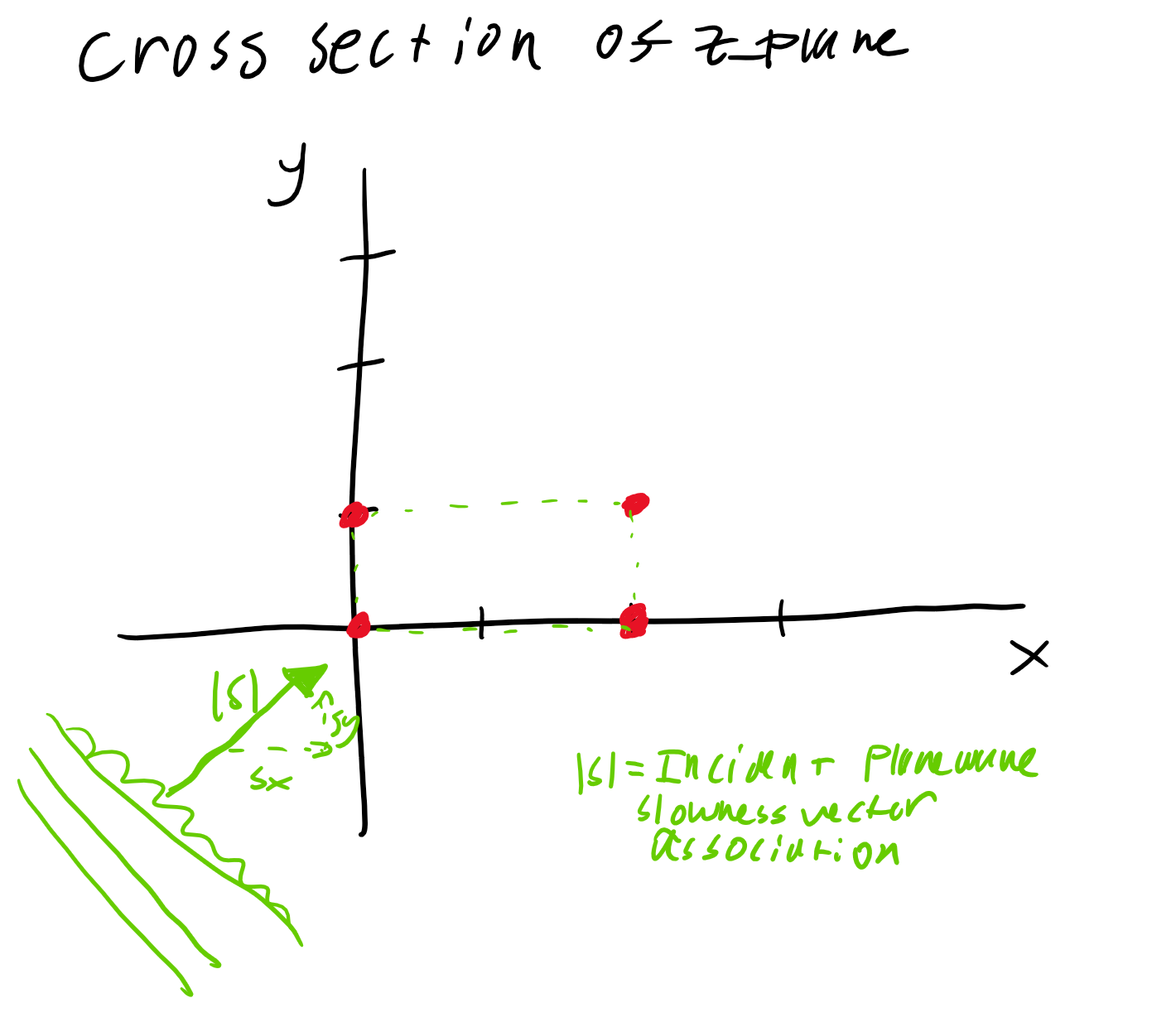
Matthew Luu

Quiz #3

1. Suppose that you’re working with a two-dimensional array of four receiving transducers. By two-dimensional, I mean that the element positions can be described completely by (x,y) coordinates—all elements have the same z coordinate, which you can take to be zero. The elements are located at the corners of a rectangle and have the following (x,y) coordinates in meters: (0.0, 0.0), (2.0, 0.0), (0.0, 1.0), (2.0, 1.0) for elements 1, 2, 3, and 4 respectively. For one measurement, you are given the x- and y-components of the slowness vector associated with the incident plane wave, Sx = 0.475e-3 s/m; Sy = 0.475e-3 s/m, and that the z-component is zero.

a. Make a sketch of the elements and include the direction of the slowness vector. Don’t worry about the length of the slowness vector—its units are not meters—just show the correct direction.



b. Is the array more likely to be in air or in the ocean? Why?

By calculating the speed of sound using the slowness vector and position coordinates, I can figure out if the measurements were in the ocean or air. The calculated speed of sound was: 1489 m/s, considering the speed of sound in water is 1500 m/s, it is safe to say that the array is in the **ocean**.

­c\_slowness = round((1/sqrt(S(1)^2 + S(2)^2)));

disp(['Speed of Sound: ', num2str(c\_slowness),' m/s']);

1. For the plane wave with the above slowness vector, what are the times of arrival at elements 2, 3, and 4, if using element 1 as the reference (so the arrival time at 1 is 0.0 s)?

|  |  |
| --- | --- |
| Td | Times (ms) |
| t11 | 0 |
| t12 | 0.9 |
| t13 | 0.5 |
| t14 | 1.4 |

To calculate these values, the equation: Td = r\*S is used.

x = 1;

y = 2;

r11 = [0,0];

r12 = [elem2\_loc(1) - elem1\_loc(1),elem2\_loc(2) - elem1\_loc(2)];

r13 = [elem3\_loc(1) - elem1\_loc(1),elem3\_loc(2) - elem1\_loc(2)];

r14 = [elem4\_loc(1) - elem1\_loc(1),elem4\_loc(2) - elem1\_loc(2)];

r = [r11;r12;r13;r14];

td = r\*(S.')

For another measurement with a different signal (but still arriving as a plane wave at the array), you measure the following arrival times: t1 = 0.0; t2 = 0.598 ms; t3 = -0.635 ms; t4 = -0.003 ms.

1. Find the x and y components of the slowness vector for this plane wave.

For this part, a new slowness vector must be calculated with the time delays. In order to do so, S = r\T is used. This is essentially r\*inverse(T). With the slowness vector, the speed of sound is calculated.

Sx = 0.3047e-0.3

Sy = -0.6237e-0.3

%% Part d

t1 = 0;

t2 = 0.598\*10^-3;

t3 = -0.635\*10^-3;

t4 = -0.003\*10^-3;

T = [t1,t2,t3,t4].';

S\_new = r\T;

1. What is the sound speed?

The speed of sound is now just one over the magnitude of the slowness vector.

c\_slowness = round((1/sqrt(S\_new(1)^2 + S\_new(2)^2)));

disp(['Speed of Sound: ', num2str(c\_slowness),' m/s']);

c = 1441 m/s

1. A pulse with a frequency that varies quadratically with time (i.e., traces part of a parabola in time-frequency space) can be useful in imitating some natural sounds. In this problem, you’ll construct an approximation to the dominant component of a crow’s “caw”. It won’t sound much like a crow because there are other significant components but it’s a place to start.

a. Write a function to take as input three time/frequency pairs (i.e., a 3x1 vector of times and a 3x1 vector of frequencies) and fit those three points with a frequency function that is quadratic in time. You can either generate the coefficients of the second-order polynomial or write the function to produce a vector of frequency values given an input vector of time values. You may use the polynomial-fit function in MatLab/Octave/Python or you can write the relevant matrix equation (three equations in three unknowns) and solve directly.

f\_pt and t\_pt are the 3x1 vectors, f\_poly is the polynomial equation based on time and f\_coef are the coefficients of the quadratic equation. This function utilizes the polyfit built in function in order to get said coefficients.

function [f\_poly,f\_coef,t] = Quadratic\_Frequency\_Sweep\_Freq(f\_pt,t\_pt,fs)

f\_coef = polyfit(t\_pt,f\_pt,2);

dt = 1/fs;

t = (t\_pt(1):dt:t\_pt(3));

f\_poly = f\_coef(1).\*t.^2 + f\_coef(2).\*t + f\_coef(3);

end

b. How can you check to be sure your function produces reasonable results? Execute that check and show the results.

To test this function, I created a plot of the frequency vs. time and saw the quadratic profile. I made this simple test case, which was easy to understand:

f\_pt = [100,500,1000];

t\_pt = [0,3,6];

Therefore, at t=0, 100Hz should occur, t=3, 500 Hz should occur, t=6, 1000 Hz should occur. So I could see this on a plot in a quadratic formulation.



I also performed one more test to see the quadratic nature.

f\_pt = [100,500,100];

t\_pt = [0,3,6];

Therefore, at t=0, 100Hz should occur, t=3, 500 Hz should occur, t=6, 100 Hz should occur. So I could see this on a plot in a quadratic formulation. This case should be a downward facing quadratic function.



The profile matched what was expected and the times match the frequencies well.

c. Modify that function to also generate the corresponding instantaneous phase. As before, you can either find the coefficients of whatever polynomial properly expresses the phase or produce a vector of phase values given an input vector of time values.

The function for angle poly fit and frequency poly fit are essentially the same. This time the ang\_pt is the angular 3x1 vector.

function [ang\_poly,ang\_coef,t] = Quadratic\_Frequency\_Sweep\_Ang(ang\_pt,t\_pt,fs)

ang\_coef = polyfit(t\_pt,ang\_pt,2);

dt = 1/fs;

t = (t\_pt(1):dt:t\_pt(3));

ang\_poly = ang\_coef(1).\*t.^2 + ang\_coef(2).\*t + ang\_coef(3);

end

d. Use the following three time/frequency pairs, (0.145 s, 1020 Hz), (0.190 s, 1410 Hz), (0.280 s, 990 Hz), to generate a pulse with an instantaneous frequency that varies quadratically. Create a vector that is 0.4 seconds long and insert the pulse so that it starts at t = 0.145 s and ends at t = 0.280. Before and after this interval the values in the vector should be zeros. When the pulse is not zero, the amplitude of the oscillation peaks should be 0.9 WU. (In practice, you’d smooth the leading and trailing edges of the pulse—don’t worry about that in this problem.)

To create an xn waveform I need to take the integral of the f(t) function. The f(t) function is a quadratic function so the ϕ(t) function is going to be:

The wave form equation is now:

This was implemented in matlab where A, B, and C are the polynomial coefficients calculated with the function in part 2a.

% Insert the points

f\_pt = [1020,1410,990];

t\_pt = [0.145,0.190,0.280];

% Execute

[f\_t,f\_coef,t\_pulse] = Quadratic\_Frequency\_Sweep\_Freq(f\_pt,t\_pt,fs);

%% Part d and e

% Need the integral for instantaneous phase

f\_t\_integral = f\_coef(1).\*(t\_pulse.^3)./3 + f\_coef(2).\*(t\_pulse.^2)./2 + f\_coef(3).\*t\_pulse;

phase\_t = 2\*pi\*f\_t\_integral;

The resultant waveform looks as follows:



e. How would you test the pulse to be sure the frequency variation is what you intended? Execute that check and show the results.

I can use a spectrogram to test the results and see the frequency variation with time and amplitude. By plotting the spectrogram, I get this result:



The spectrogram clearly shows zeros from 0 to 0.145s and 0.280 to 0.4s. Furthermore, the quadratic signal is shown as the bright red area. This spectrogram proves the quadratic chirp nature of the signal.

f. The wav file, Bird\_call\_iso1.wav, contains a pair of actual crow “caw”s. Using some tool from this class, compare your synthetic “caw” to the actual recording. Describe what you would do to make the synthetic “caw” sound more realistic. You don’t have to execute these improvements, just describe what you’d do.

The figure below shoes the spectrogram of the bird call. To understand how to make the synthetic caw, I would need to understand what the basic spectrogram looks like. The synthetic caw, around 1000 – 1500 Hz looks like this quadratic pulse that we created. I assume there are harmonics of this caw, so to create this synthetic pulse I would create 3-5 quadratic pulses spaced at the harmonics of the fundamental range (1000 – 1500 Hz)



3. suggests a simple time-domain filter with input, x, output, y, and a single parameter, r. In fact, this filter is sometimes used to filter random, white noise in order to produce a noise-like signal that is more representative of natural noise processes. The constant, r, specifies how much similarity there is between adjacent output samples.

a. What does this filter do if r = 0? If r = 1?

If r = 0, then yn = xn-1, therefore the filter is an all pass filter with a delay, i.e. a phase shift.

If r = 1, then yn = yn-1, therefore the filter is a not pass filter because there is not input xn signal.

b. Find the a and b filter coefficients so that you can use the MatLab filter function to implement the filter.

Filter coefficient:

aa = [1, -r];

bb = [0, sqrt(1-(r^2))];

a1 = 1, a2 = -r

b1 = 0, b2= sqrt(1-r^2)

To find the filter coefficients by seeing the constant before the y term or x term. Xn and yn are denoted as a1 and b1 while anything with (n-m) is denoted as higher order terms.

c. How would you test the filter? Try that test (or tests) and discuss the results.

To test the filter, we can plot the response through a unit sample response and the FFT of that function. Therefore, we can plot the frequency and phase response. I plotted the filter at r = 1 and r = 0 to show that the filter would be a no pass and all pass filer. The results sow this and what was expected from part a

R=0:



R=1:



d. Find and plot the magnitude and phase of the frequency response of this filter from 0 to fs/2 Hz using a sampling frequency, fs, of 5000 and r = 0.85.

The code below shows the plot filter function. This plot filter function takes the impulse response and FFT of the impulse response to show the frequency response.



r = 0.85;

aa = [1, -r];

bb = [0, sqrt(1-(r^2))];

fs = 5000;

MyDSP.plot\_filter\_simple\_Ak\_BK(aa,bb,1000,fs,1,[0 fs/2]);

fx = -fs\*log(r)/(2\*pi);

figure(1);

labasd = 0:1/fs/2:0.8-(1/fs);

subplot(2,1,1); plot(fx\*ones(1,length(labasd)),labasd,'Linewidth',1.6);

e. According to literature on this filter, the value fx = -fs\*ln(r)/(2π) gives a frequency that has0 a special meaning with regard to the frequency response. (The “ln” here is the MatLab log or natural log.). What relationship do you see?

This relationship is the 3dB point of the lowpass filter. The fx value found and the 3dB point found is:

fx = 129.33 Hz

f3dB = 130 Hz

This 3dB, in terms of frequency response, is the red line in the frequency response of part d.

[dn, ~] = MyDSP.unit\_sample(1000,'n');

h = filter(bb,aa,dn);

[H,f] = MyDSP.MyFFT(h,fs,'n',0);

dB\_down = 0.707\*max(abs(H));

Idx = MyDSP.find\_val(abs(H),dB\_down,0.001,0.001);

f\_dB\_down = f(Idx(1));

fx = -fs\*log(r)/(2\*pi);

disp(['Determined 3dB down pt: ',num2str(f\_dB\_down)]);

disp(['Determined fx: ',num2str(fx)]);