L25. Working with Sound Files

```
wavread
sound
wavwrite
```

Reading and Playing

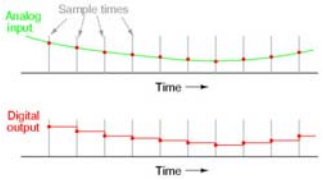
```
[y,rate,nBits] = wavread('noCry.wav')
sound(y,rate)
```

How is audio information encoded?

We will work with .wav files

Discrete vs Continuous

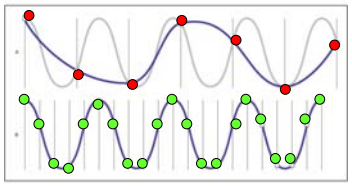
Sound is continuous.
Capture its essence by sampling.



Digitized sound = 1-dim array of numbers

Quality Issue: Sampling Rate

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



Too slow

OK

Sampling Rate (Cont'd)

Given human perception, about 20000 samples/second is pretty good.

i.e., 20000 Hertz

i.e., 20kHZ

Sampling Rate (Cont'd)

8,000 Hz	required for speech over the telephone
44,100 Hz	required for audio CD, MP3 files
192,400 Hz	required for HD-DVD audio tracks

Quality Issue: Resolution

Each sampled value is encoded as an eight-bit integer in the .wav file.

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

Loud: -120, 90, 122, etc

Quiet: 3, 10, -5

8-bits usually enough. Sometimes 16-bits

Back to **wavread**

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

n = length(y)

n =
       77804
rate =
       11025

nBits =
        8
```

noCry.wav encoded the sound with 77,804 8-bit numbers which were gathered over a span of about 77804/11025 secs

Let's Look at the Sample Values

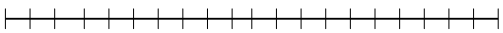
```
 0.07031250000000
-0.13281250000000
-0.67968750000000
-0.51562500000000
 0.21875000000000   ← y(50000:50010)
 0.48437500000000
 0.71093750000000
 0.57031250000000
 0.14843750000000
 0.39062500000000
 0.25000000000000
```

values in between -1 and 1.

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

Resolution with 8 Bits

For each sample, **wavread** produces one of 256 possible floating point values:

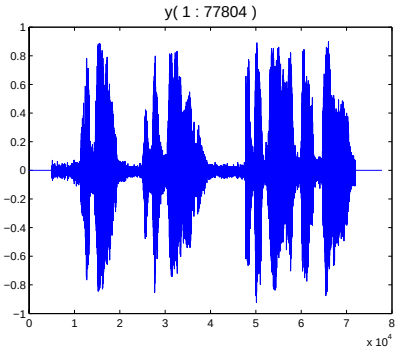


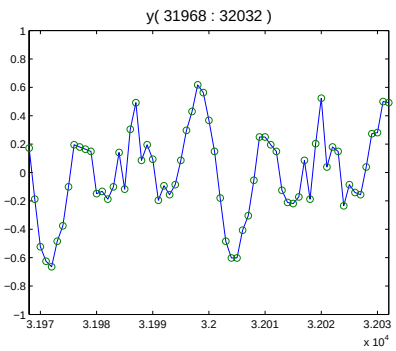
-1 +1

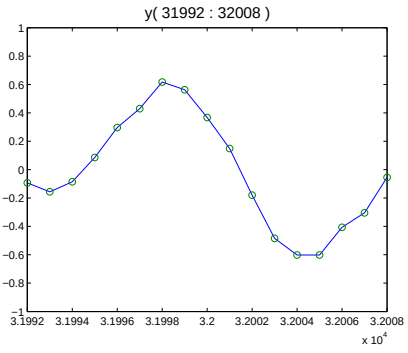
256 possible values: -1 + k/128

Let's Plot the Digitized Sound

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







Summary

Name of the source file.

Example:

```
[data,rate,nBits] = wavread('noCry.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

Shortcuts

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

[data,rate] = wavread('noCry')
```

Usually don't care about **nBits**

Don't need the .wav suffix

Playback via sound

```
[data,rate]= wavread('noCry')

sound(data,rate)
```

Usually want playback at a rate equal to the sampling rate.

To Compress or Not Compress

The .wav format does not involve compression.

The .mp3 format does involve compression and it reduces storage by a factor of about 10.

Playlist

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

```
Casablanca.wav
AustinPowers.wav
GoneWithWind.wav
BackToSchool.wav
```

and wish to play them in succession.

Three Problems

1. Making a Playlist.
2. Segmentation
3. Splicing

Problem 1: Making a Playlist

```
PlayList = {'Casablanca',…
            'AustinPowers',…
            'GoneWithWind',…
            'BackToSchool'}

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

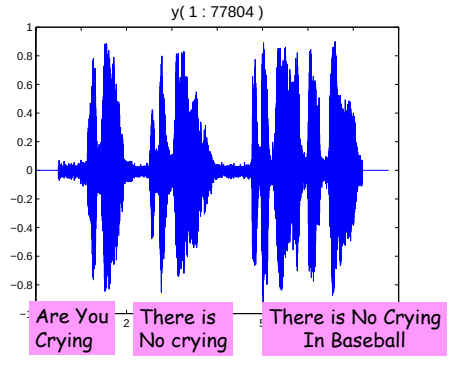
Prob: Will start playing Austin Powers before Casablanca finishes.

Built-In Delays

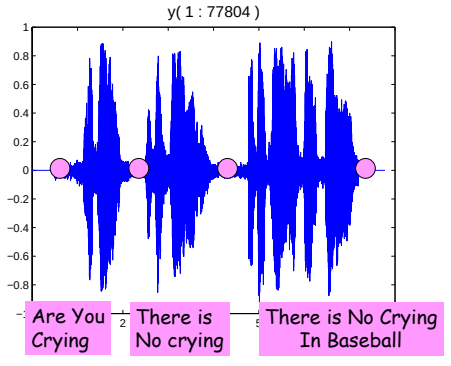
```
for k=1:length(PlayList)
   [y,rate] = wavread(PlayList{k});
   sound(y,rate)

   L = length(y)/rate;
   pause(L+1)

end
```

Compute how long it'll take and add one second.

Problem 2: Segmentation



Segmentation



Segmenting a `.wav` File

Read in a .wav file.
Display.
Identify segments using mouseclicks.
Store segments in a struct array.

Getting the Breakpoints

```
[data,rate] = wavread(F);
plot(data)
[x,y] = ginput;
x = floor(x);
nSegments = length(x)-1
```

Revisit ginput

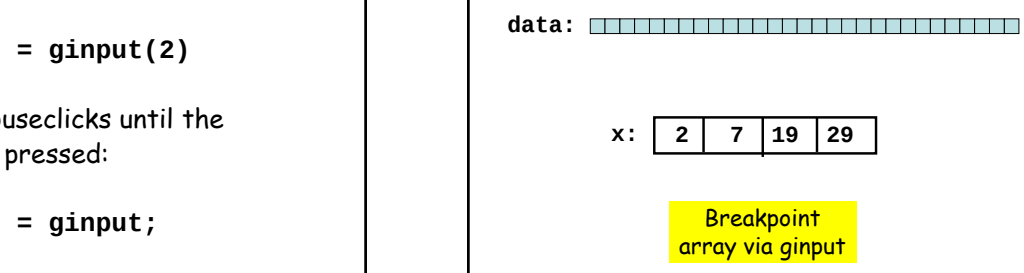
This solicits two mouseclicks:

```
[x,y] = ginput(2)
```

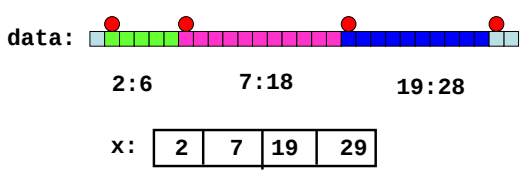
This solicits mouseclicks until the <return> key is pressed:

```
[x,y] = ginput;
```

From Breakpoints to Subarray Extraction



From Breakpoints to Subarray Extraction



For kth segment:
First = x(k); Last = x(k+1)-1

Setting up the Struct Array

```
for k=1:nSegments
   First = x(k);
   Last =  x(k+1)-1;
   z = data(First:Last);
   C(k) = struct('rate',rate,…
                         'z',z);
end
```

Playing the Segments

```
function PlaySegments(C)

for k=1:length(C)
   theData  = C(k).z;
   theRate  = C(k).rate;
   sound(theData,theRate)
   pause(length(theData)/theRate +1)
end
```

That's how long it takes the Kth segment to play.

Making a .wav File

Suppose C is a struct array consisting of N sound segments all sampled at the same Rate.

Make a single .wav file that plays the segments in reverse order.

Reversing the Segments

```
function Reverse(C,F)

n = length(C);
y = [];
for k = 1:n
   y = [ C(k).z ; y];
end
wavwrite(y,C(1).rate,8,[F '.wav']);
```

wavwrite

```
wavwrite(data,rate,nBits,fname)
```

Target file, e.g., **'F.wav'**