Computational Methods (PHYS 2030)

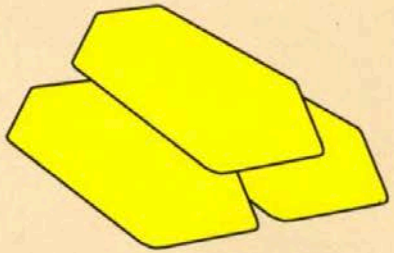
Instructors: Prof. Christopher Bergevin (cberge@yorku.ca)

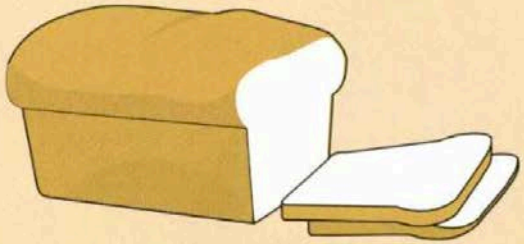
Schedule: Lecture: MWF 11:30-12:30 (CLH M)

Website: http://www.yorku.ca/cberge/2030W2018.html

York University
Winter 2018
Lecture x

Simply match up the pictures to the words. There's a particular sort of person that would find this puzzle very easy.

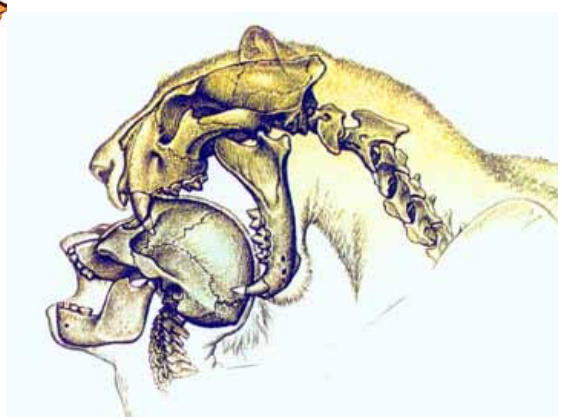




LIT DENT

OR

BRAS PAIN



Bed

Tooth

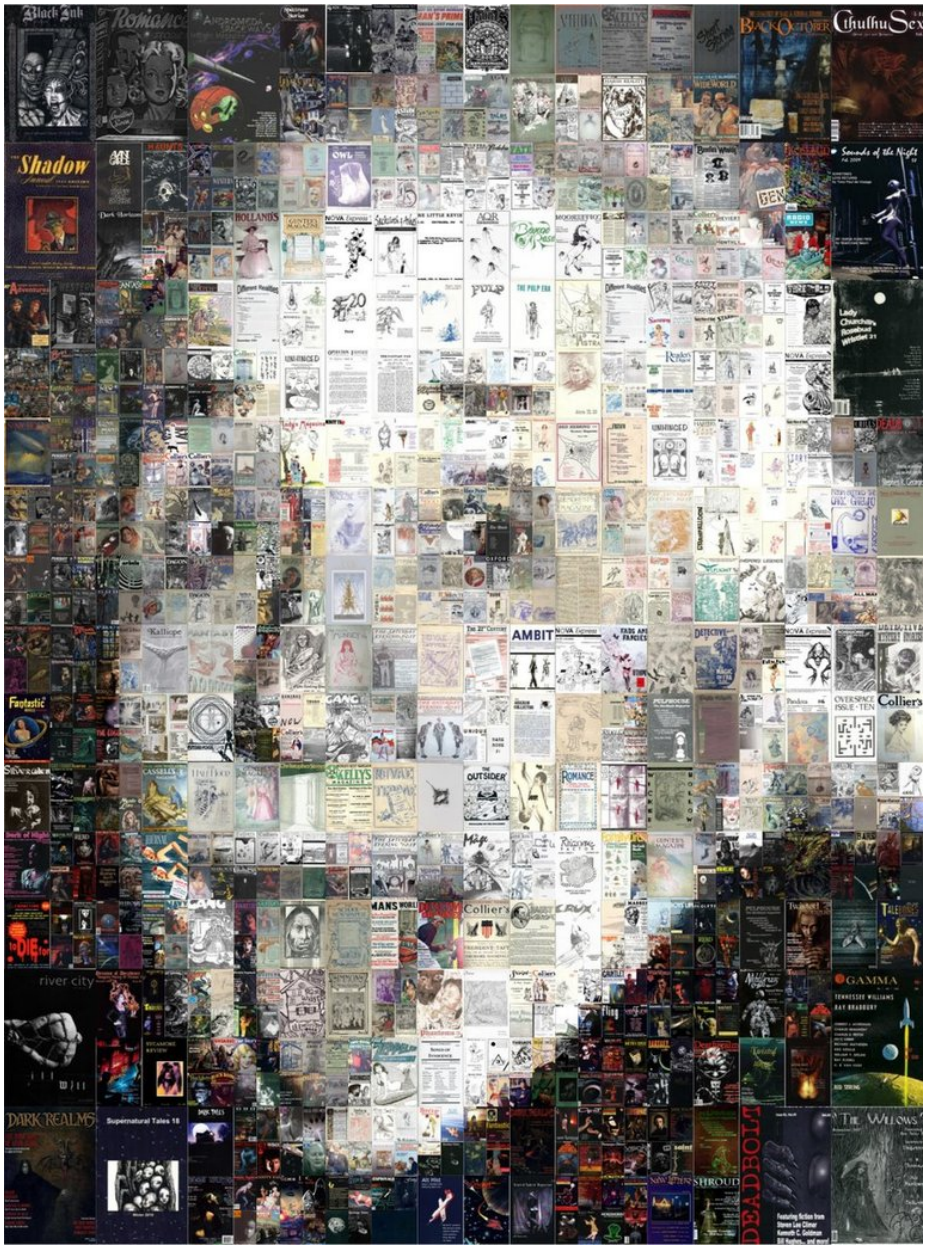
Gold

Arm

Bread


PULP
THE PULP ERA
AMBIT
NOVA
UNHINGED
DETECTIVE


PULP
THE PULP ERA
AMBIT
UNINVITED
DETECTIVE



- ‘Photo mosaics’ use images as an underlying set of ‘basis functions’

- Note that we could just as easily choose a different set of basis images....

... and it’s not too hard to imagine that some choices might be better than others!

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nt + \sum_{n=1}^{\infty} b_n \sin nt$$

→ Similar idea underlies the notion of Fourier analysis, the choice of basis functions being sinusoids

http://cornejo-sanchez.deviantart.com/art/Einstein-Mosaic-227306670

Sinusoids as basis functions

$$\int_{-\pi}^{\pi} \sin mt \sin nt \, dt = \begin{cases} \pi\delta_{m,n}, & m \neq 0, \\ 0, & m = 0, \end{cases}$$

$$\int_{-\pi}^{\pi} \cos mt \cos nt \, dt = \begin{cases} \pi\delta_{m,n}, & m \neq 0, \\ 2\pi, & m = n = 0, \end{cases}$$

$$\int_{-\pi}^{\pi} \sin mt \cos nt \, dt = 0, \quad \text{all integral } m \text{ and } n.$$

- Sinusoids make a good choice for basis functions, as they are 'complete' in that they are orthogonal to one another over the interval [0,2π]

→ The sum of the product over the interval is zero for disparate frequencies (i.e., they cancel one another out!)

→ *Similar idea as orthogonal unit vectors in coordinate space*

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nt + \sum_{n=1}^{\infty} b_n \sin nt$$

Fourier coefficients

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos nt \, dt,$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin nt \, dt.$$

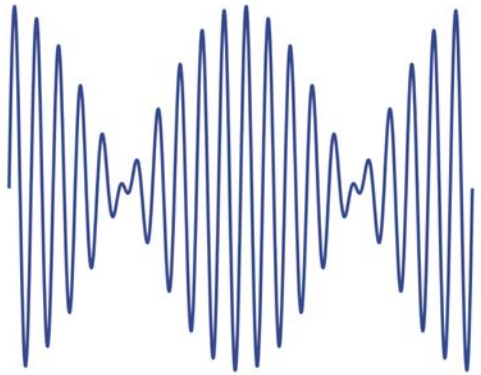
- This provides a 'recipe' for figuring out the appropriate weighting for each term

Devries (1994)

Reminder

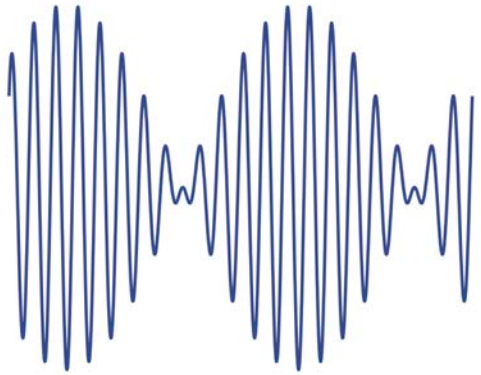
$$x(t) = A_1 \sin(2\pi f_1 t + \phi_1) + A_2 \sin(2\pi f_2 t + \phi_2)$$

$f_1=1, f_2=1.1$
$A_1=1, A_2=1$
$\phi_1=0, \phi_2=0$



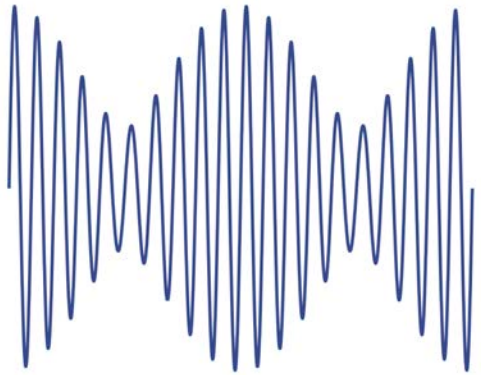
→ Changing (relative) phase affects summation

$f_1=1, f_2=1.1$
$A_1=1, A_2=1$
$\phi_1=\pi/2, \phi_2=0$



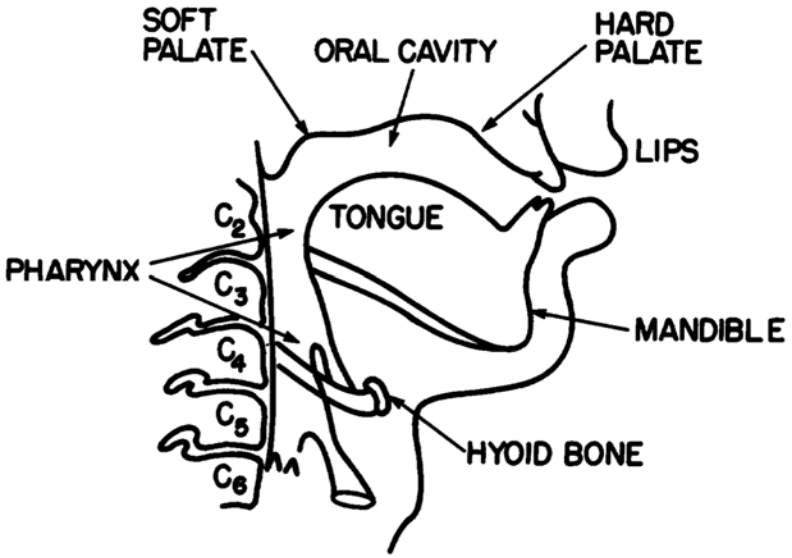
→ Changing (relative) amplitudes affects summation

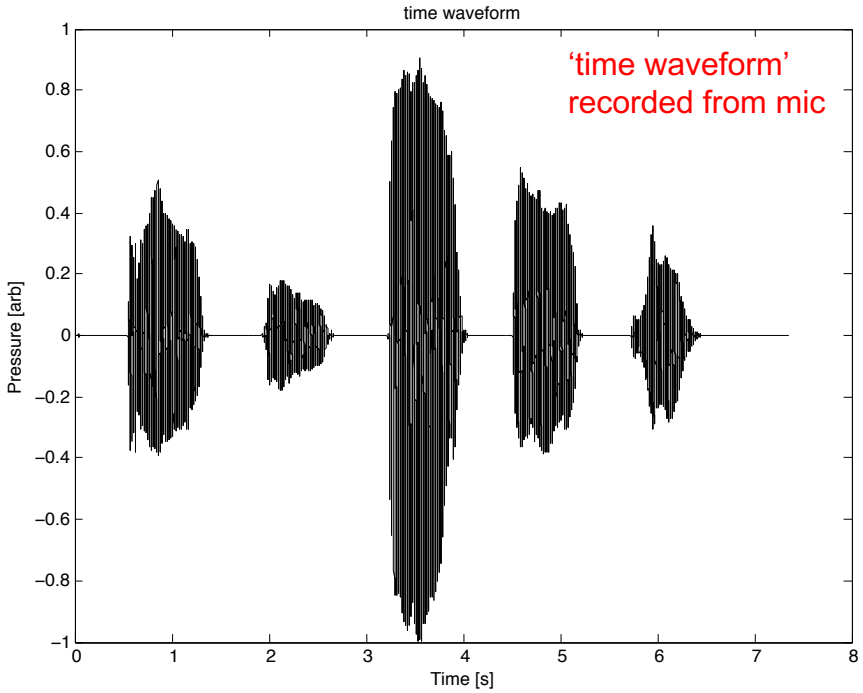
$f_1=1, f_2=1.1$
$A_1=2, A_2=1$
$\phi_1=0, \phi_2=0$

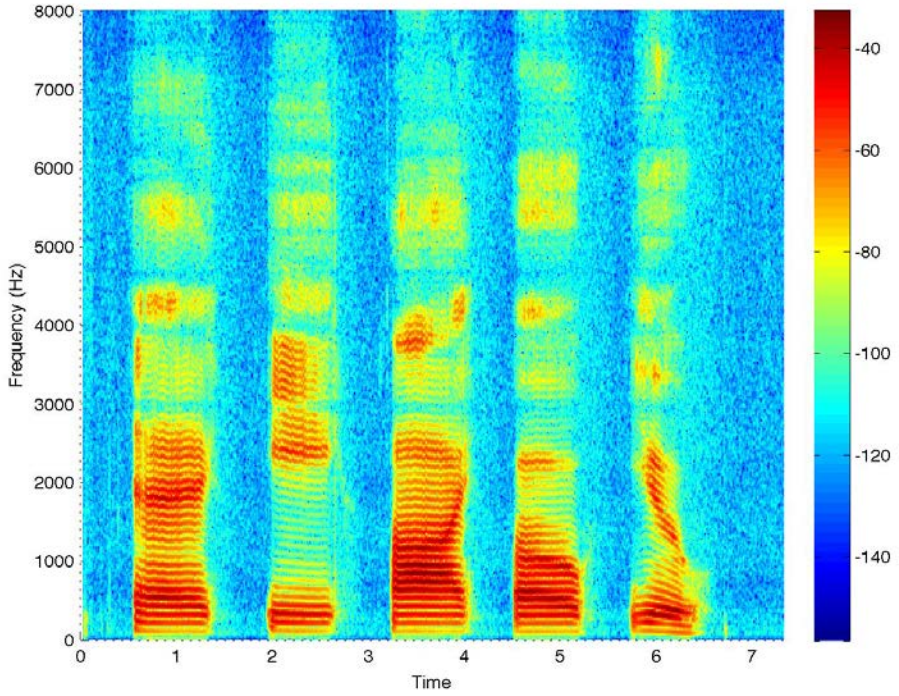


Speech & Vowels: A E I O U

EXspectrogram.m



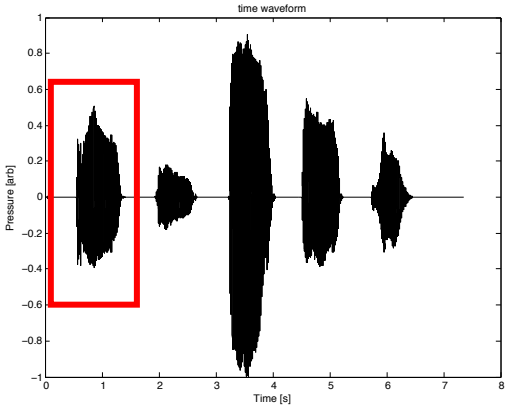


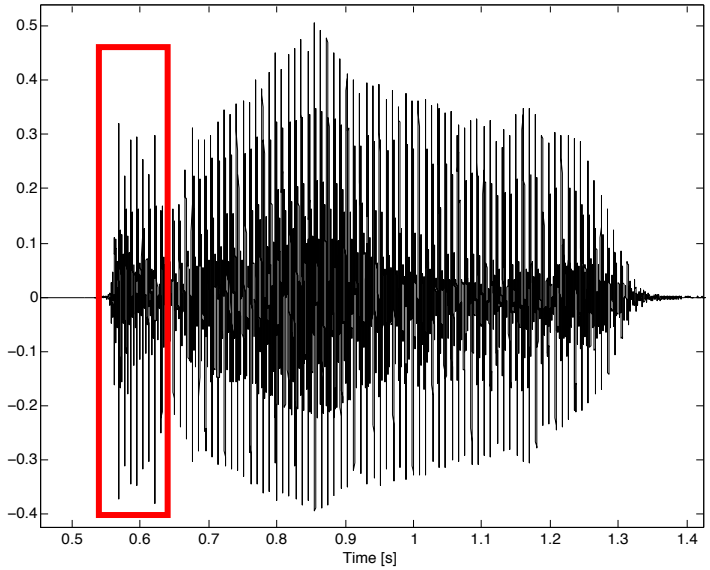


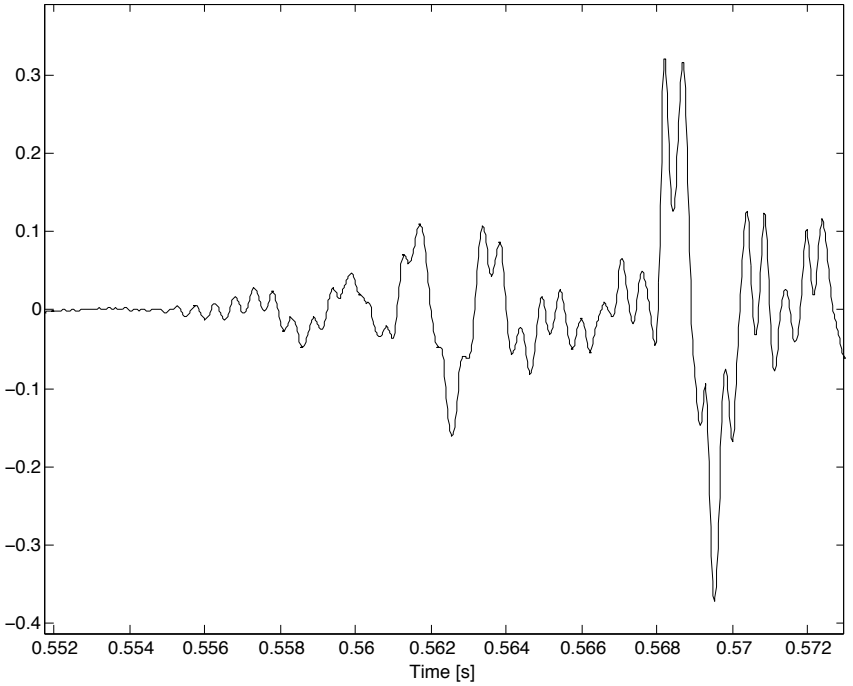
Stevens (2000)

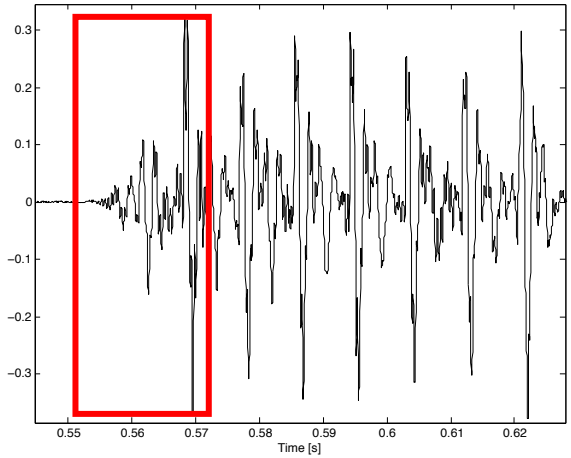
Speech & Vowels: A E I O U

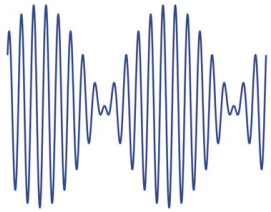
- Let's just focus on the initial vowel A (/*aI*/):







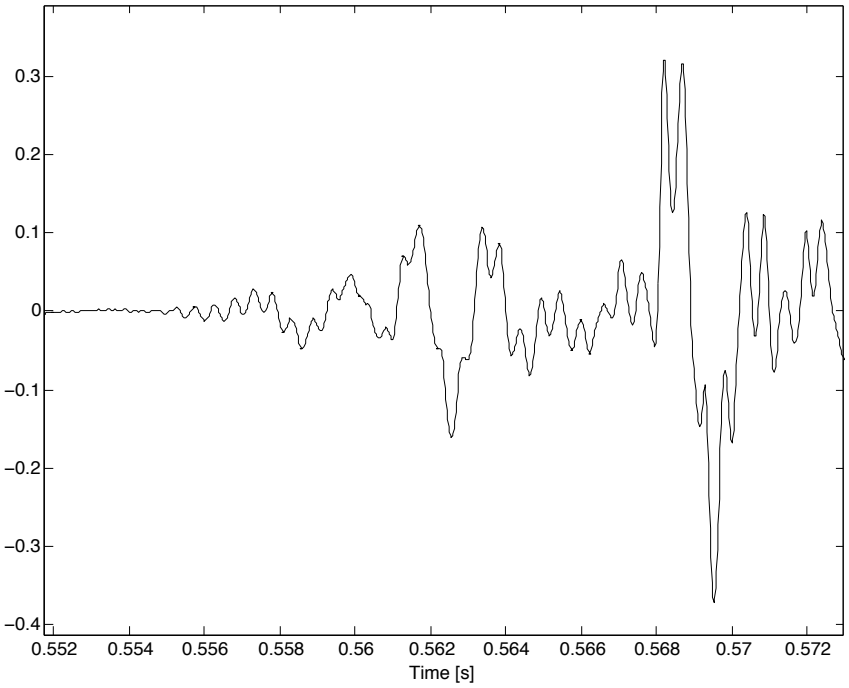




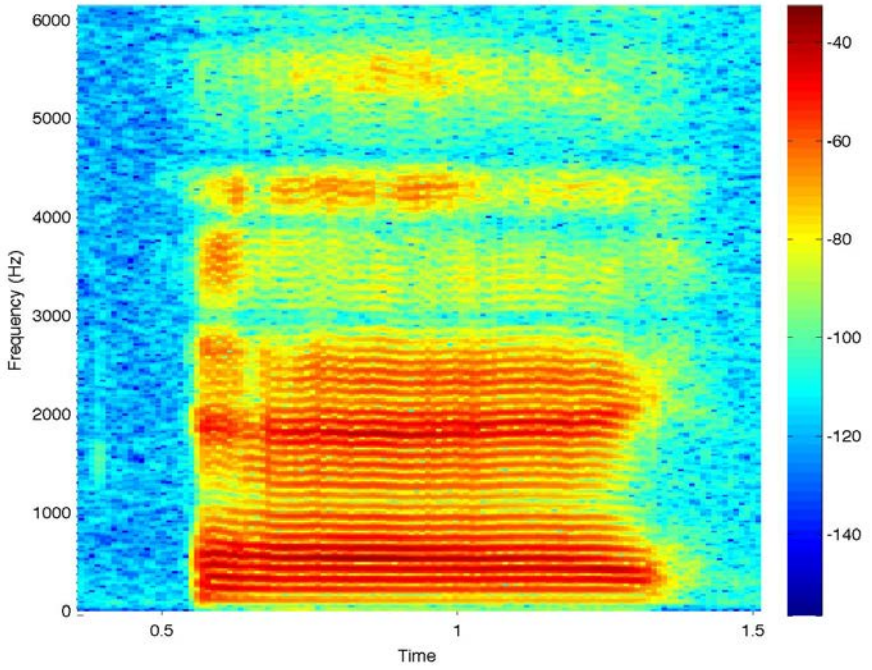
Just a sum of a bunch of sinusoids?

Speech & Vowels: A E I O U

- Let's just focus on the initial vowel A ($/aI/$):



Short Time Fourier Transform (STFT)
(we'll come back to what specifically this is)



- Speech is just one of *many* types of signals that can be very efficiently be described in terms of the summation of sinusoids

Fourier transforms

We now *define* $g(\omega)$ to be the Fourier transform of $f(t)$,

$$\mathcal{F}[f(t)] = g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt,$$

and $f(t)$ to be the *inverse* transform of $g(\omega)$,

$$\mathcal{F}^{-1}[g(\omega)] = f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega) e^{i\omega t} d\omega.$$

- Seemingly abstract/simple idea has vast implications in terms of how we encode and decipher information (e.g., signal processing) as well as mathematical methods in physics and linear systems theory

Devries (1994)

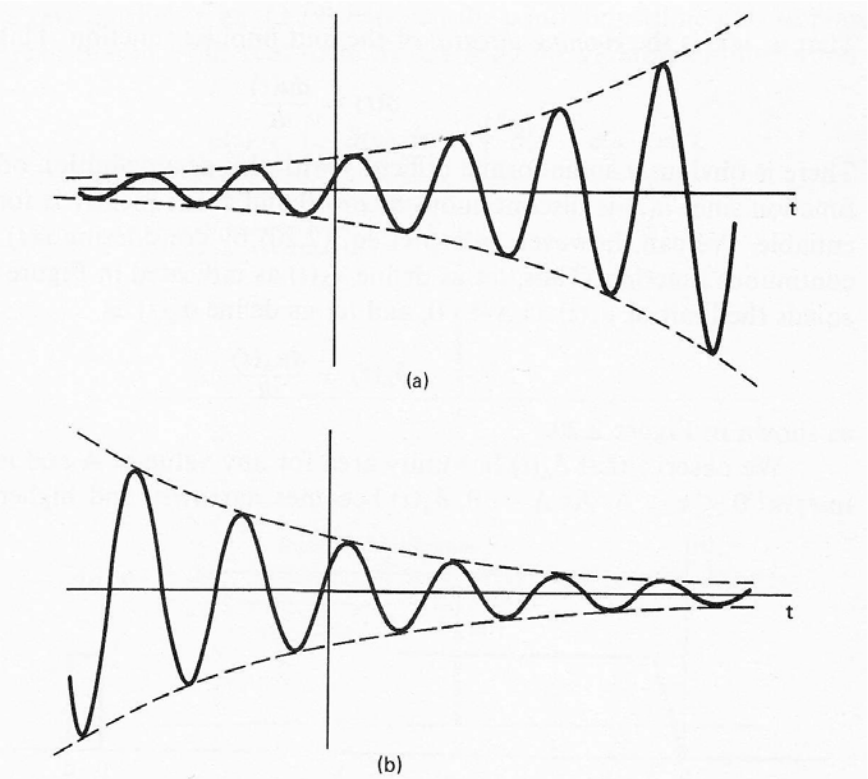


Figure 2.17 (a) Growing sinusoidal signal $x(t) = Ce^{rt}\cos(\omega_0 t + \theta)$, $r > 0$; (b) decaying sinusoid $x(t) = Ce^{rt}\cos(\omega_0 t + \theta)$, $r < 0$.

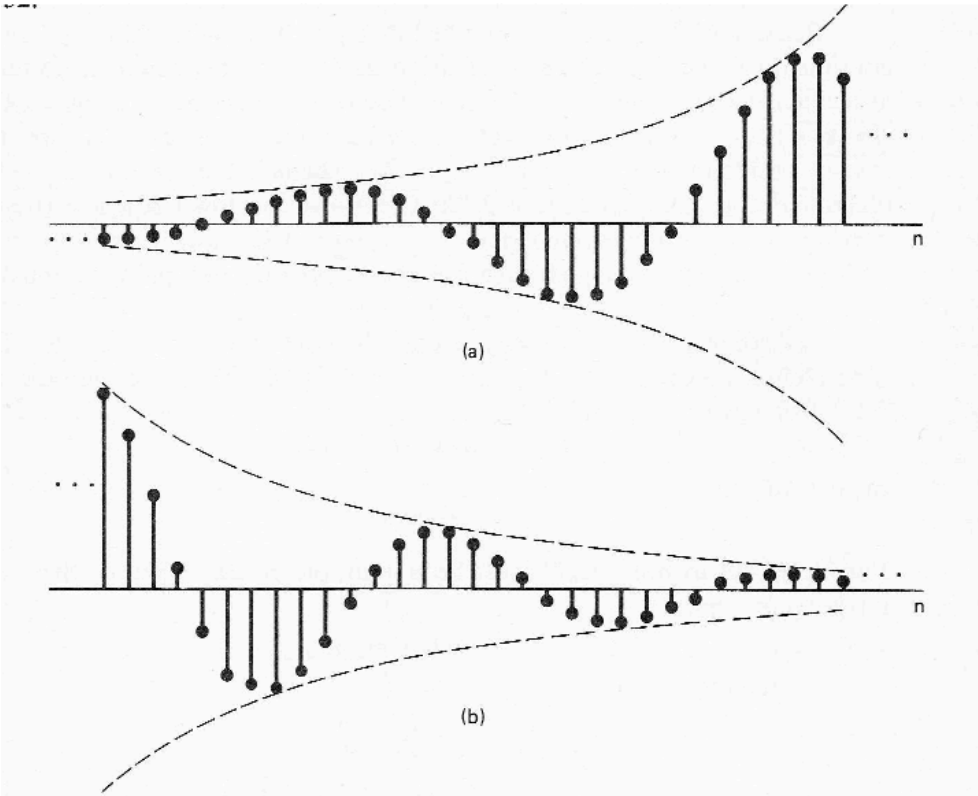


Figure 2.32 (a) Growing discrete-time sinusoidal signal; (b) decaying discrete-time sinusoid.

Continuous signal (analog)

Discretized signal (digital)

Question: Does Fourier analysis 'care' whether things are continuous or discrete?

Yes & no (we'll come back to this a bit later)

Oppenheim & Willsky

FFT (Fast Fourier Transform)

An Algorithm for the Machine Calculation of Complex Fourier Series
Author(s): James W. Cooley and John W. Tukey
Source: *Mathematics of Computation*, Vol. 19, No. 90 (Apr., 1965), pp. 297-301

- Means to efficiently deal with discrete Fourier transforms (e.g., a digitally sampled waveform)

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Consider the problem of calculating the complex Fourier series

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W^{jk}, \qquad j = 0, 1, \cdots, N-1, \tag{1}$$

where the given Fourier coefficients $A(k)$ are complex and W is the principal Nth root of unity,

$$W = e^{2\pi i/N}. \tag{2}$$

A straightforward calculation using (1) would require N^2 operations where "operation" means, as it will throughout this note, a complex multiplication followed by a complex addition.

The algorithm described here iterates on the array of given complex Fourier amplitudes and yields the result in less than $2N \log_2 N$ operations without requiring more data storage than is required for the given array A. To derive the algorithm, suppose N is a composite, i.e., $N = r_1 \cdot r_2$. Then let the indices in (1) be expressed

$$\begin{aligned} j &= j_1 r_1 + j_0, \qquad j_0 = 0, 1, \cdots, r_1 - 1, \qquad j_1 = 0, 1, \cdots, r_2 - 1, \\ k &= k_1 r_2 + k_0, \qquad k_0 = 0, 1, \cdots, r_2 - 1, \qquad k_1 = 0, 1, \cdots, r_1 - 1. \end{aligned} \tag{3}$$

Then, one can write

$$X(j_1, j_0) = \sum_{k_0} \sum_{k_1} A(k_1, k_0) \cdot W^{jk_1 r_2} W^{jk_0}. \tag{4}$$

Received August 17, 1964. Research in part at Princeton University under the sponsorship of the Army Research Office (Durham). The authors wish to thank Richard Garwin for his essential role in communication and encouragement.

FFT (Fast Fourier Transform)

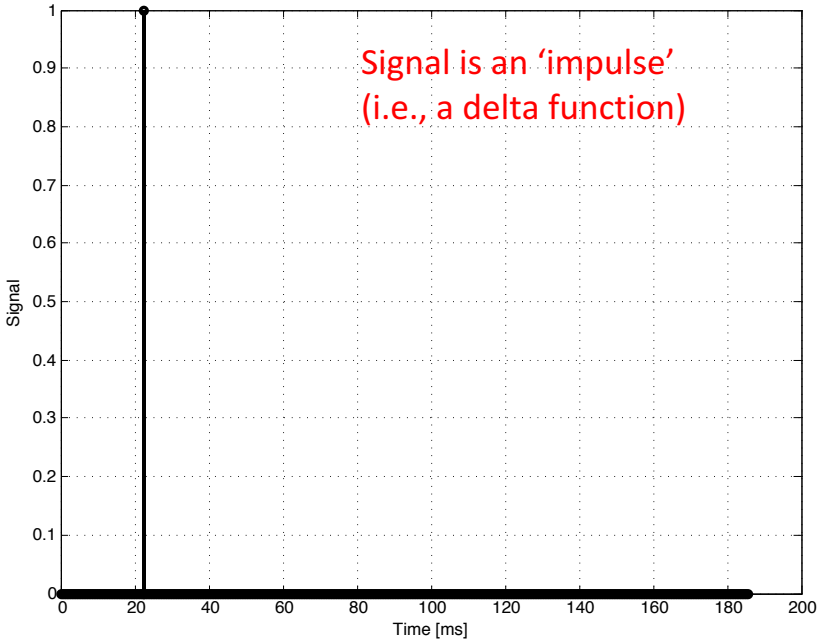
The Fourier transform is actually relatively easy to compute via the **FFT**

The calculation of the Fourier coefficients using our equations involves N evaluations of the sine or cosine, N multiplications, and N additions for each coefficient. There are N coefficients, so that there must be N^2 evaluations of the sines and cosines, which uses a lot of computer time. Cooley and Tukey (1965) showed that it is possible to group the data in such a way that the number of multiplications is about $(N/2)\log_2 N$ instead of N^2 and the sines and cosines need to be evaluated only once, a technique known as the *Fast Fourier Transform* (FFT). For example, for $1024 = 2^{10}$ data points, $N^2 = 1\,048\,576$, while $(N/2)\log_2 N = (512)(10) = 5120$. This speeds up the calculation by a factor of 204.

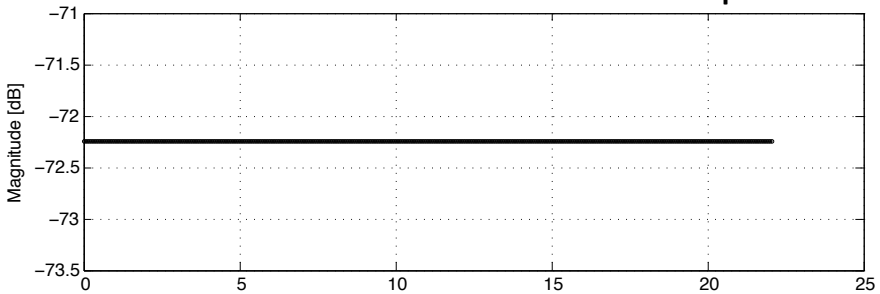
Hobbie & Roth (2007)

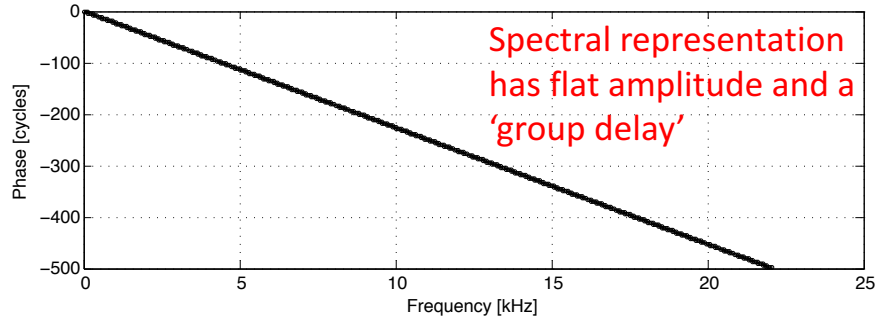
```
% ### EXbuildImpulse.m ###       11.03.14
% Code to visually build up a signal by successively adding higher and
% higher frequency terms from corresponding FFT
clear; clf;
% --------------------------------
SR= 44100;          % sample rate [Hz]
Npoints= 8192;      % length of fft window (# of points) [should ideally be 2^N]
                    % [time window will be the same length]
INDXon= 1000;      % index at which click turns 'on' (i.e., go from 0 to 1)
INDXoff= 1001;    % index at which click turns 'off' (i.e., go from 1 to 0)
% --------------------------------

dt= 1/SR;  % spacing of time steps
freq= [0:Npoints/2];    % create a freq. array (for FFT bin labeling)
freq= SR*freq./Npoints;
t=[0:1/SR:(Npoints-1)/SR]; % create an appropriate array of time points
% build signal
clktemp1= zeros(1,Npoints); clktemp2= ones(1,INDXoff-INDXon);
signal= [clktemp1(1:INDXon-1) clktemp2 clktemp1(INDXoff:end)];
% -------------------------------
% *******
% plot time waveform of signal
if 1==1
    figure(1); clf; plot(t*1000,signal,'ko-','MarkerSize',5)
    grid on; hold on; xlabel('Time [ms]'); ylabel('Signal'); title('Time Waveform')
end
% *******
% now compute/plot FFT of the signal
sigSPEC= rfft(signal);
% MAGNITUDE
figure(2); clf;
subplot(211); plot(freq/1000,db(sigSPEC),'ko-','MarkerSize',3)
hold on; grid on; ylabel('Magnitude [dB]'); title('Spectrum')
% PHASE
subplot(212); plot(freq/1000,cycs(sigSPEC),'ko-','MarkerSize',3)
xlabel('Frequency [kHz]'); ylabel('Phase [cycles]'); grid on;
% *******
% now make animation of click getting built up, using the info from the FFT
sum= zeros(1,numel(t)); % (initial) array for reconstructed waveform
figure(3); clf;
for nn=1:numel(freq)
    sum= sum+ abs(sigSPEC(nn))*cos(2*pi*freq(nn)*t + angle(sigSPEC(nn)));
    plot(t,sum); xlabel('Time [s]');
    legend(['Highest freq= ',num2str(freq(nn)/1000),' kHz'])
    pause(2/(nn))
end
```

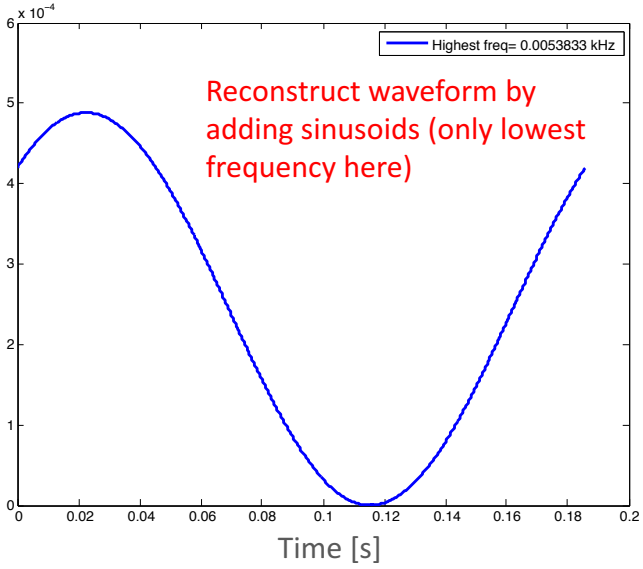
Temporal

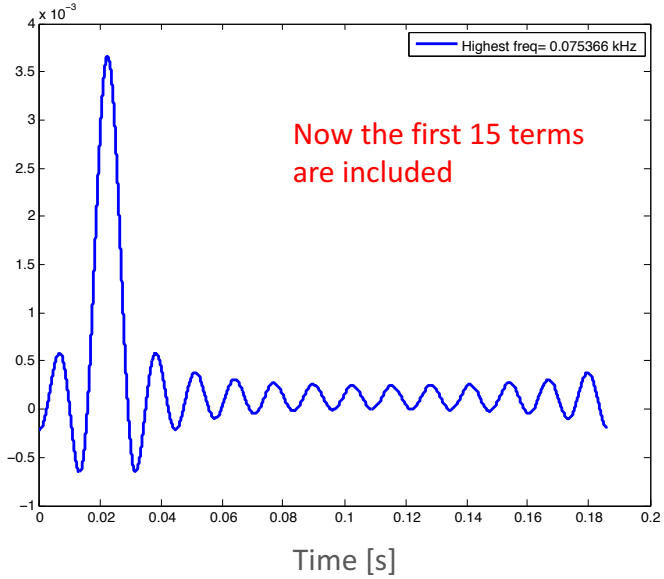


Spectral









→ Eventually all the sinusoids add up such that things cancel out everywhere except at the point of the impulse!

```
% ### EXspecREP3.m ###       10.29.14
% Example code to just fiddle with basics of discrete FFTs and connections
% back to common real-valued time waveforms
% --> Demonstrates several useful concepts such as 'quantizing' the frequency
% Requires: rfft.m, irfft.m, cycs.m, db.m, cyc.m
% ------
% Stimulus Type Legend
% stimT= 0 - non-quantized sinusoid
% stimT= 1 - quantized sinusoid
% stimT= 2 - one quantized sinusoid, one un-quantized sinusoid
% stimT= 3 - two quantized sinusoids
% stimT= 4 - click I.e., an impulse)
% stimT= 5 - noise (uniform in time)
% stimT= 6 - chirp (flat mag.)
% stimT= 7 - noise (Gaussian; flat spectrum, random phase)
% stimT= 8 - exponentially decaying sinusoid (i.e., HO impulse response)

clear; clf;
% ---------------------------------
SR= 44100;          % sample rate [Hz]
Npoints= 8192;      % length of fft window (# of points) [should ideally be 2^N]
                    % [time window will be the same length]
stimT= 8;   % Stimulus Type (see legend above)
f= 2580.0;          % Frequency (for waveforms w/ tones) [Hz]
ratio= 1.22;    % specify f2/f2 ratio (for waveforms w/ two tones)
% Note: Other stimulus parameters can be changed below
% ---------------------------------
dt= 1/SR;  % spacing of time steps
freq= [0:Npoints/2];    % create a freq. array (for FFT bin labeling)
freq= SR*freq./Npoints;
% quantize the freq. (so to have an integral # of cycles in time window)
df = SR/Npoints;
fQ= ceil(f/df)*df;   % quantized natural freq.
t=[0:1/SR:(Npoints-1)/SR];  % create an array of time points, Npoints long
% ----
% compute stimulus
if stimT==0 % non-quantized sinusoid
    signal= cos(2*pi*f*t);
    disp(sprintf(' \n *Stimulus* - (non-quantized) sinusoid, f = %g Hz \n', f));
    disp(sprintf('specified freq. = %g Hz', f));
elseif stimT==1     % quantized sinusoid
    signal= cos(2*pi*fQ*t);
    disp(sprintf(' \n *Stimulus* - quantized sinusoid, f = %g Hz \n', fQ));
    disp(sprintf('specified freq. = %g Hz', f));
    disp(sprintf('quantized freq. = %g Hz', fQ));
elseif stimT==2     % one quantized sinusoid, one un-quantized sinusoid
    signal= cos(2*pi*fQ*t) + cos(2*pi*ratio*fQ*t);
    disp(sprintf(' \n *Stimulus* - two sinusoids (one quantized, one not) \n'));
elseif stimT==3     % two quantized sinusoids
    fQ2= ceil(ratio*f/df)*df;
    signal= cos(2*pi*fQ*t) + cos(2*pi*fQ2*t);
    disp(sprintf(' \n *Stimulus* -  two sinusoids (both quantized) \n'));
elseif stimT==4     % click
    CLKon= 1000;     % index at which click turns 'on' (starts at 1)
    CLKoff= 1001;   % index at which click turns 'off'
    clktemp1= zeros(1,Npoints);
    clktemp2= ones(1,CLKoff-CLKon);
    signal= [clktemp1(1:CLKon-1) clktemp2 clktemp1(CLKoff:end)];
    disp(sprintf(' \n *Stimulus* - Click \n'));
elseif stimT==5     % noise (flat)
    signal= rand(1,Npoints);
    disp(sprintf(' \n *Stimulus* - Noise1 \n'));
elseif stimT==6     % chirp (flat)
    f1S= 2000.0;     % if a chirp (stimT=2) starting freq. [Hz] [freq. swept linearly w/ time]
    f1E= 4000.0;   % ending freq.  (energy usually extends twice this far out)
    f1SQ= ceil(f1S/df)*df;      %quantize the start/end freqs. (necessary?)
    f1EQ= ceil(f1E/df)*df;
    % LINEAR sweep rate
    fSWP= f1SQ + (f1EQ-f1SQ)*(SR/Npoints)*t;
    signal = sin(2*pi*fSWP.*t)';
    disp(sprintf(' \n *Stimulus* - Chirp \n'));
elseif stimT==7     % noise (Gaussian)
    Asize=Npoints/2 +1;
    % create array of complex numbers w/ random phase and unit magnitude
    for n=1:Asize
        theta= rand*2*pi;
        N2(n)= exp(i*theta);
    end
    N2=N2';
    % now take the inverse FFT of that using Chris' irfft.m code
    tNoise=irfft(N2);
    % scale it down so #s are between -1 and 1 (i.e. normalize)
    if (abs(min(tNoise)) > max(tNoise))
        tNoise= tNoise/abs(min(tNoise));
    else
        tNoise= tNoise/max(tNoise);
    end
    signal= tNoise;
    disp(sprintf(' \n *Noise* - Gaussian, flat-spectrum \n'));
elseif stimT==8 % exponentially decaying cos
    alpha= 500;
    signal= exp(-alpha*t).*sin(2*pi*fQ*t);
    disp(sprintf(' \n *Exponentially decaying (quantized) sinusoid*  \n'));
end

% --------------------------------
% *******
figure(1); clf      % plot time waveform of signal
plot(t*1000,signal,'k.-','MarkerSize',5); grid on; hold on;
xlabel('Time [ms]'); ylabel('Signal'); title('Time Waveform')
% *******
% now plot rfft of the signal
% NOTE: rfft just takes 1/2 of fft.m output and nomalizes
sigSPEC= rfft(signal);
figure(2); clf; % MAGNITUDE
subplot(211)
plot(freq/1000,db(sigSPEC),'ko-','MarkerSize',3)
hold on; grid on;
ylabel('Magnitude [dB]')
title('Spectrum')
subplot(212)    % PHASE
plot(freq/1000,cycs(sigSPEC),'ko-','MarkerSize',3)
xlabel('Frequency [kHz]'); ylabel('Phase [cycles]'); grid on;
% -------
% play the stimuli as an output sound?
if (1==1),  sound(signal,SR);   end
% -------
% compute inverse Fourier transform and plot?
if 1==1
    figure(1);
    signalINV= irfft(sigSPEC);
    plot(t*1000,signalINV,'rx','MarkerSize',4)
    legend('Original waveform','Inverse transformed')
end
```

Fourier transforms of basic (1-D) waveforms

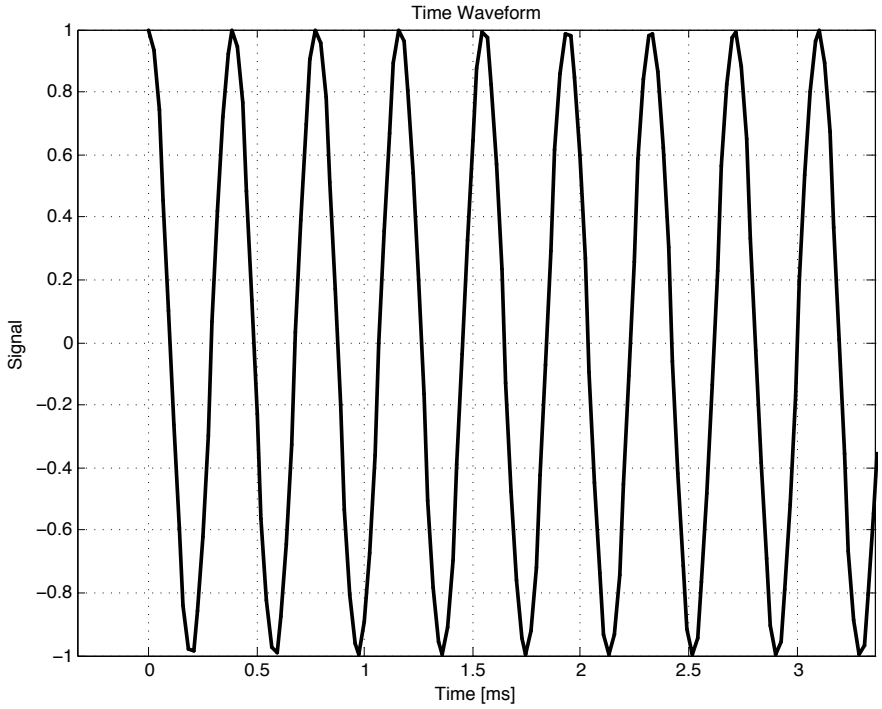
EXspecREP3.m

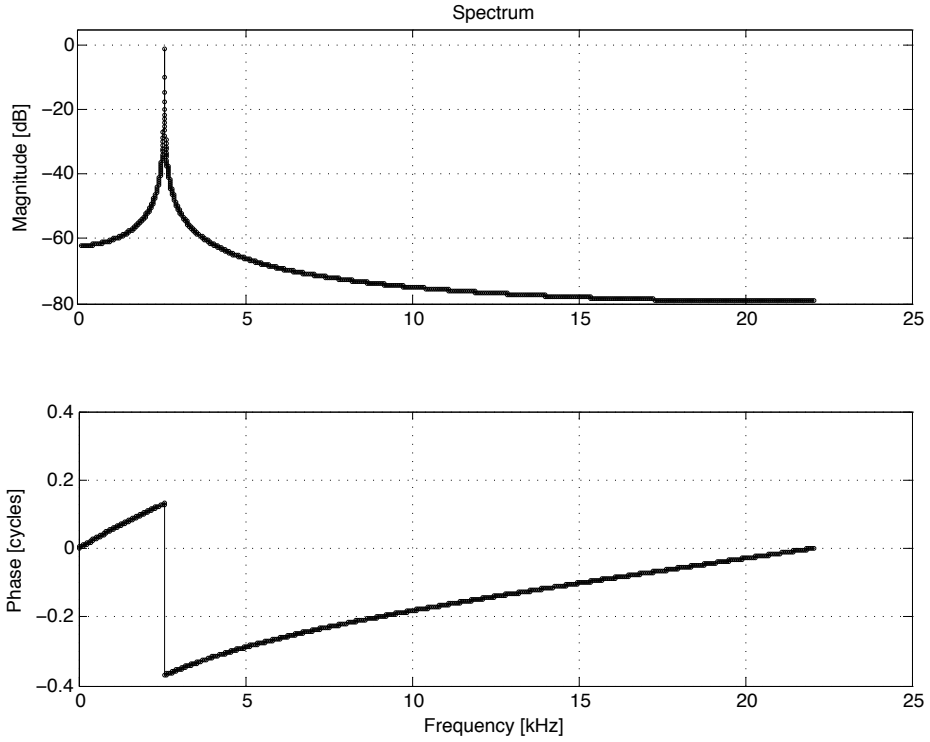
```
stimT= 0 - non-quantized sinusoid
```

```
SR= 44100;  % sample rate [Hz]
Npoints= 8192; % length of fft window
```

Time domain

Spectral domain





Note: The phase is 'unwrapped' in all the spectral plots

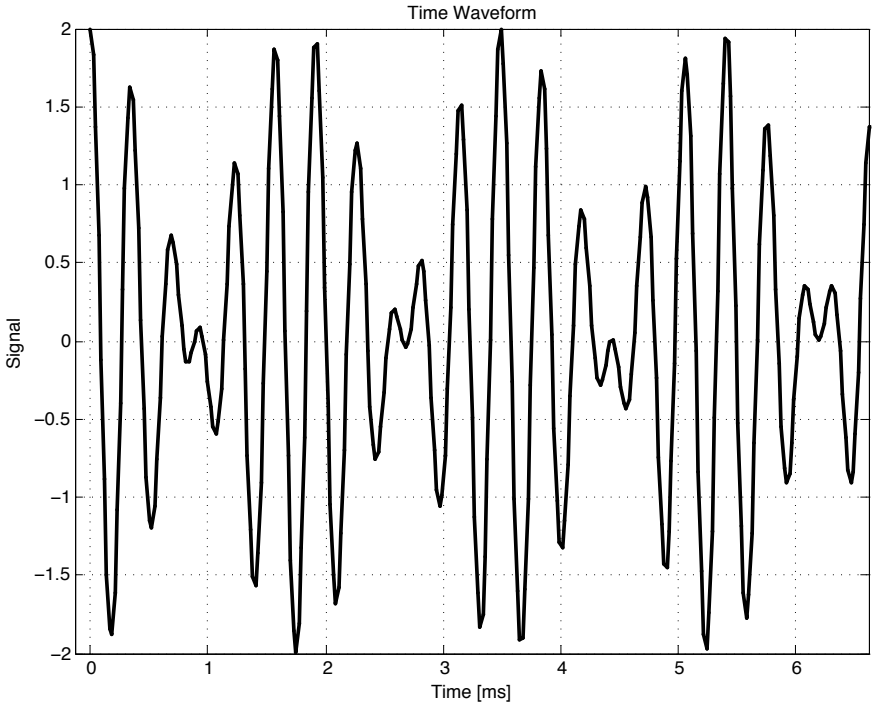
- Magnitude shows a peak at the sinusoid's frequency

Fourier transforms of basic (1-D) waveforms

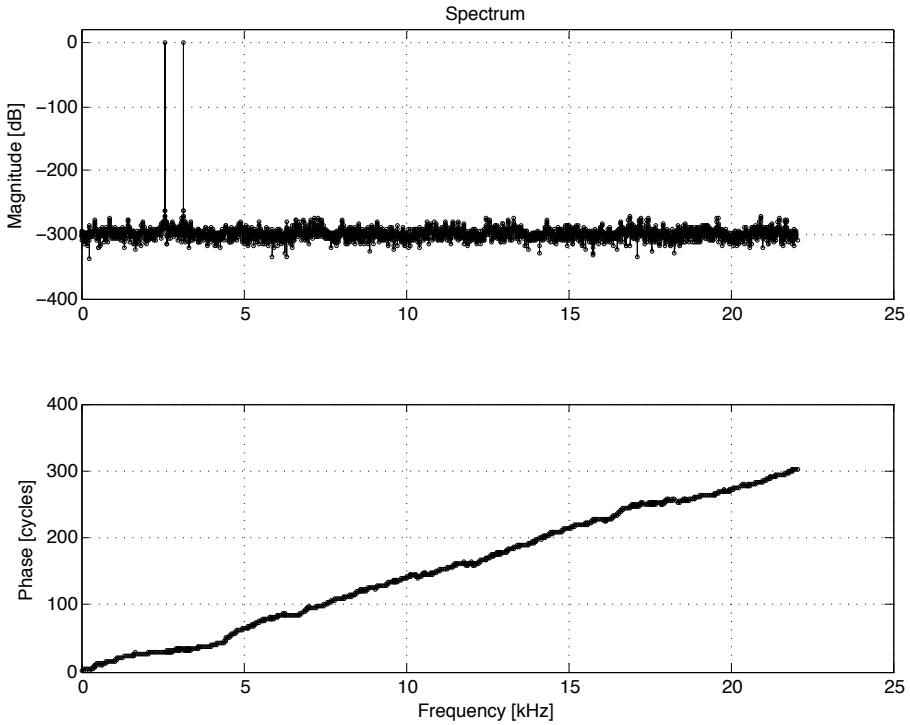
EXspecREP3.m

`stimT= 3 - two quantized sinusoids`

Time domain



Spectral domain



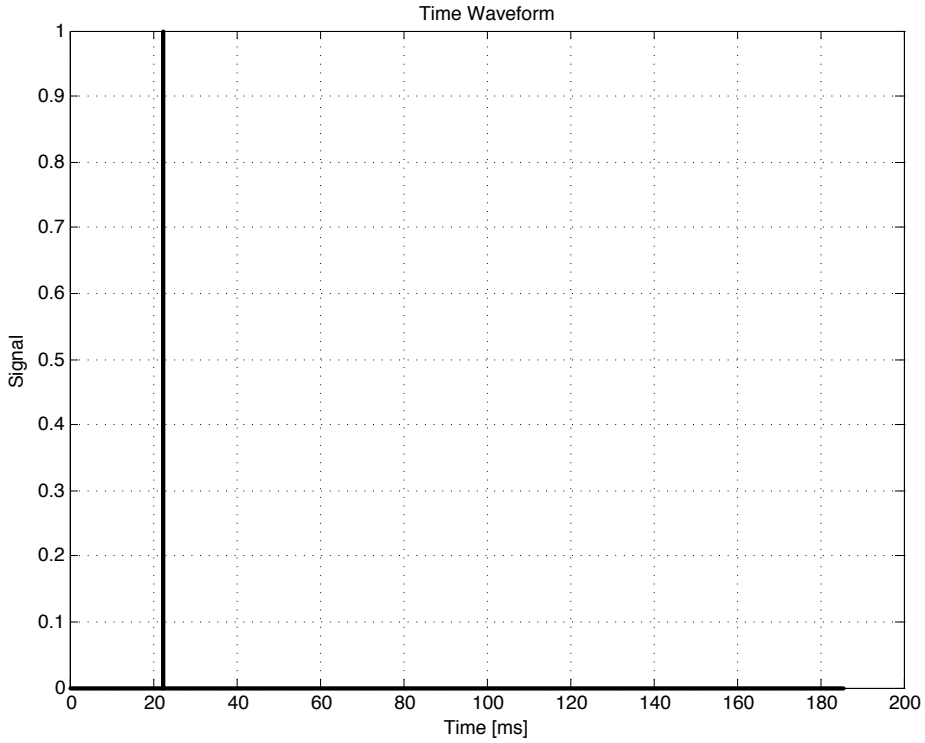
- Magnitude shows two peaks (note the ‘beating’ in the time domain)

Fourier transforms of basic (1-D) waveforms

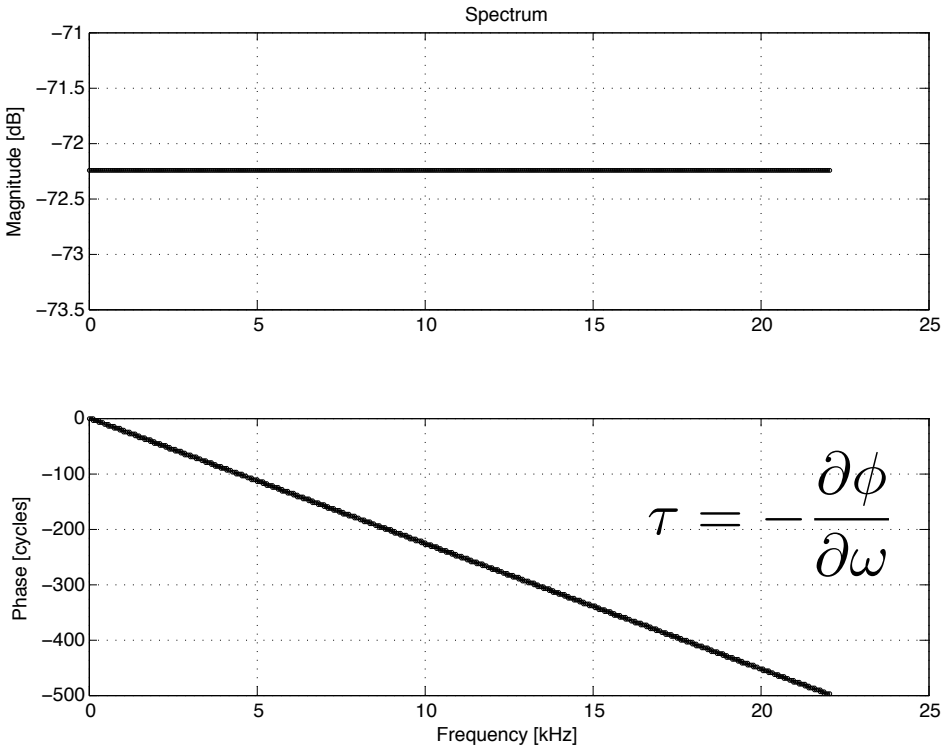
EXspecREP3.m

```
stimT= 4 - click (i.e., an impulse)
```

Time domain



Spectral domain



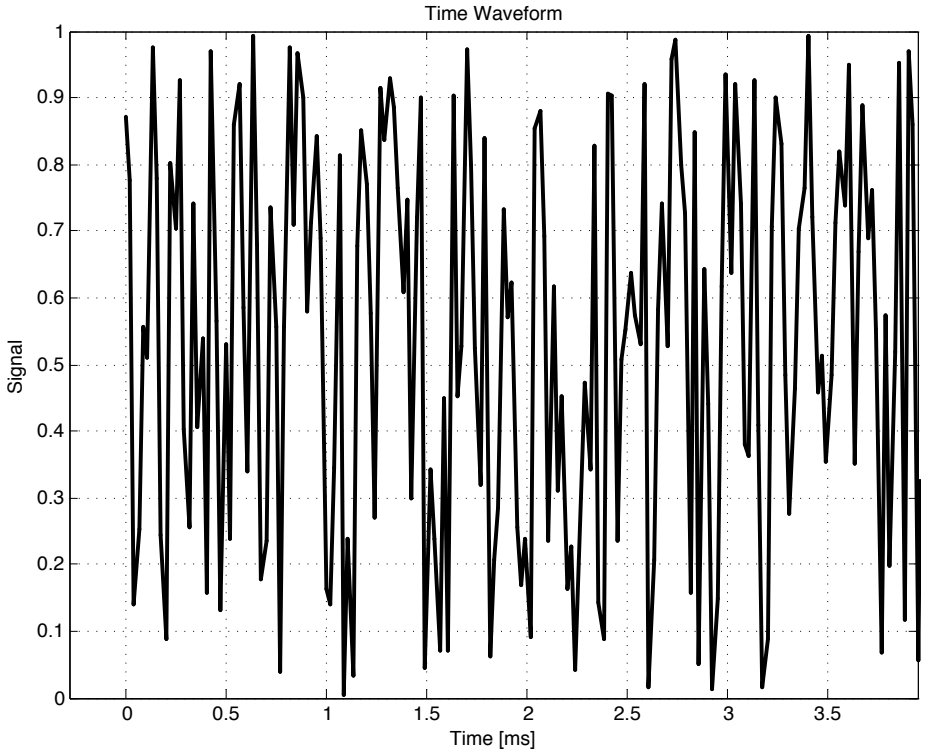
- Click has a flat magnitude (This is also a good place to mention the concept of a '*group delay*')

Fourier transforms of basic (1-D) waveforms

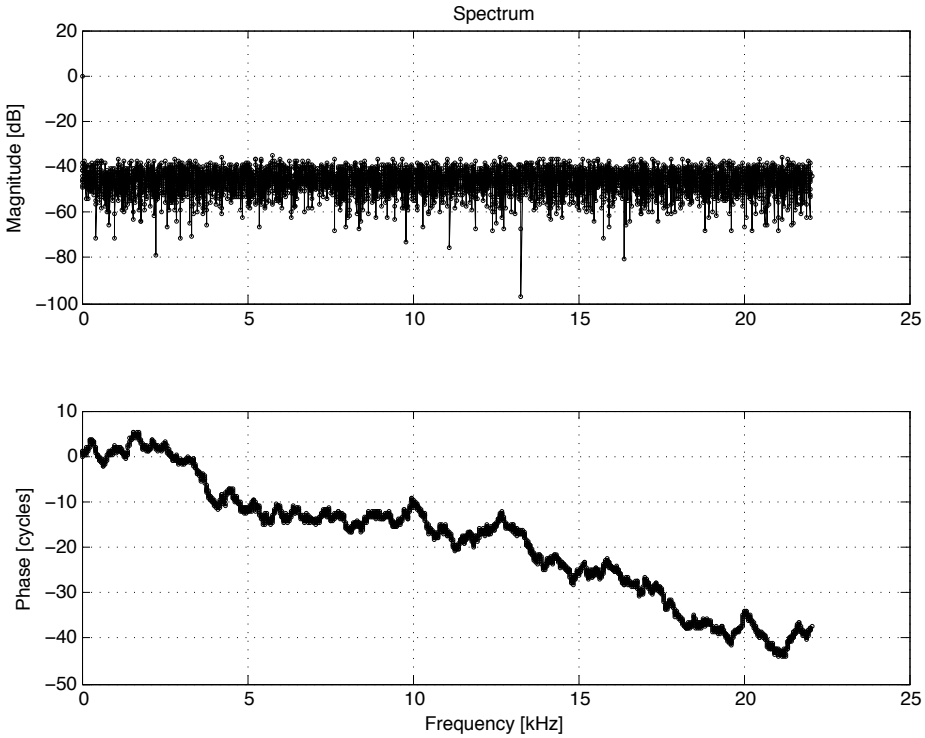
EXspecREP3.m

```
stimT= 5 - noise (uniform distribution)
```

Time domain



Spectral domain



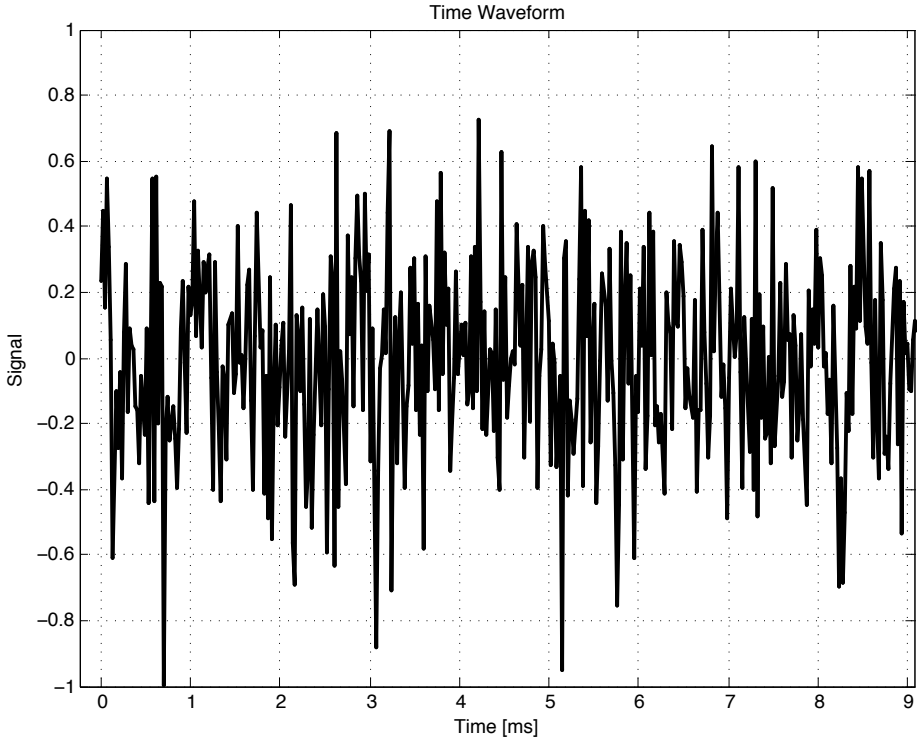
- Magnitude is flat-ish (on log scale), but actually noisy. Phase is noisy too.

Fourier transforms of basic (1-D) waveforms

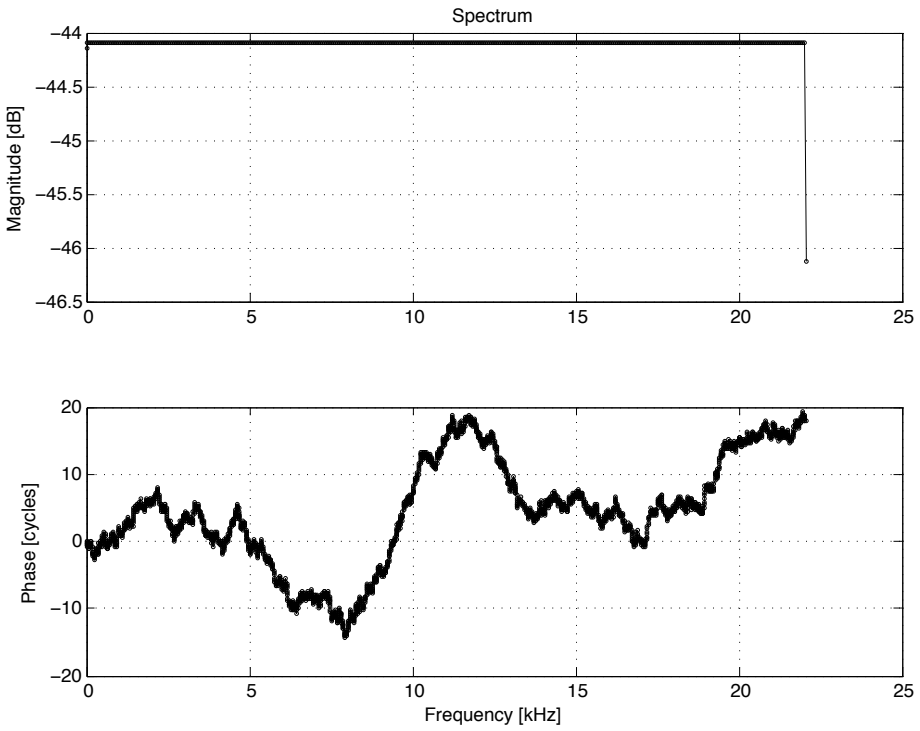
EXspecREP3.m

```
stimT= 7 - noise (Gaussian distribution)
```

Time domain



Spectral domain



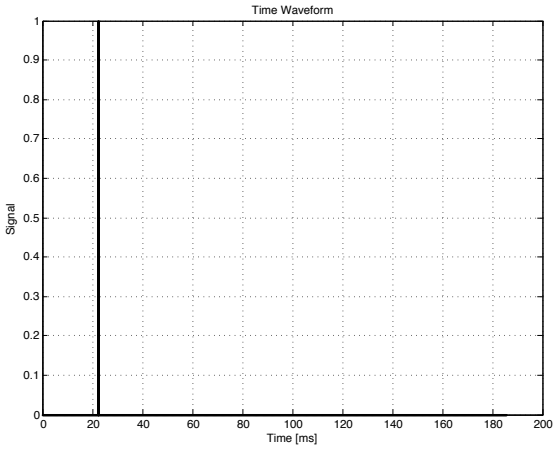
- Magnitude is flat just like an impulse (i.e., flat), but the phase is random

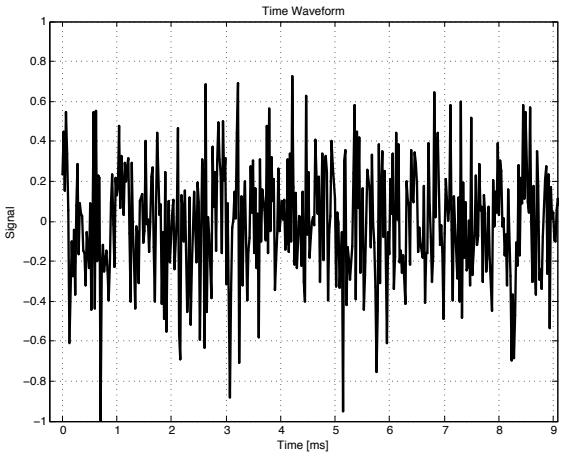
Fourier transforms of basic (1-D) waveforms

Impulse

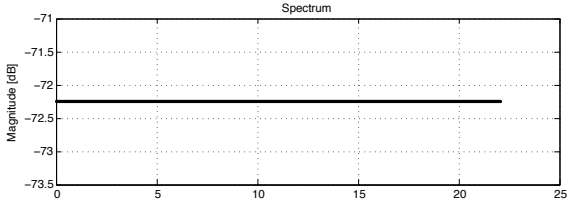
Noise

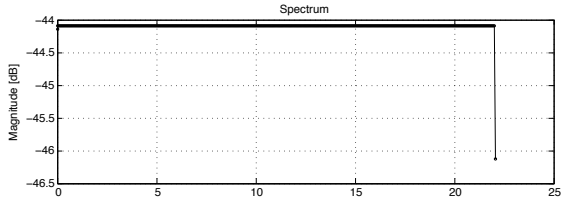
Time domain

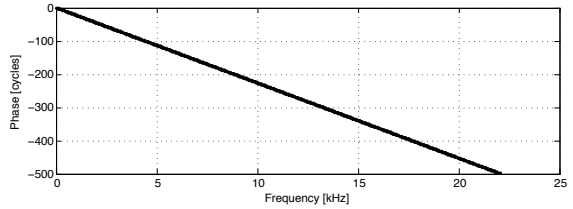


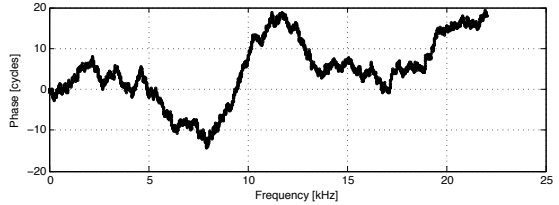


Spectral domain

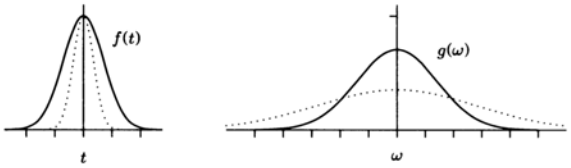








→ Remarkable that the magnitudes are identical (more or less) between two signals with such different properties. The key difference here is the phase: *Timing is a critical piece of the puzzle!*

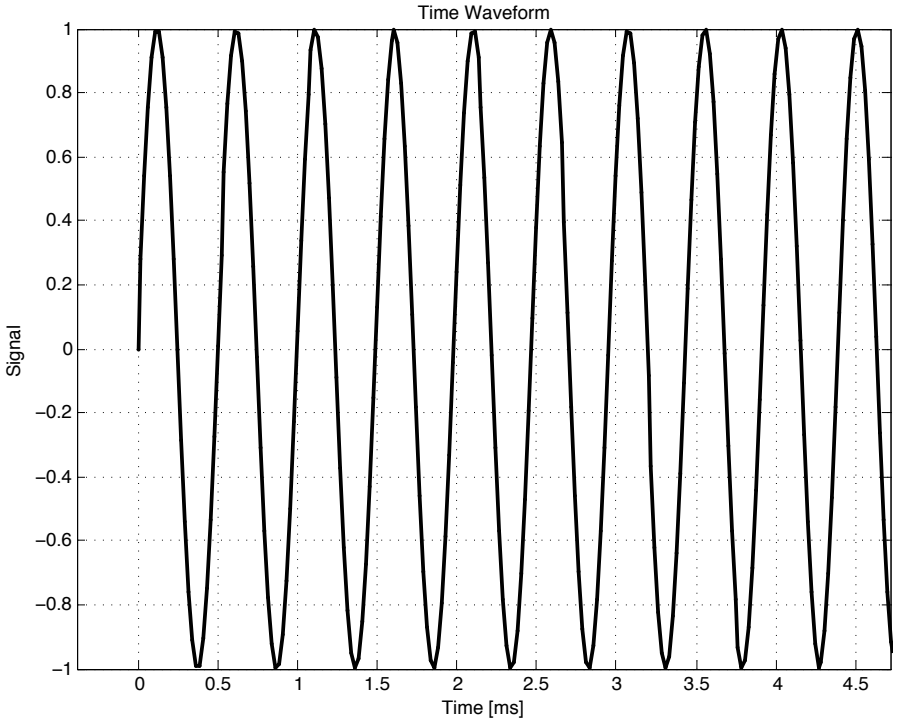


Fourier transforms of basic (1-D) waveforms

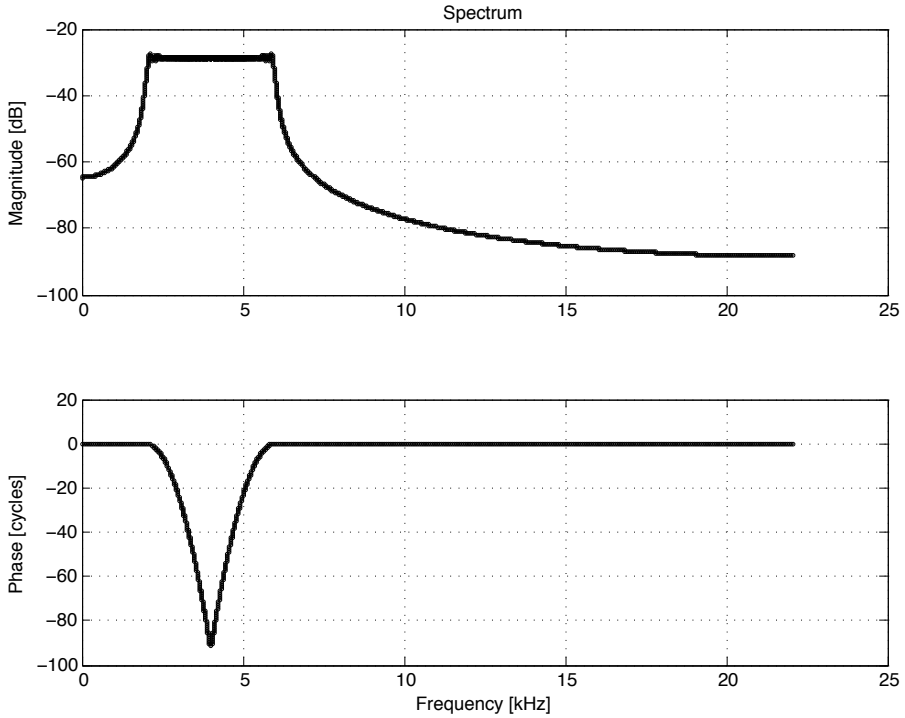
EXspecREP3.m

```
stimT= 6 - chirp (flat mag.)
```

Time domain



Spectral domain



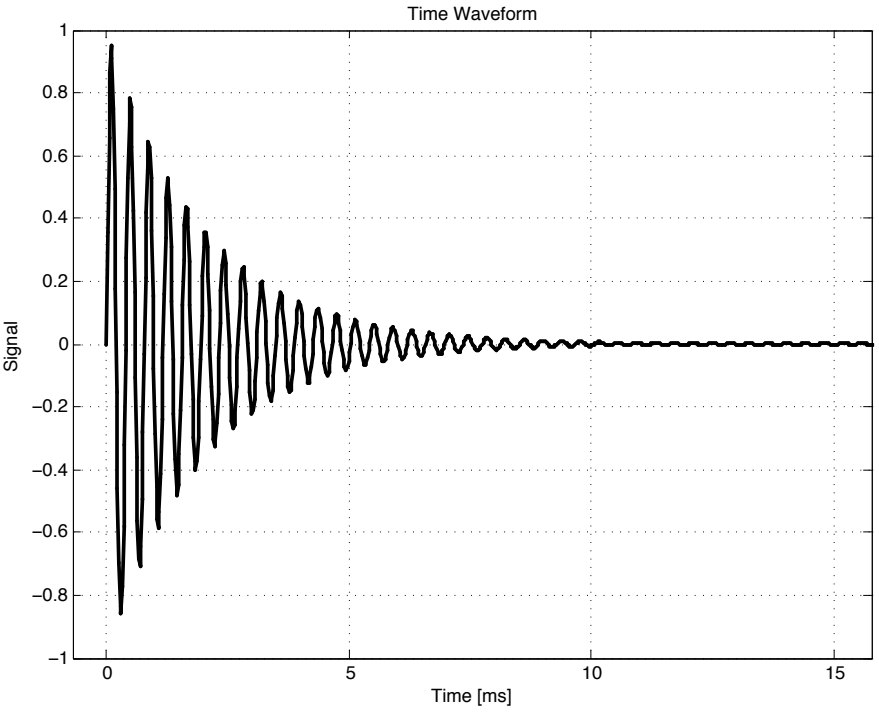
Hard to see on this timescale, but frequency is changing (increasing) with time

Fourier transforms of basic (1-D) waveforms

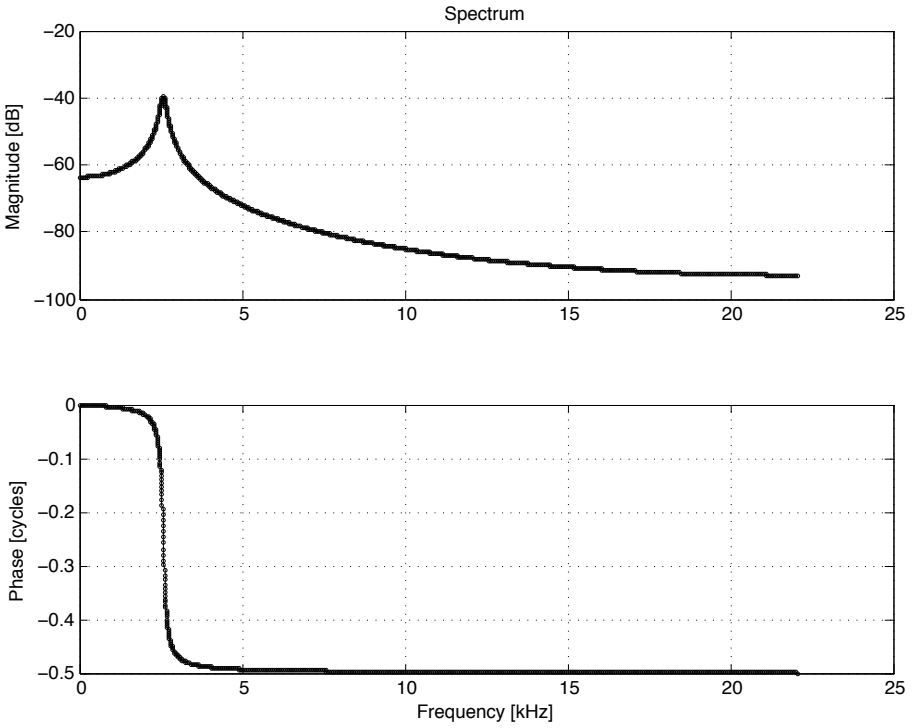
EXspecREP3.m

```
stimT= 8 - exponentially decaying sinusoid
```

Time domain

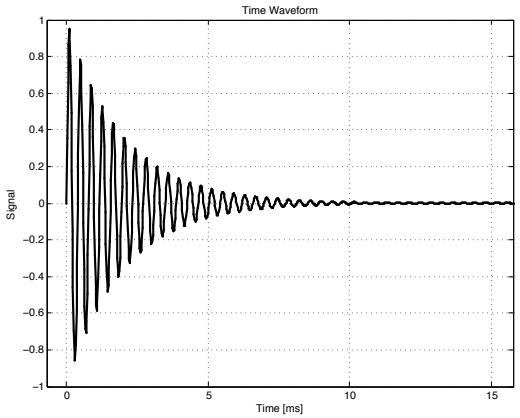


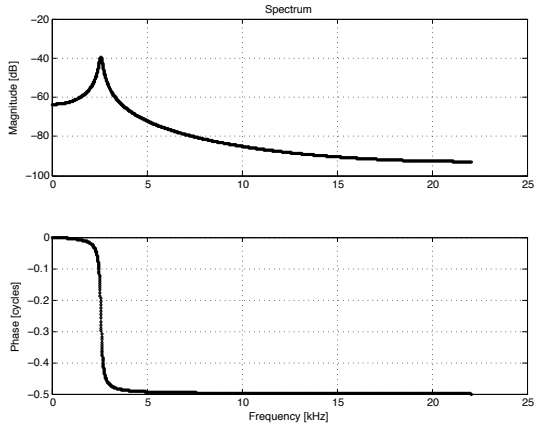
Spectral domain

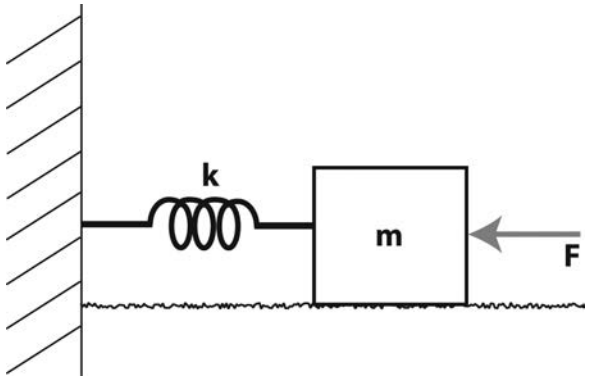


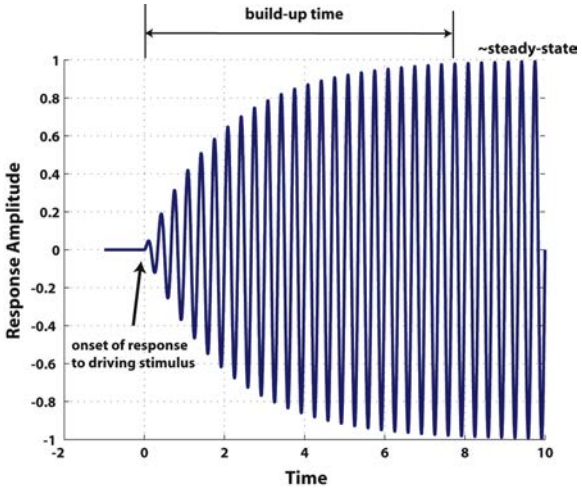
- This seems to look familiar....

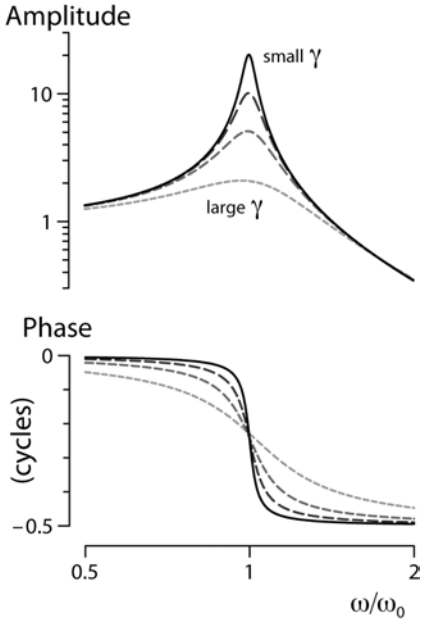
Connection back to.... ... the harmonic oscillator!







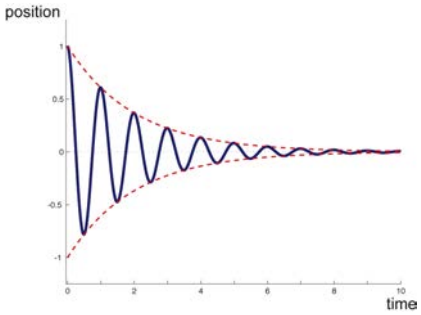




- The *steady-state* response of the sinusoidally-driven harmonic harmonic oscillator acts like a ***band-pass filter***

- Distinction between *steady-state* response & ***impulse response***

  [we'll come back to this]



Post-class exercises

- Write a simple Matlab script that creates two sinusoids, adds them together, then visualizes the resulting waveform. Explore how does the 'beating pattern' depends upon the relative frequencies, phases, and/or amplitudes.

- Similar to above, try adding three sinusoids together and observe the variety of effects you can get by changing the relative properties.

- Fiddle around with `EXbuildImpulse.m` and `EXspecREP3.m` to get a feel for how different time waveforms have different spectral representations and vice versa

- Using `EXbuildImpulse.m`, determine the relationship between the impulse, the phase, and the 'group delay'

- Consider the difference between `fft.m` & `rfft.m`, and consider how such relates to the bottom spectral plot shown here