The University of Texas at Austin

Spring 2025 EE 445S Real-Time Digital Signal Processing Laboratory Prof. Evans

Solutions for Homework #1 on Sinusoids, Transforms and Transfer Functions

1. Transfer Functions. 48 points.

With $x[n]$ denoting the input signal and $y[n]$ denoting the output signal, give the difference equation relating the input signal to the output signal in the discrete-time domain, give the initial conditions and their values, and find the transfer function in the z-domain and the associated region of convergence for the z-transform function, for the following linear time-invariant discrete-time systems.

Prolog: Note that (a) and (b) are finite impulse response (FIR) filters, and (c) and (d) are infinite impulse response (IIR) filters. More on filters in lectures 5 & 6, and lab 3. A necessary condition for a system to be LTI is that it be "at rest" (i.e. all the initial conditions must be zero). An LTI system is uniquely defined by its impulse response. The z-transform of the impulse response is a way to compute the transfer function $H(z)$ in the z-domain for the LTI system. In the z-domain, $Y(z) = H(z)\,X(z)$ where $Y(z)$ is the z-transform of the output signal and $X(z)$ is z-transform of the input signal. So, another way to compute the z-domain transfer function of the LTI system is $H(z) = Y(z) / X(z)$.

(a) Causal averaging filter with five coefficients. See Designing Averaging Filter Handout.

Output is the running average of the current and 4 previous input values. The difference equation is

$$y[n] = \frac{1}{5}x[n] + \frac{1}{5}x[n-1] + \frac{1}{5}x[n-2] + \frac{1}{5}x[n-3] + \frac{1}{5}x[n-4]$$

We find the initial conditions by first letting n=0: $y[0] = (1/5)\,(x[0] + x[-1] + x[-2] + x[-3] + x[-4])$. Then, we let n = 1, 2, … until no initial conditions appear. Using this approach, initial conditions are $x[-1]$, $x[-2]$, $x[-3]$ and $x[-4]$. They must be zero as a necessary condition for the system to be linear and time-invariant (LTI). To find the transfer function in the z-domain, we take the z-transform of both sides of the difference equation with the initial conditions being zero and then isolate $Y(z)/X(z)$:

$$Y(z) = \frac{1}{5}X(z) + \frac{1}{5}z^{-1}X(z) + \frac{1}{5}z^{-2}X(z) + \frac{1}{5}z^{-3}X(z) + \frac{1}{5}z^{-4}X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{5}(1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}) = \frac{1}{5}\left(\frac{z^5 + z^4 + z^3 + z^2 + z}{z^5}\right)$$

Rewriting the above transfer function shows that there are 5 repeated poles at the origin. The Region of Convergence (ROC) does not include the poles because at a pole location, the transfer function would be infinite. The ROC is the entire complex z-plane excluding the origin, i.e. $z \neq 0$.

(b) Causal discrete-time approximation to first-order differentiator. See lecture slide 3-24.

The first-order derivative operation with output $y(t)$ and input $x(t)$ can be defined in terms of a limit:

$$y(t) = x'(t) = \lim_{\Delta t \to 0} \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

After sampling the input and output signals, the smallest separation between time-domain samples is the sampling time T_s so that $\Delta t = T_s$. However, we cannot drive T_s to zero in practice:

$$y[n] \approx \frac{1}{T_s}(x[n] - x[n-1])$$

Due to sampling, the approximation is only valid for continuous-time frequencies up to one-half of the sampling rate in value. From the sampling theorem, $f_s > 2 f_{max}$, or equivalently $f_{max} < ½ f_s$.

In discrete time, we abstract away sampling rate/time when possible. The difference equation becomes

$$y[n] = x[n] - x[n-1]$$

The initial condition $x[-1] = 0$ for the system to be LTI. That is, an LTI system must be at rest.

Transfer function: $H(z) = \frac{Y(z)}{X(z)} = 1 - z^{-1} = \frac{z-1}{z}$

Region of convergence (ROC): $z \neq 0$. More generally, the ROC for a finite impulse response filter is the entire complex z-plane except the origin.

(c) Causal discrete-time approximation to first-order integrator. See homework hints.

In continuous time, the output of a causal first-order integrator is defined by

$$y(t) = \int_0^t x(u)du$$

for $t \geq 0$ where $x(t)$ is the input. The discrete-time version obtained from sampling is

$$y[n] = \sum_{m=0}^{n} x[m]$$

This is inefficient because it requires an unbounded amount of memory to store the previous input values as $n \to \infty$. Instead, we can use a recursive difference equation, a.k.a. a running summation,

$$y[n] = y[n-1] + x[n]$$

for $n \geq 0$ with initial condition $y[-1] = 0$ as a necessary condition for LTI to hold. An LTI system must be "at rest". The transfer function: $H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - z^{-1}} = \frac{z}{z-1}$

The above $H(z)$ is a special case of the z-transform of the causal signal $v[n] = a^n u[n]$ which follows:

$$V(z) = \sum_{n=-\infty}^{\infty} v[n]\, z^{-n} = \sum_{n=-\infty}^{\infty} a^n\, u[n]\, z^{-n} = \sum_{n=0}^{\infty} a^n\, z^{-n} = \sum_{n=0}^{\infty} \left(\frac{a}{z}\right)^n = \frac{1}{1 - a\, z^{-1}} \; if \; \left|\frac{a}{z}\right| < 1$$

We can rewrite the condition $\left|\frac{a}{z}\right| < 1$ as $|z| > |a|$ provided that $z \neq 0$. The condition $|z| > |a|$ is the region of convergence, and represents the area of the complex z plane outside of disk of radius $|a|$.

$H(z)$ is the same as $V(z)$ if $a = 1$. The ROC for $H(z)$ is $|z| > 1$. The z-transform of a causal signal will be the area of the complex z plane outside of disk whose radius is equal to the largest radius among the poles. $H(z)$ has a zero at the origin ($z = 0$) and a pole at $z = 1$.

(d) Causal bandpass filter with center frequency ω_0 given by the input-output relationship

$$y[\mathrm{n}] = (2\ \cos \omega_0)\, r\, y[n-1] - r^2\, y[n-2] + x[n] - (\cos \omega_0)\, x[n-\mathrm{l}]$$

where $0 < r < 1$. Here, r is the radius of the two pole locations. (Note: r has a constant value.)

The initial conditions $y[-1]$, $y[-2]$ and $x[-1]$ must be set to zero as a necessary condition for LTI to hold.

Taking the z-transform of both sides of the difference equation, we get

$$Y(z) = (2 \cos \omega_0)\, r\ z^{-1}\, Y(z) - r^2\, z^{-2}\, Y(z) + X(z) - (\cos \omega_0)\, z^{-1}\, X(z)$$

$$\Rightarrow \frac{Y(z)}{X(z)} = \frac{1 - \cos \omega_0 z^{-1}}{1 - 2(\cos \omega_0) r z^{-1} + r^2 z^{-2}} \Rightarrow H(z) = \frac{1 - \cos \omega_0 z^{-1}}{1 - 2(\cos \omega_0) r z^{-1} + r^2 z^{-2}}$$

We need to find the poles of this transfer function by finding the roots of the denominator as follows:

$$1 - (2 \cos \omega_0)\, r\, z^{-1} + r^2 z^{-2} = 0$$

By multiplying each side by z^2 (assuming that $z \neq 0$):

$$z^2 - (2 \cos \omega_0)\, r\, z + r^2 = 0$$

Roots are located at ½ (-b ± sqrt (Δ)). Here,

$$\Delta = 4r^2 \cos^2 \omega_0 - 4r^2 = -4r^2 \sin^2 \omega_0$$

Since $\Delta < 0$, there are complex roots at $r \exp(j\, \omega_0)$ and $r \exp(-j\, \omega_0)$:

$$x_1 = \frac{2r \cos \omega_0 + 2rj \sin \omega_0}{2} = r(\cos \omega_0 + j \sin \omega_0) = re^{j\omega_0}$$

$$x_2 = \frac{2r \cos \omega_0 - 2rj \sin \omega_0}{2} = r(\cos \omega_0 - j \sin \omega_0) = re^{-j\omega_0}$$

Poles x_1 and x_2 have magnitude r. For a causal system, the region of convergence will be outside of a disk of radius equal to the magnitude of the pole with the greatest magnitude, i.e. $|z| > r$ in this case.

We can see this by considering the z-transform of the causal signal $v[n] = a^n\, u[n]$:

$$V(z) = \sum_{n=-\infty}^{\infty} v[n]\, z^{-n} = \sum_{n=-\infty}^{\infty} a^n\, u[n]\, z^{-n} = \sum_{n=0}^{\infty} a^n\, z^{-n} = \sum_{n=0}^{\infty} \left(\frac{a}{z}\right)^n = \frac{1}{1 - az^{-1}} \; if \; \left|\frac{a}{z}\right| < 1$$

We can rewrite the condition $\left|\frac{a}{z}\right| < 1$ as $|z| > |a|$ provided that $z \neq 0$. The condition $|z| > |a|$ is the region of convergence, and represents the area of the complex z plane outside of disk of radius $|a|$.

Consider the z-transform of $y[n] = v[n] + w[n]$ which is $Y(z) = V(z) + W(z)$. The region of convergence for Y(z) is the set of all z values that are valid: ROC{ $Y(z)$ } = ROC{ $V(z)$ } ∩ ROC{ $W(z)$ }. For $v[n] = a^n\, u[n]$ and $w[n] = b^n\, u[n]$, ROC{ $Y(z)$ } = { $|z| > |a|$ } ∩ { $|z| > |b|$ } = $|z| > \max\{ |a|, |b| \}$.

2. Chirp Signals. 20 points.

Before starting this problem, please read slides 1-14 to 1-20 on chirp signals and spectrograms (which also contain a lot of examples of Matlab code) in the Common Signals in Matlab slide deck.

A chirp signal is used in sonar and radar systems, indoor positioning, and test/measurement. A chirp signal is a sinusoid whose principal frequency increases (or decreases) over time:

$$c(t) = \cos(\theta(t)) \text{ where } \theta(t) = 2\pi\,(f_0 + \tfrac{1}{2} f_{step}\, t)\, t = 2\pi f_0\, t + \pi f_{step}\, t^2$$

The principal frequency is f_0 when $t = 0$ and then changes at a rate of f_{step} for each second that passes. The frequency of a sinusoid at a given point in time is called the instantaneous frequency, and it is defined as $d\theta(t) / dt$ in units of rad/s. Here, $d\theta(t) / dt = 2\pi f_0 + 2\pi f_{step}\, t = 2\pi (f_0 + f_{step}\, t)$.

(a) Generate a chirp signal for 10s with f_0 = 20 Hz and f_{step} = 420 Hz/s. Pick a sampling rate f_s of 44100 Hz. The chirp will sweep through the frequencies of the keys on an 88-key piano.

(b) Plot chirp signal in time and frequency domains using `plotspec` from the JSK book.

(c) Play the chirp signal as an audio signal. You might consider using headphones so as not to bother folks around you. Provide the Matlab code. Describe what you hear.

(d) Plot the chirp signal in the time-frequency domain using the spectrogram function in Matlab and describe the visual representation.

Each part is worth 5 points. This problem is inspired by JSK, Exercise 3.8 on page 46

Prolog: Pipestrelle bats use chirps that sweep down from 70 to 45 kHz for echolocation: https://www.wildlife-sound.org/resources/equipment?view=article&id=233:recordings-of-ultrasonic-vocalisations-of-bats&catid=2

Active sonar systems transmit a "ping" chirp sound, and the delay in receiving the chirp indicates the time for the chirp to bounce off an object and return, which can be converted to a distance.

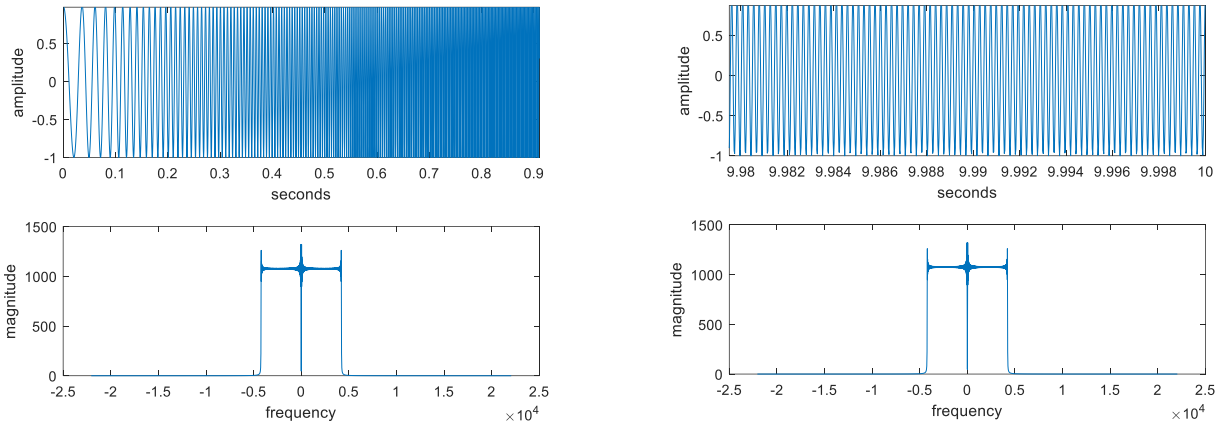
When measuring a system frequency response, one could input a sinusoid, measure the output, and repeat using many frequencies. Inputting a chirp allows the measurement in one take.

Cellular LTE systems send a Zadoff-Chu chirp to measure distortion from transmitter to receiver.

Solution: a) MATLAB code:

```
time = 10;                  % length of time in seconds
fs = 44100;                 % sampling rate
Ts = 1/fs;                  % sampling time: time interval between samples
t = Ts : Ts : time ;        % create a time vector
f0 = 20;                    % specify starting principal frequency
fstep = 420;                % specify frequency slope
phi = pi*fstep*t.^2;        % specify phase
x=cos(2*pi*f0*t + phi);     % create chirp waveform
plotspec(x, Ts)             % plot waveform in time domain and its spectrum
```

b) Plots: Zoomed in (0s to 0.9s) Zoomed in (9.98s to 10.0s)



Center frequency is about 2110 Hz and bandwidth is approximated at 4200 Hz by zooming in.

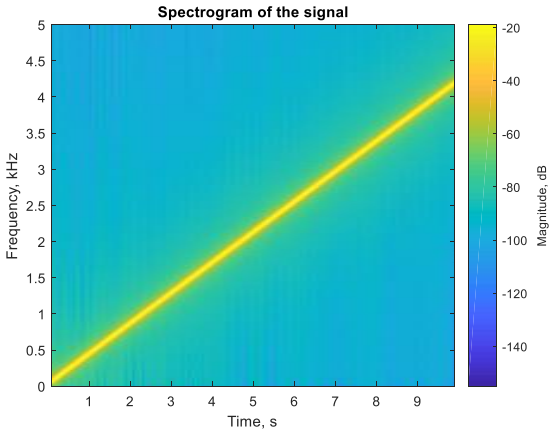
The **instantaneous frequency** is $2\pi f_0 + 2\pi f_{step}\, t = 2\pi (f_0 + f_{step}\, t)$. As t goes from 0s to 10s, the instantaneous frequency increases from 20 Hz to 4220 Hz, with an average value of 2110 Hz.

c) The chirp signal can be played as an audio signal by using the following MATLAB code:

```
sound(x, fs);           % play back chirp signal
```

This chirp linearly sweeps frequencies 20 to 4220 Hz. On the Western music scale, 'A' notes are at 27.5, 55, 110, 220, 440, 880, 1760 and 3520 Hz. https://en.wikipedia.org/wiki/Piano_key_frequencies. When you play the chirp signal on a laptop or tablet, whether it is over their speakers or through headphones, you might not hear frequencies below 200 Hz. To hear those frequencies, the playback system would need to be able to convert low frequencies in the electrical signal into large wavelength pressure waves via piezoelectric devices. The corresponding size of these devices does not fit in laptops and tables. In a conventional audio system, a sub-woofer would handle these low frequencies.

d) A spectrogram displays frequency content as it evolves over time. The color at any point shows the magnitude of the frequency component at that point in time. Contrast that with a Fourier transform that shows average frequency content in a signal without knowing when frequencies occurred. A spectrogram is an aerial view of a 3-D time-frequency-magnitude plot.



The spectrogram for the chirp signal shows that the principal frequency is changing over time with a linear slope. At the beginning, the principal frequency is at 20 Hz and increases linearly to 4220 Hz. The principal frequency has the highest magnitude at every instant of time throughout the entire duration of the chirp signal. Since we are observing over a finite length of time, the chirp has a wider bandwidth than a two-sided chirp (recall homework problem 0.1). This bandwidth is visible as the width of the yellow line and the reddish area that falls on either side of the principal frequency line.

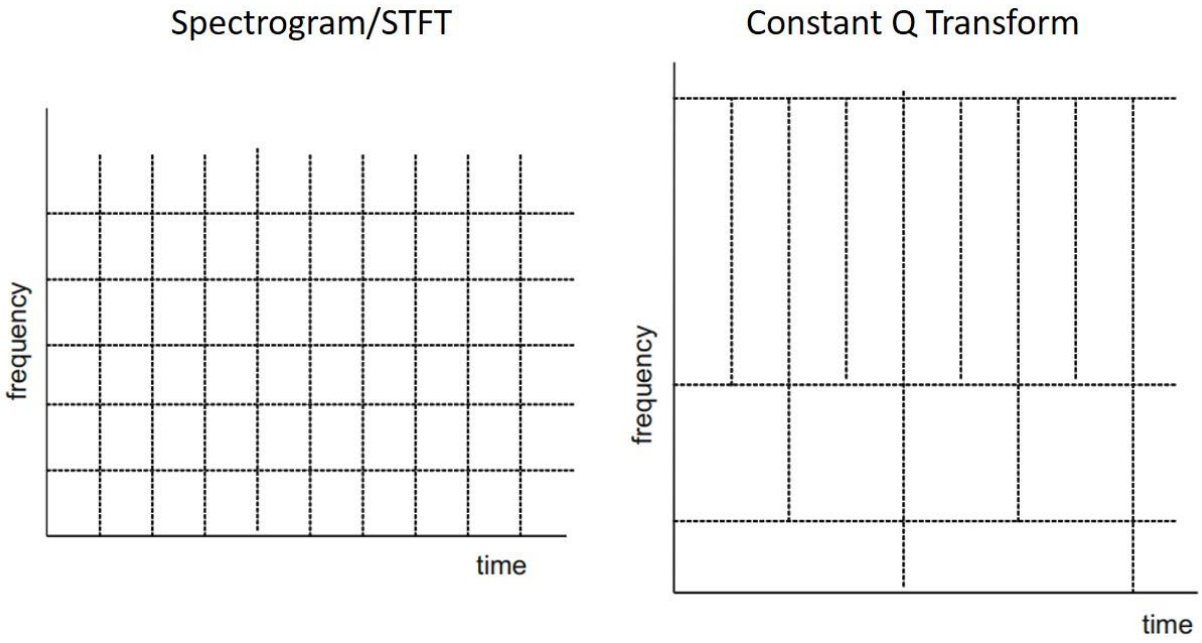
A spectrogram takes the first N_{win} samples of the signal, weights the values (by a rectangular pulse by default), applies the fast Fourier transform, and plots the magnitude of the FFT output. The spectrogram then shifts the time signal to the right and repeats the previous steps using a block of the N_{win} samples. The frequency resolution of the spectrogram is f_s / N_{win}. On homework problem 0.1, the frequency resolution of observing a signal for T seconds is proportional to $1/T$. **There is a tradeoff between frequency resolution f_s / N_{win} and time resolution $N_{win} T_s$.** To obtain 20 Hz of frequency resolution, one would have to observe the signal for 0.05s. At a sampling rate of 44100 samples/s, 0.05s would mean (44100 samples/s)(0.05s) = 2205 samples. Using 10240 samples would give a frequency resolution of 4.3 Hz. The amount of samples in one block that overlap with the previous block is given by $N_{overlap}$, so the shift is by N_{win} - $N_{overlap}$ samples. Weighting by the Hamming pulse is a common alternative to the rectangular pulse: replace `nwin` with `hamming(nwin)`.

```
nwin = 10240;           % divide chirp signal into block of nwin samples
noverlap = 3/4*10240;   % number of samples in each block of chirp signal
                        % that overlaps with the previous block
nfft = [];              % specifies the number of frequency points used to
                        % calculate the discrete Fourier transforms.
figure; spectrogram(x, nwin, noverlap, nfft, fs, 'yaxis');
h = colorbar;           % set the colorbar(dB) in y axis
ylabel(h, 'Magnitude, dB'); ylabel('Frequency, kHz');
ylim([0,5]); xlabel('Time, s'); title('Spectrogram of the signal');
```

Epilog: [By Dan Jacobellis] In this problem, a spectrogram was used to visualize a chirp signal by displaying the signal as a function of both time and frequency. The spectrogram is constructed by taking the DFT of a sliding time window, resulting in a uniform resolution in both time and frequency.

In many applications, especially audio and music processing, a logarithmic resolution in frequency is preferred, either to match the human auditory response or the match a musical scale

As we increase the window size of the spectrogram, we will increase our frequency resolution at the expense of time resolution. In other words, the 'Area' of a pixel in the spectrogram can only be so small. However, we can change the 'shape' of each pixel, so that the frequency resolution increases at low frequencies and the time resolution increases at high frequencies, so that the important details of audio signals become clear.



STFT means the short-time Fourier transform. The spectrogram is an implementation of it.

An efficient transform that achieves logarithmic frequency resolution is called the Constant Q Transform. The 'Q' refers to quality factor, which is the ratio of the bandwidth to the center frequency of a filter. The transform can be thought of as a bank of filters whose bandwidths increase linearly with frequency. Implementations in MATLAB[1] and Python [2] are available.

[1] https://www.mathworks.com/help/wavelet/ref/cqt.html

[2] https://librosa.github.io/librosa/generated/librosa.core.cqt.html

3. **Spectral Analysis for a Squaring Block. 30 points.**

A squaring block can be used for downconversion of an upconverted signal to a baseband signal. In that case, the squaring block would be followed by a lowpass filter, as mentioned on lecture slide 3-7.

JSK Section 3.5 on page 52 analyzes a squaring block for an input of a two-sided sinusoid at constant frequency. At the output, half of the input power appears at DC (zero frequency) and the other half appears at twice the input frequency. The output has a DC offset. If the input had two frequencies f_1 and f_2, the output will have frequencies 0, $|f_1 - f_2|$, 2 f_1, $f_1 + f_2$, and 2 f_2. Again, there is a DC offset. The DC offset can be an issue when the output is an integer, or when floating-point values are converted to an integer, because the DC offset would take up many of the bits in the integer representation, thereby leaving very few bits to represent the rest of the signal. The DC offset can also be an issue during playback. For example, the Matlab `sound` command clips amplitudes outside the range [-1, 1].

This problem uses the gong waveform that accompanies the JSK book as the message signal, and uses the upconvertGong.m file for upconversion and downconversion:

http://users.ece.utexas.edu/~bevans/courses/realtime/lectures/00_Introduction/upconvertGong.m

In order for the squaring approach to work for downconversion, a DC offset has to be added to the message signal in the transmitter. (This is how AM radio works.) In upconvertGong.m, please replace

```
modulated = basebandInput .* carrier;
```

with

```
modulated = (1 + basebandInput) .* carrier;
```

(a) In the upconvertGong.m file, please add code in the receiver to remove the DC offset that was inserted in the transmitter. The Matlab function `mean` will compute the DC component (average value) of a vector. *8 points*
   i. Compute the mean squared error between the original gong waveform and the downconverted gong waveform with the DC offset removed in upconvertGong.m.
   ii. Describe any differences that you hear between the two versions of the gong waveform.

(b) In the upconvertGong.m file, replace the downconversion code with a squaring device followed by a lowpass filter. You will need to change the design of the lowpass filter. *12 points*.
   i. Compute the mean squared error between the original gong waveform and the downconverted gong waveform for upconvertGong.m that you had modified.
   ii. Describe any differences that you hear between the two versions of the gong waveform.
   iii. What happened to the principal frequencies in the gong signal after squaring the upconverted signal? You might try using the `plotspec` command from the JSK book.

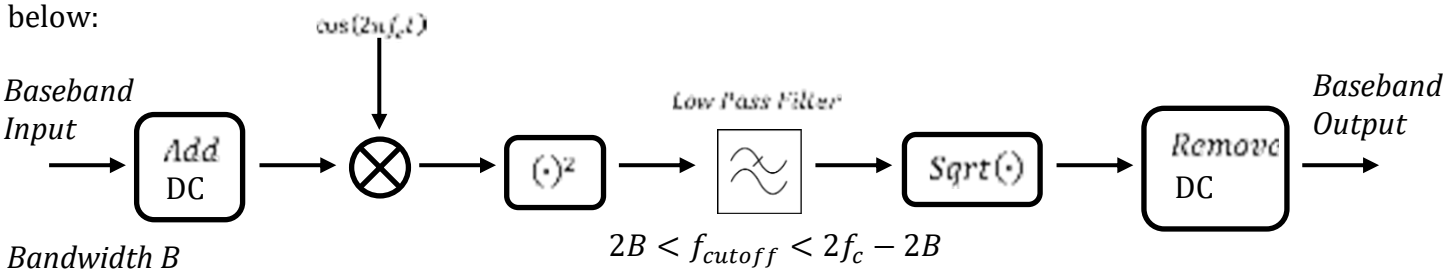
(c) The squaring block distorts the amplitude of the input signal. To your solution in part (b), add a square root operation after the lowpass filter. *8 points*.

i. Compute the mean squared error between the original gong waveform and the downconverted gong waveform for upconvertGong.m that you had modified.
ii. Describe any differences that you hear between the two versions of the gong waveform.

(d) After the square root block, remove the DC (average) value. *3 points.*

i. Compute the mean squared error between the original gong waveform and the downconverted gong waveform for upconvertGong.m that you had modified.

The transmitter and alternate receiver for (b), (c) and (d) are described in the block diagram below:



This problem is inspired by JSK, Exercise 3.19 on page 52.

Solution: (a) Mean squared error (MSE) is a measure of signal quality computed on two signals of interest, i.e. the original gong signal $x[n]$ and the gong signal obtained from upconversion and then downconversion $y[n]$ in this problem. MSE between $x[n]$ and $y[n]$ is defined as $\sum_{n=0}^{M-1}(x[n]-y[n])^2$ where M is the number of samples in both signals. There is a related normalized mean-squared error (NMSE): $\frac{1}{M}\sum_{n=0}^{M-1}(x[n]-y[n])^2$. For example, the NMSE for discrete-time signals $x_1[n]$ and $y_1[n]$ of length M_1 samples and $x_2[n]$ and $y_2[n]$ of length M_2 samples can be directly compared. MSE is related to the signal-to-noise ratio: $SNR = \frac{Signal\ Power}{Noise\ Power} = \frac{\sum_{n=0}^{M-1} x^2[n]}{\sum_{n=0}^{M-1}(x[n]-y[n])^2} = \frac{\frac{1}{M}\sum_{n=0}^{M-1} x^2[n]}{\frac{1}{M}\sum_{n=0}^{M-1}(x[n]-y[n])^2}$

In part (a), one should account for the delay in the transmitter and receiver to compute the mean squared error between the message signal before upconversion at the transmitter (basebandInput) and the message signal downconverted at the receiver (basebandOutput). The only delay is in the filter.

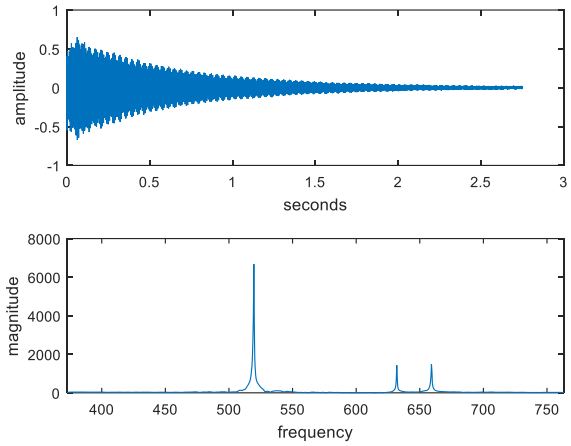
Group delay through a finite impulse response (FIR) filter whose N coefficients are even symmetric about its midpoint is (N-1)/2. That is, the FIR filter would take (N-1)/2 samples to fully respond to the input signal. The first (N-1)/2 samples of the downconverted gong waveform are discarded. To keep the gong signals the same length, the last (N-1)/2 samples of the original gong waveform are also discarded. (We will derive the group delay for FIR filters in lecture 5.) Changes in code are in **bold**:

```
[basebandInput, fs] = audioread('gong.wav');
basebandInput = basebandInput';
f1 = 660;

Ts = 1/fs;                              %%% Sampling time
numSamples = length(basebandInput);
n = 1 : numSamples;
t = n * Ts;
tmax = Ts * numSamples;

fc = 4*f1;                              %%% Upconvert
carrier = cos(2*pi*fc*t);               %%% to be centered at frequency fc
modulated = (1 + basebandInput) .* carrier;

carrier = cos(2*pi*fc*t);
modulateAgain = modulated .* carrier;

FIRlength = floor(fs/f1);               %%% Use an odd-length FIR filter
if 2*floor(FIRlength/2) == FIRlength
  FIRlength = FIRlength - 1;
end

lowpassCoeffs = ones(1, FIRlength) / FIRlength;
basebandOutput = 2*filter(lowpassCoeffs, 1, modulateAgain);
basebandOutput = basebandOutput - mean(basebandOutput);

delay = (FIRlength-1)/2;
diff = basebandInput(1:numSamples-delay) - basebandOutput(delay+1:numSamples);
meanSquaredError = sum(diff .^ 2)

figure; plotspec(basebandInput, 1/fs);
sound(basebandInput, fs);
pause(tmax+1);
figure; plotspec(modulated, 1/fs);
% sound(modulated, fs);
% pause(tmax+1);
figure; plotspec(basebandOutput, 1/fs);
sound(basebandOutput, fs);
pause(tmax+1);
```

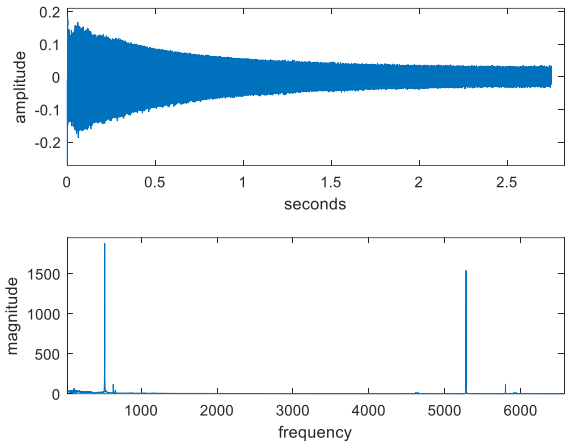
The Matlab command `sum` adds up the elements of a vector. **The mean squared error is about 1075 and the normalized mean squared error is 0.00886.**

The original gong signal has principal frequencies at 520, 630, and 660 Hz. The carrier frequency is 2640 Hz. The downconverted gong waveform has a principal frequency at 520 Hz. Due to the lowpass filter, the downconverted gong waveform has small peaks at 630 and 660 Hz, a very strong peak at 5280 Hz (twice the carrier frequency) and small peaks at 4760 Hz and 5800 Hz (twice the carrier frequency plus/minus 520 Hz). The 520 Hz frequency component is about one-fourth of that of the original gong waveform.

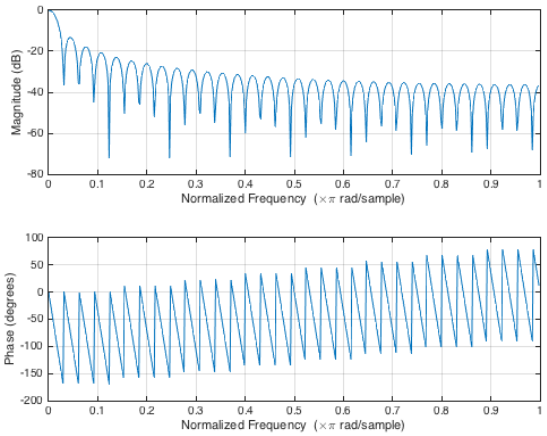
Original gong waveform



Downconverted gong waveform



In the downconversion stage, the lowpass finite impulse response (FIR) filter is an averaging filter of N coefficients. Its impulse response is a rectangular pulse, which is periodic sinc function in the Fourier domain. Per the handout on Designing Averaging Filters, the null bandwidth is $\omega_0 = 2\,\pi\,/\,N$ in rad/sample. We can convert the discrete-time frequency ω_0 to a continuous-time frequency f_0 via $\omega_0 = 2\,\pi\,f_0\,/\,f_s = 2\,\pi\,/\,N$ and solve for $N = f_s\,/\,f_0 = 66$ and $f_0 = f_s\,/\,N = 668.18$ Hz. This filter also zeros out multiples of 668.18 Hz. The amount of attenuation in the magnitude response increases with frequency as shown by `freqz(lowpassCoeffs)`.



b) A **squaring block** followed by a lowpass filter can realize downconversion. An advantage is the receiver would not need to know the exact carrier frequency to perform downconversion.

In the squaring approach, the baseband bandwidth of the demodulating filter is doubled. The doubling is due to the fact that squaring the modulated signal in the time domain becomes convolution of the modulated signal's frequency content with itself. The FIR filter length is half that in part (a) since the null bandwith is inversely proportional to the length. See the handout on Designing Averaging Filters.

A gain of 2 is used for sinusoidal demodulation. In trying out different gain values, one gets an MSE of 133 for a gain of 2 and 39.3 for a gain of 3 and 43.1 for a gain of 4. In terms of MSE, a filter gain of 3 is a better choice than a filter gain of 2 or 4. Keeping the filter gain at 2 is also fine.

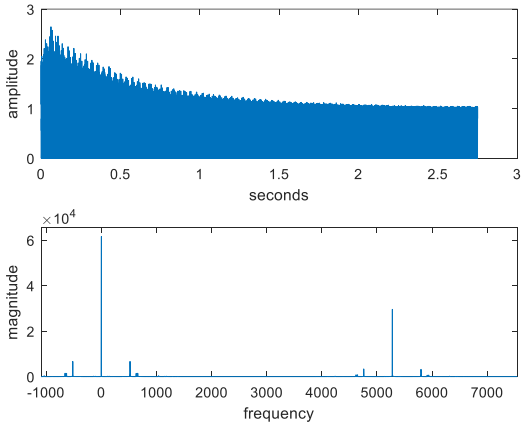
```
% part (b) Squaring Operation and Low-Pass Filtering
modulateAgain = modulated .^ 2;
figure; plotspec(modulateAgain, 1/fs);

FIRlength = floor(fs/(2*f1));       %%% Use an odd-length FIR filter
if 2*floor(FIRlength/2) == FIRlength
  FIRlength = FIRlength - 1;
end

lowpassCoeffs = ones(1, FIRlength) / FIRlength;
basebandOutput = 2*filter(lowpassCoeffs, 1, modulateAgain);

% plotting
figure; plotspec(basebandOutput, 1/fs);
sound(basebandOutput, fs);
pause(tmax+1);
```

The mean squared error is about 125660 and the normalized mean squared error is 1.0361.

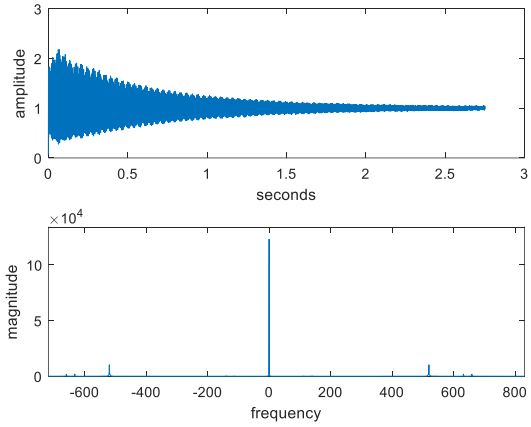
The original gong signal has principal frequencies at 520, 630 and 660 Hz. We added a DC component at baseband. The carrier frequency is 2640 Hz. So the upconverted signal has principal frequencies of 2640 Hz +/- 0, 520, 630, and 660 Hz, which gives 1980, 2010, 2120, 2640, 3160, 3270, and 3300 Hz.

The following two plots are the `modulateAgain` signal which results from squaring the upconverted signal and its filtered version. We only show the one-sided frequency spectrum for `modulateAgain` signal. We compare the filtered version of `modulateAgain` signal side by side. There is a noticeable DC component in the filtered `modulateAgain` signal.

`modulateAgain` waveform



Lowpass Filtered `modulateAgain` waveform



Here are the principal non-negative frequency components in the `modulateAgain` signal:

- 0 and 5280 Hz, which are the largest components by far

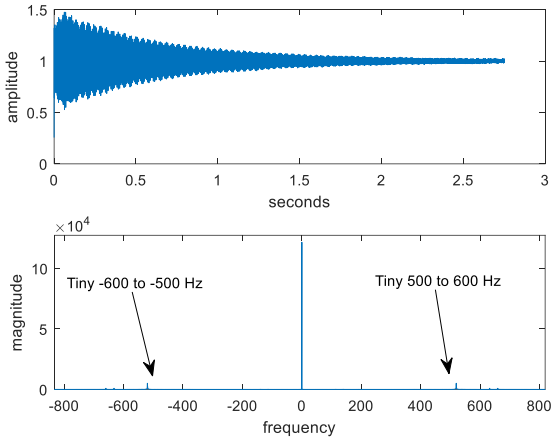
- 520, 630 and 660 Hz
- 4620, 4650 and 4760 Hz, which are 5280 Hz - 520, 630, and 660 Hz
- 5800, 5910, and 5940 Hz, which are 5280 Hz + 520, 630, and 660 Hz

With sampling rate 44100 Hz, we can represent frequency component up to 22050 Hz.

c) The square root operation is performed after the lowpass filtering operation but before the plotting..

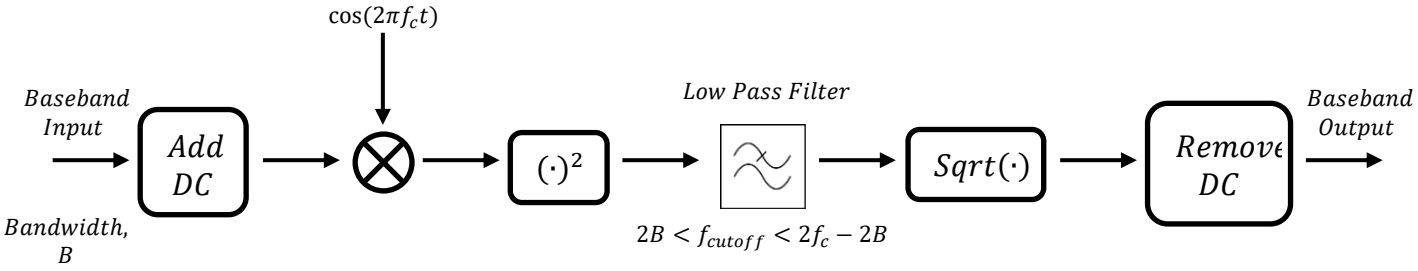
```
% part (c) Square-root Operation
basebandOutput = basebandOutput.^(0.5);
```

The mean square error is 122,194 and the normalized mean squared error is 1.0075. The square root operation increases the energy of the zero frequency component.



The DC offset of approximately 1 is visible in the time domain plot at the top. The frequency plot has a very large DC component at 0 Hz. When zoomed in, we will see other principal positive frequencies of 520, 630 and 660 Hz (not shown).

d) Here is the block diagram for the transmitted and the alternate receiver using the squaring block.

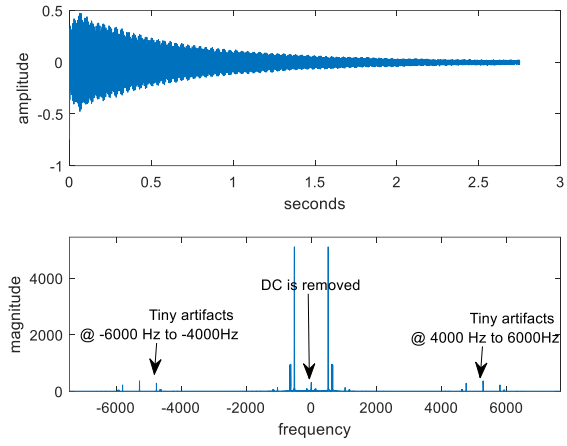


The DC component is removed after the square root operation but before plotting.

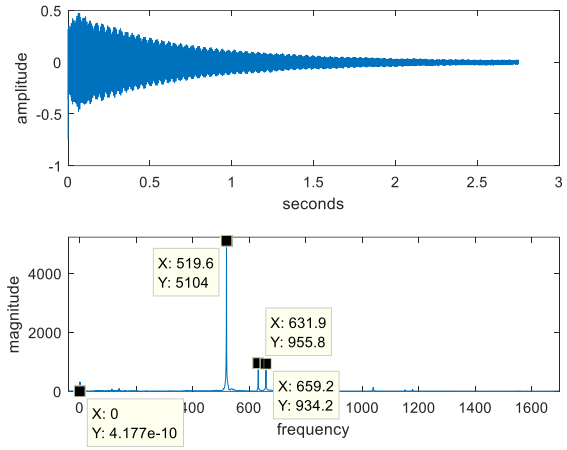
```
% part (d) DC Removal
basebandOutput = basebandOutput - mean(basebandOutput);
```

The mean square error is about 133 and the normalized mean squared error is 0.001099. The MSE has greatly improved with the sequential processing taken in the previous steps.

`basebandOutput` DC Removal



DC Removal (Zoomed in One-sided Spectrum)



The principal non-negative frequencies are 520, 630, 660, 1040, 4760, 5280, and 5800 Hz. By subtracting out the mean, the DC component has been removed. Other major frequency components could be easily measured with cursors in the figure. The higher frequency components above 660 Hz that are not originally in the gong signal are significantly attenuated but not completely removed. This is due to the quality of digital lowpass filter we use. We will explore the design and analysis of digital filters in upcoming lecture and homework.

Note: The `sound` command in Matlab will clip amplitude values that fall outside of the range from -1 to 1. It is important to check to make sure that any vector of signal values sent to the sound are in range, e.g.

```
>> max(abs(basebandInput))
0.6723
```

Epilog: Why did we need to add a DC offset to the baseband signal?

Upconverted signal is $s(t) = (1 + m(t)) \cos(2 \pi f_c t)$ where $m(t)$ is on the interval [-1, 1].

Apply a squaring device to $s(t)$ to yield $y(t) = s^2(t) = (1 + m(t))^2 \cos^2(2 \pi f_c t) = (1 + m(t))^2 (½ + ½ \cos(4 \pi f_c t))$

Apply a lowpass filter with double bandwidth of $m(t)$ and a gain of 2 to $y(t)$ to yield $v(t) = (1 + m(t))^2$

Apply a square root operation to yield $1 + m(t)$ provided that $1 + m(t) \geq 0$

Remove the DC component of $1 + m(t)$ to obtain $m(t)$.

If we hadn't added the DC offset, then baseband output signal would have been $| m(t) |$ and not $m(t)$.

Adding the DC offset also greatly simplifies the receiver when using a squaring block but doubles the transmission power. The squaring block represents the voltage effect of a nonlinear circuit such as a transistor or vacuum tube (before transistors). The receiver can be implemented with a single transistor and a handful of resistors and capacitors. Here's an example AM radio receiver circuit.

AM radio stations started widespread broadcasting in the 1920s using this style of double sideband large carrier (DSB-LC) amplitude modulation (i.e. adding a DC offset to the baseband speech/audio signal). They made the broadcasting transmitter more expensive so that receiver would be inexpensive and more people would be able to buy one for their homes. The invention of the transistor in 1947 enabled the invention of the transistor radio. The first transistor radio reached the market in 1954.

The DC component is at baseband and will be modulated to be centered at the carrier frequency. This places a lot signal power at the carrier frequency, which gives rise to the name Double Sideband - Large Carrier Amplitude Modulation. Adding a DC component at baseband is wasteful in the sense that it is not being used to make the message signal (basebandInput) more powerful. In AM radio transmissions, the DC component has as much power in it as the message signal. After modulation, the extra power in the DC component shifts to the carrier frequency, which greatly simplifies the receiver. A receiver can be constructed out of a single transistor. For more information, see the Web page for lecture 19 on sinusoidal modulation (optional content).