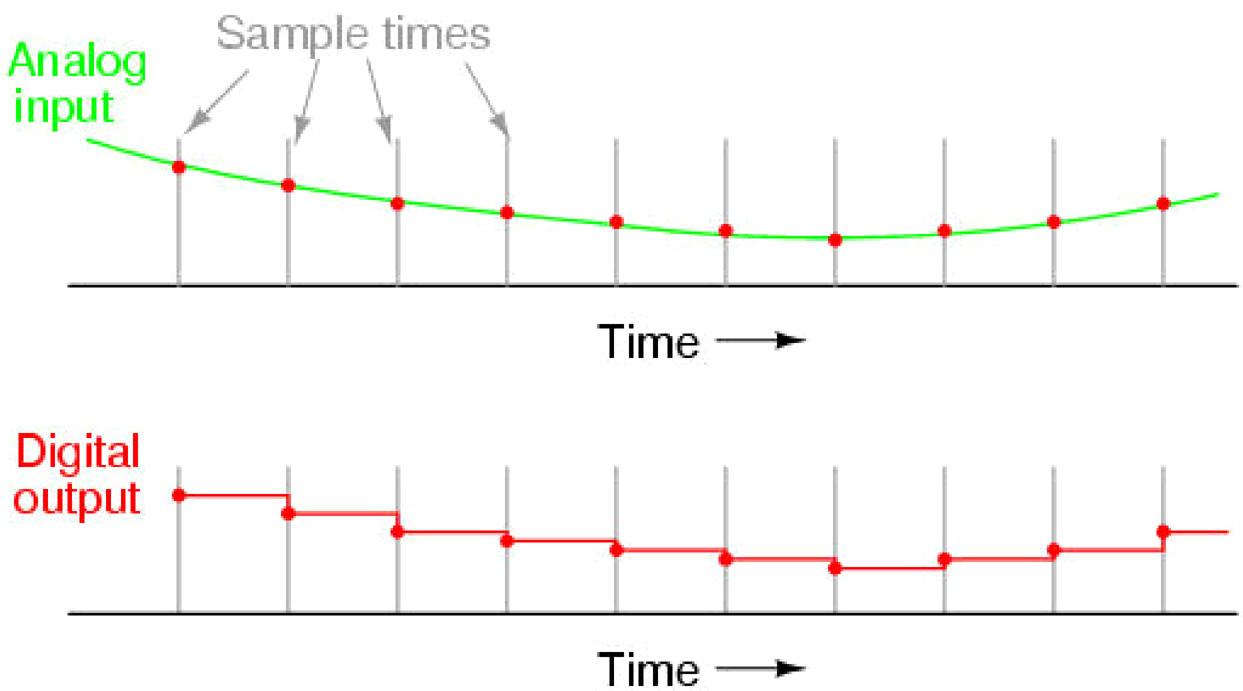
```
[y,rate,nBits] = wavread('austin.wav')
sound(y,rate)
```

A wav file is for the computer to process—software is required to play the sound.

Computing with sound in Matlab requires that we first convert the wav format data into simple numeric data—the job of `wavread`.

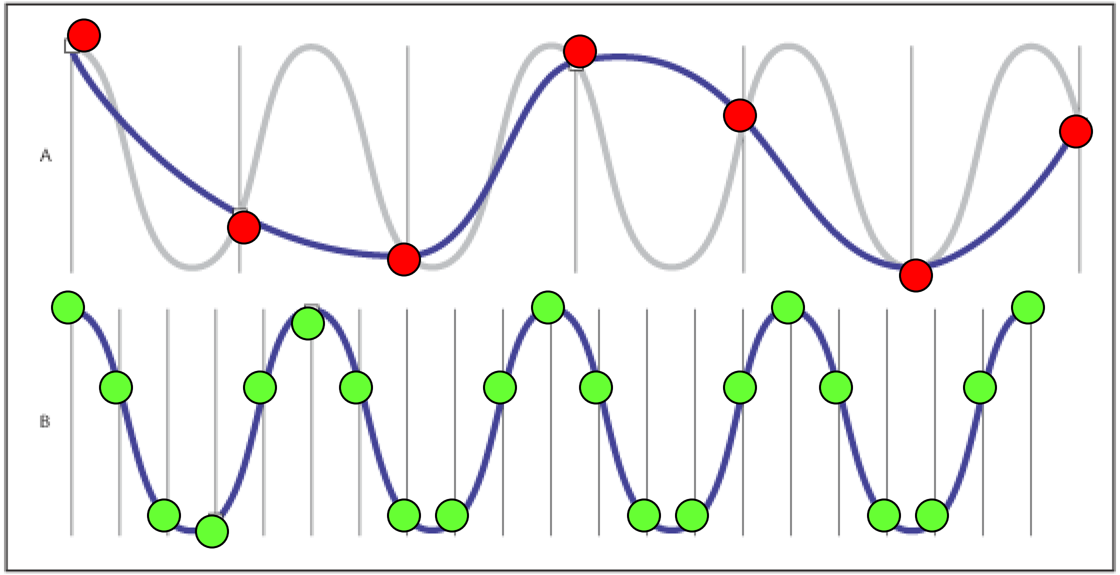
Computing with sound requires digitization

- Sound is continuous; capture its essence by sampling
- Digitized sound is a vector of numbers



Sampling rate affects the quality

If sampling not frequent enough, then the discretized sound will not capture the essence of the continuous sound…



Too slow

OK

Sampling Rate

Given human perception, 20000 samples/second is pretty good (20000Hz or 20kHz)

8,000 Hz	required for speech over the telephone
44,100 Hz	required for audio CD
192,400 Hz	required for HD-DVD audio tracks

Resolution also affects the quality

Typically, each sampled value is encoded as an 8-bit integer in the .wav file.

Possible values: -128, -127,…,-1,0,1,…,127

Loud: -120, 90, 122, etc.

Quiet: 3, 10, -5

Magnitude
determines loudness

16-bit used when very high quality is required.

`wavread` converts the 8-bit values to floating point values between -1 and 1

```
[y,rate,nBits]= wavread('austin.wav')
```

```
 0.4609
 0.3516
 0.2734
 0.2891
 0.2500
 0.1484
 0.1094
 0.1641
 0.1484
 0.0000
-0.1641
-0.2734
-0.3281
```

`y(50000:50012)`

wavread

Name of the source file

```
[data,rate,nBits]= wavread('austin.wav')
```

The vector of sampled sound values is assigned to this variable

The sampling rate is assigned to this variable

The resolution is assigned to this variable

wavread

```
[y,rate,nBits]= wavread('austin.wav');
n = length(y);

n =
       54453
rate =
       11025
nBits =
         8
```

austin.wav

encoded the sound with 54,453 8-bit numbers that were taken at 11025 samples per second

What is the play duration?

A: `rate*n`

B: `rate/n`

C: `n/rate`

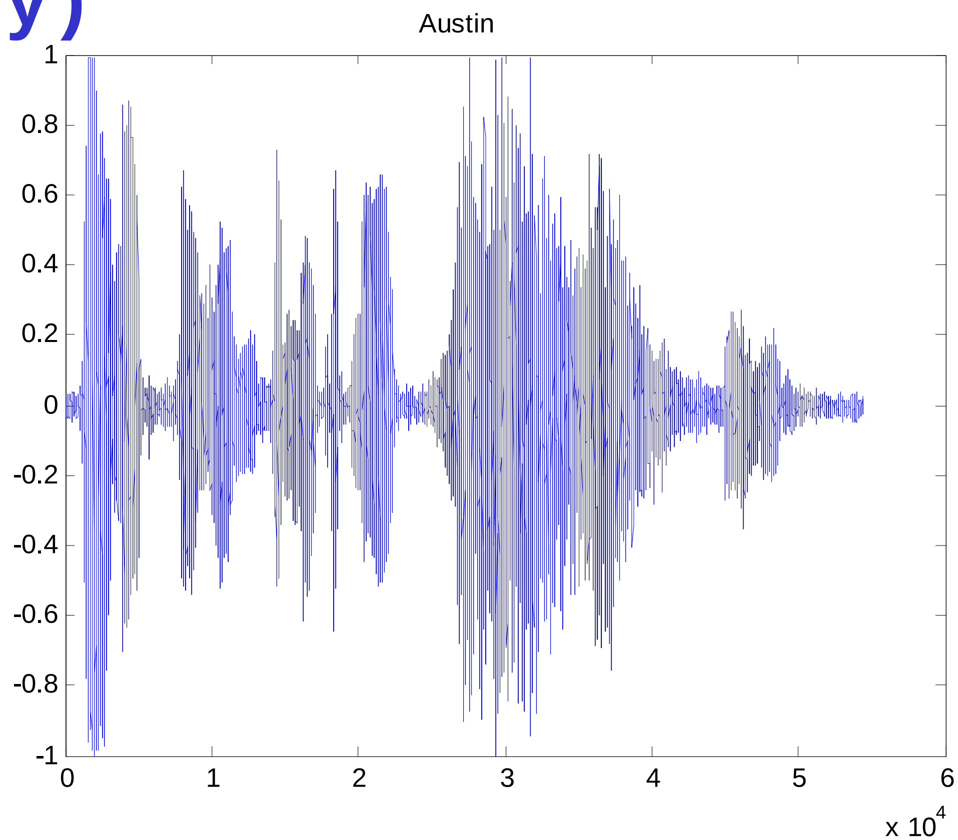
D: *none of the above*

Hearing and “seeing” the sound

```
[y,rate]=  wavread('austin');
sound(y, rate)
plot(1:length(y), y)
```

Usually playback at a rate equal to the sampling rate

See **showAustin.m**



movies.m

Example: playlist

Suppose we have a set of .wav files, e.g.,

```
austin.wav
sp_beam.wav
sp_oz6.wav
```

and wish to play them in succession.

Possible solution

```
playList = {'austin',…
            'sp_beam',…
            'sp_oz6'};

for k=1:length(playList)
   [y,rate] = wavread(playList{k});
   sound(y,rate)
end
```

Store the data from wav files as a struct array for play back later

```
function SA = wavSegments(wnames)
% Build a struct array SA such that
%   SA(k).data  stores the data of wnames{k}
%   SA(k).rate  stores the sampling rate of
%               wav file wnames{k}

for k= 1:length(wnames)
   [y,rate] = wavread(wnames{k});
   SA(k)= struct('data', y, 'rate', rate);
end
```

```
function playSegments(SA)
% Play sound data stored in struct array SA.
%   SA(k).data  stores the k-th segment of
%               sound data (from wavread)
%   SA(k).rate  is sampling rate of k-th seg.

for k= 1:length(SA)
    theData= SA(k).data;
    theRate= SA(k).rate;
    sound(theData,theRate)
end
```

Next call to sound will not begin until after the previous call is complete.

Not true in older versions! Calculate and add your own pause in that case.