EEC 193AB

Application Note

DSP for Radar

Said Mansoor Wahab

Professor: Xiaoguang "Leo" Liu

Introduction:

Signal Processing deals with the representation, transformation, and manipulation of signals and the information the signals contain. Discrete-time signal processing is based on processing numeric sequences indexed on integer variables rather than functions of continuous independent variable (analog signals). In *Digital Signal Processing* (DSP), signals are represented by a sequence of finite-precision numbers, and processing is implemented using digital computation. Discrete –time signal processing includes DSP as a subset, but the distinction between the two is of minor importance since both are concerned with discrete-time signals.

Although not as intricate as signal processing for communication systems, radar signal processing still presents myriad challenges. A major application of radar is distance detection and tracking. This introduces the concept of frequency estimation due to the linear relationship between distance and frequency. We need to find the intermediate frequency (IF) to calculate distance at a given time. This calls for some real-time DSP that can be implemented on a computer, FPGA, or a microprocessor. The sampling rate of the ADC is critical for range and resolution.

Radar and Signal Processing literature contain numerous techniques for frequency estimation proposed by experts in the field. Some of the methods are fine search, interpolation, and approximation of nonlinear models. In this application note, I will discuss 3 rudimentary algorithms sufficient for a student taking a part in radar senior design project.

Structure:

We begin with a brief review of the sampling theorem. This is a vital step in understanding DSP, and we strongly encourage the reader to use *Discrete-Time Signal Processing* by Oppenheim and Schafer for reference. It would be ideal for the student to be concurrently enrolled in a DSP course such as EEC150B or EEC201. After discussing Shannon's sampling theorem, we introduce three possible algorithms for frequency estimation and provide Matlab code for verification and implementation purposes. The paper concludes with demonstrations and results of a piece of data recorded by students in the lab.

Sampling Theorem:

DSP was born in the early 1960s with the invention of digital computers. Since a machine can only store finite amount of data, engineers had to come up with a way to extract information from analog signals without losing anything invaluable. It turns out a technique had already existed. It was called the Shannon-Nyquist Sampling Theorem proposed in 1928 by two all time famous engineers Claude Shannon and Harry Nyquist. The sampling theorem deals with sampling an analog signal and storing the samples as opposed to working with the entire analog signal. There is one constraint and one rule, however. The signal must be bandlimited and we

must sample at least **twice** the bandwidth for the samples to fully characterize the original signal. We next describe some basic mathematics of sampling.

Time-domain:

Let $x_c(t)$ *be the continious, ana*log *signal and* $x(n)$ *be the discrete, sampled signal*

Let $T_s = \frac{1}{F_s}$ *be the sampling period* (F_s *is the sampling rate or frequency*). *Then*

$$x(n) = x_c(nT) \qquad \forall n$$

In other words, the function $x(n)$ *has values of* $x_c(t)$ *only at multiples of* T_s.

$$\begin{aligned} T_s &\leftrightarrow 1 \\ 2T_s &\leftrightarrow 2 \\ 3T_s &\leftrightarrow 3 \\ &\vdots \\ nT_s &\leftrightarrow n \end{aligned}$$

$x_s(t)$ *is the continous sampled version.*

Let $s(t) = impulse\ train = \sum_{\forall k \in Z} \delta(t - kT_s)$. *Then*

$$x_s(t) = x_c(t)s(t) = \sum_{\forall k \in Z} x_c(t)\delta(t - kT_s) = \sum_{\forall k \in Z} x_c(kt)\delta(t - kT_s)$$

This is our equation for the sampled signal. It may look complicated, but it is quite intuitive. Consider the following diagrams. They make the concept less obscure.

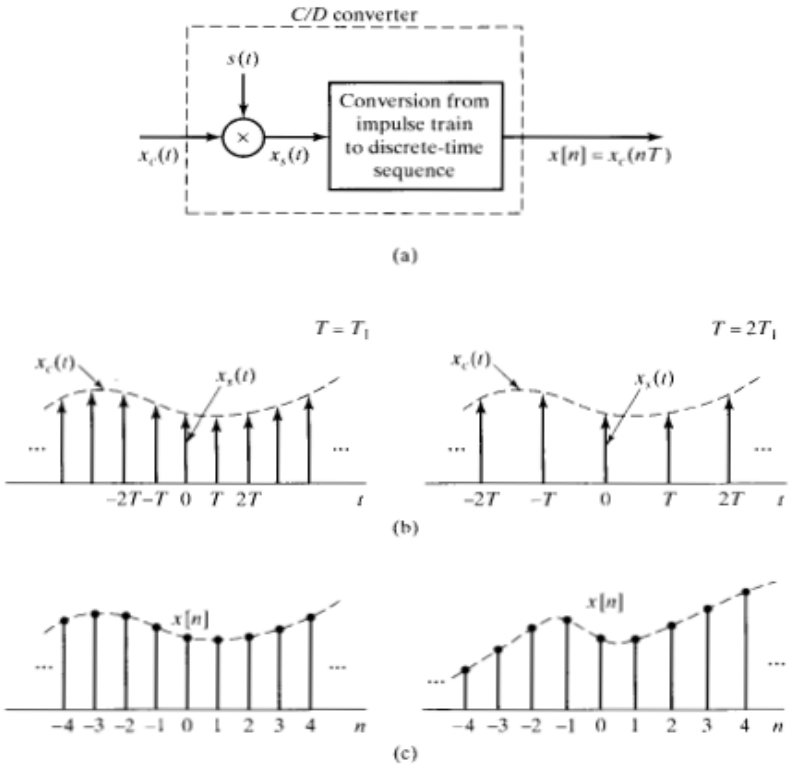


Figure 4.2 Sampling with a periodic impulse train followed by conversion to a discrete-time sequence. (a) Overall system. (b) $x_s(t)$ for two sampling rates. (c) The output sequence for the two different sampling rates.

Frequency-domain:

$$X(j\Omega) = \int_{-\infty}^{\infty} x_c(t) e^{-j\Omega t} dt \qquad Continous-time\ Fourier\ Transform\ with\ \ frequency\ \Omega$$

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x(n) e^{-jwn} \qquad Discrete-time\ Fourier\ Transform\ with\ \ frequency\ \ w$$

The two transforms are related by the following formula:

$$X(e^{jw}) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X_c\left[j(\frac{w}{T_s} - \frac{2\pi k}{T_s}) \right]$$

$$\Rightarrow \qquad w = \Omega T_s$$

The relationship between Continous-time frequency and Discrete-time frequency is very important. All the signals processed in Matlab are sampled by the computer's sound card, and it is up to the student to convert the frequencies displayed in Matlab to frequencies corresponding to an actual analog signal. We note the Discrete-time Fourier Transform contains infinite number of shifted Continous-time Fourier Transforms that are scaled both in frequency and amplitude.

Nyquist-Shannon Sampling Theorem:

Let $x_c(t)$ *have finite bandwidth of* Ω_N, *then* $x_c(t)$ *is uniquely characterized by its samples*

$$x(n) = x_c(nT_s), \qquad n = 0, \pm 1, \pm 2, \ldots$$

if $\Omega_s = \frac{2\pi}{T_s} \geq 2\Omega_N$

Although having the sampling rate equal exactly twice the bandwidth is mathematically correct, it makes a poor practical choice. Due to imperfectations of commercial ADCs, it is recommended to samples strictly above the Nyquist rate (twice the bandwidth).

Frequency estimation algorithms:

Before introducing the reader to 3 possible algorithms, we like to say a few words about the calculation of DTFT. From the above formula, we see DTFT is a summation of infinite exponentials (single tones in frequency domain). Although the signal is discrete in time, its frequency response (DTFT) is general continuous. This makes it impossible to represent the whole signal in a finite-memory machine. For this reason, we sample DTFT and call it DFT (Discrete-Fourier Transform). DFT depends on N, which is the size of the discrete-time signal and determines frequency resolution. FFT is the efficient computation of DFT. Matlab has a built-in FFT function, thus eliminating the need and pain of writing one oneself.

Recall, FFT is index based. Matlab gives no indication of frequency. When calculating DTFT, we need frequency (w) values and response at those values. The following short Matlab function, a supplement to the DSP book, calculates DTFT at $w \in [-\pi, \pi]$. We remind the reader Discrete-time Fourier Transform is periodic with 2π unlike the Continous-Time Fourier Transform which in general has no periodicity.

Short DTFT code based on FFT

```
function [H, W] = dtft(h, N)
%DTFT   calculate DTFT at N equally spaced frequencies
%----
%   Usage:   [H, W] = dtft(h, N)
%
%      h : finite-length input vector, whose length is L
%      N : number of frequencies for evaluation over [-pi,pi)
%              ==> constraint: N >= L
%      H : DTFT values (complex)
%      W : (2nd output) vector of freqs where DTFT is computed

%-------------------------------------------------------------
% copyright 1994, by C.S. Burrus, J.H. McClellan, A.V. Oppenheim,
% T.W. Parks, R.W. Schafer, & H.W. Schussler.  For use with the book
% "Computer-Based Exercises for Signal Processing Using MATLAB"
% (Prentice-Hall, 1994).
%-------------------------------------------------------------

N = fix(N);
L = length(h);  h = h(:);  %<-- for vectors ONLY !!!
if( N < L )
   error('DTFT: # data samples cannot exceed # freq samples')
end
W = (2*pi/N) * [ 0:(N-1) ]';
mid = ceil(N/2) + 1;
W(mid:N) = W(mid:N) - 2*pi;   % <--- move [pi,2pi) to [-pi,0)
W = fftshift(W);
H = fftshift( fft( h, N ) );  %<--- move negative freq components
```

The above function will be used in our algorithms when desired.

Recall that a sinusoid in frequency domain is two tones at ±w. Due to sampling and DFT leakage, however, we see less ideal tones which make it difficult to find the exact frequency. Thus, multiple algorithms have been proposed for frequency estimation. Some of the algorithms have been extended to account for noise which is inevitably present in all systems and signals. We will not discuss noise here since it requires a working knowledge of stochastic (random) processes at the level of EEC 260.

Algorithm 1

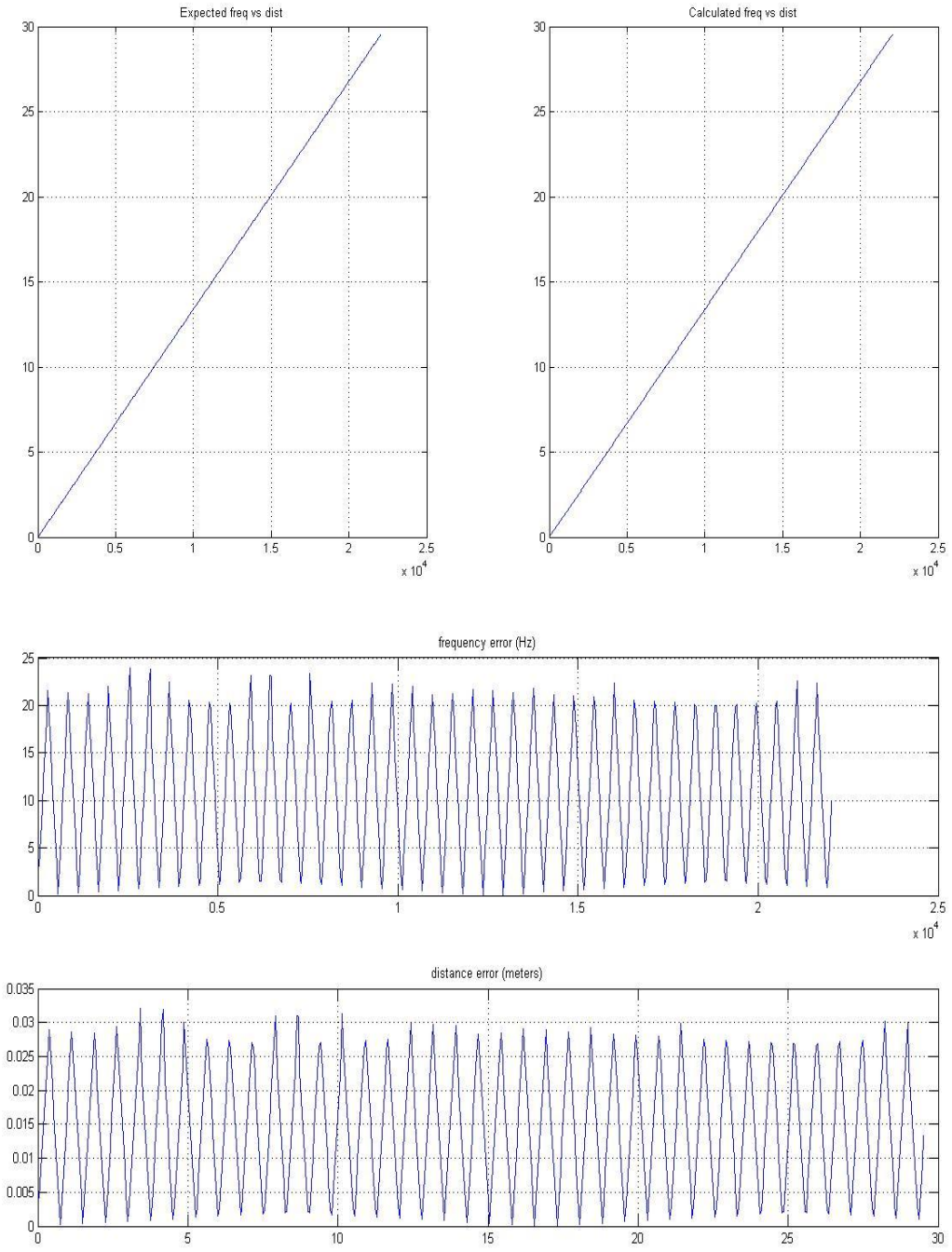
The simplest and most intuitive algorithm is based on dtft function and frequency associated with the maximum value. Due to non-idealities discussed above, we may not always land on the exact desired frequency. However, if the sampling in frequency domain is tight, i.e. the signal size is large (zero-padding), the difference between the obtained frequency and exact frequency may be negligible. This algorithm has little mathematical foundation, but has delivered satisfactory results based on experience.

The following Matlab script uses this algorithm and plots the theoretical (expected) versus calculated frequency. The reader should thoroughly read the comments at the beginning to understand the functionality.

```
% READ
%-------------------------------------------------------------------------
% We attempt to mimic a real time signal.
% The signal is a combination of sinusoids. The smallest sinusoid is 20Hz
% The last sinusoid is of freq 5000 Hz (half of sampling rate). Each
% sinusoid is 20 Hz higher than the the previous sinusoid.
% Each sinusoid is 1024 samples and thus lasts 0.1 seconds. This is a good
% representation of real life since we can assume that the change in
% distance in 0.1 seconds is very small (thus the freq change is also small)
% After generating the signal, we implement STFT (Short Time Fourier Trans)
% on the signal. From each segment of STFT, we obtain the frequency of the
% sinusoid within that segment and calculate the distance appropriately.
% Finally we plot two graph:
% 1) Distance vs freq  (theoretical/ideal)
% 2) Distance vs freq  (based on calculations)
%-------------------------------------------------------------------------
clc
clear
c = 3*10^8;
Tp = (25/2)*10^-3;
BW = 700*10^6;
Fs = 44.1*10^3;  % sampling rate of Computer Sound Card
N = 1024;  % length of each sinusoid
n = 0:1/Fs:((N-1)/Fs);   % Discrete time
f0 = 40;       % initial frequency (first sinusoid)

% This theoretically coresponds to a distance of about .05 meters
t0 = .1;    % actual time for one sinusoid
f = [];
dis_exp = [];   % theoretical distance
t = [];
tp = t0;
i = true;   % loop variable
fp = f0;
freq_exp = [];  % theoretical frequency

% Whole signal generated corresponds to about 25 seconds
while i == true
   f_ind = cos(2*pi*fp*n) + randn(1,N);   % sinusoid + noise
   f = [f f_ind];
   dis_exp = [dis_exp (fp*c*Tp)/(4*BW)];
   freq_exp = [freq_exp fp];
   t = [t tp];
   fp = fp + f0;    % new freq
   tp = tp + t0;

   if fp > Fs/2;
       i = false;
   end
end

time = linspace(0,(((Fs/2)-f0)/f0)*t0,length(f));
i = true;
index_1 = 1;
index_2 = N;
freq_calc = [];     % calculated frequency (predifinition)
dis_calc = [];      % calculated distance  (predefinition)

while index_2 <= length(f)
    s = f(index_1:index_2).*hamming(N)';
    [S W] = dtft(s,N);      % dtft is not a standard Matlab function
    k = find(W==0);         % find index of DC
    S(k) = 0;               % set DC to zero
    [~, i] = max(abs(S));
    w = abs(W(i));
    delta_F = w*Fs/(2*pi);
    freq_calc = [freq_calc delta_F];
    dis_calc = [dis_calc (delta_F*c*Tp)/(4*BW)];
    index_1 = index_1 + N;
    index_2 = index_2 + N;
end

error_freq = abs(freq_calc - freq_exp);
error_dis = abs(dis_calc - dis_exp);

figure(1)
subplot(121),plot(freq_exp,dis_exp),grid,title('Expected freq vs dist')
subplot(122),plot(freq_calc,dis_calc),grid,title('Calculated freq vs dist')

figure(2)
subplot(211),plot(freq_exp,error_freq),grid,title('frequency error (Hz)')
subplot(212),plot(dis exp,error dis),grid,title('distance error (meters)')
```

The section in the code where we use the described algorithm starts and ends inside the second ‘while’ loop. The program is a script and not a function. It does not operate on a real piece of data, but rather has been written for algorithm evaluation.

Here are the results when the program runs:



We see the results are quite accurate. From the first figure, the relationships between frequency and distance look identical in both cases (theoretical vs calculated). The second figure shows we have a maximum error of less than 25Hz in our calculation. This may seem large, but represents a small error in distance as seen above.

Algorithm II

This algorithm is called a frequency estimator and gives quite coarse results.

$$f = \frac{(\overline{k_1} + \delta)F}{N}, where\ F = sampling\ rate$$

$$N = window\ size,\ \overline{k_1}\ is\ the\ sample\ at\ the\ highest\ peak$$

$$\delta = what\ we\ estimate$$

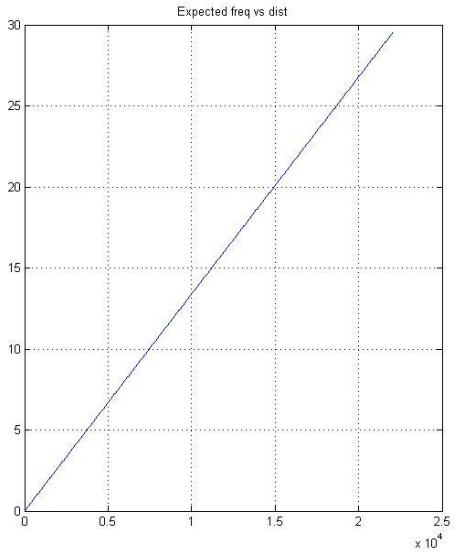
$$\delta_{\pm} = \pm \frac{|R_{\pm}|}{1+|R_{\pm}|} \qquad Rife\ and\ Vincent$$

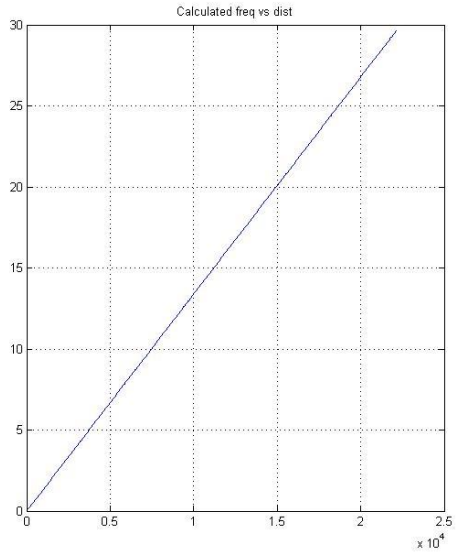
$$where\ R_{\pm} = \frac{S(\overline{k_1} \pm 1)}{S(\overline{k_1})} \qquad where\ S(k)\ is\ DFT\ of\ s(n)$$

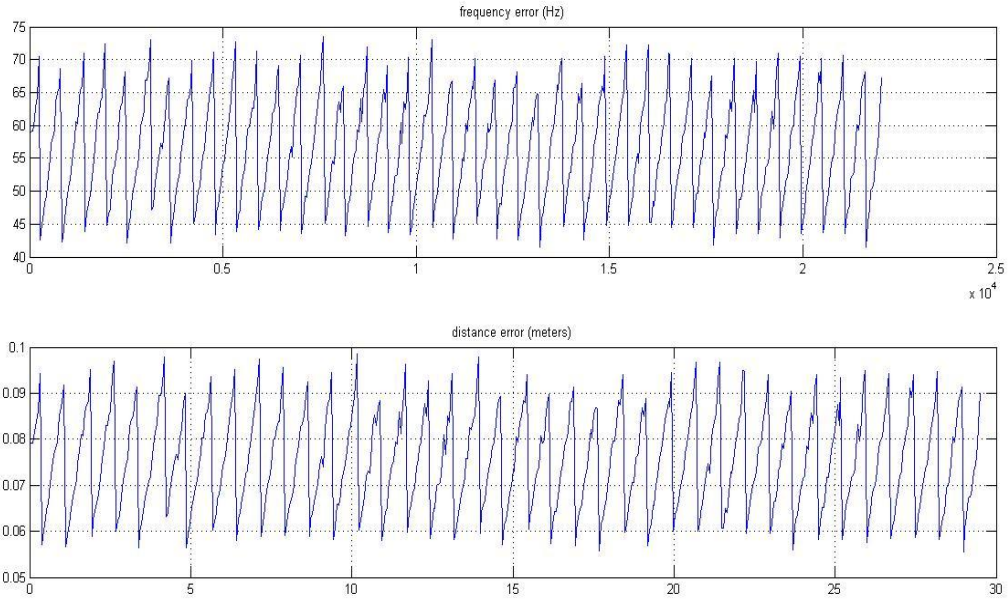
$$Gives\ exact\ f\ as\ N \to \infty$$

We see the estimated frequency can be quite inaccurate if N is small. We evaluate this algorithm in the same way as algorithm I by changing the second 'while' loop in the following way:

```
while index_2 <= length(f)
    s = f(index_1:index_2).*hamming(N)';
    S = fft(s,N);
    [~, k] = max(abs(S));
    R = S(k+1)/S(k);
    delta = abs(R)/(1+abs(R));
    delta_F = (k+delta)*Fs/N;
    freq_calc = [freq_calc delta_F];
    dis_calc = [dis_calc (delta_F*c*Tp)/(4*BW)];
    index_1 = index_1 + N;
    index_2 = index_2 + N;
end
```







We see the max error with algorithm 2 is about 3 times higher than with algorithm 1. We expected this since the algorithm gives crude estimation. The max error in frequency corresponds to about 10cm of error in distance which can be large depending on the application.

Algorithm III

This algorithm is the most mathematical of all, but gives theoretically exact results. The mathematics described relates to complex exponentials (single tones) as opposed to real sinusoids (double tones).

$$s(n) = A\lambda^n, \qquad S(k) = \frac{p}{1 - W_N^k \lambda} \text{ for } k = \{0, N-1\}$$

$$\text{with } \lambda = e^{jw},\ W_N = e^{-j2\pi m/N},\ p = A(1-\lambda^N)$$

$$\text{After Manipulation, we get:}$$

$$S(k) = [1 \;\; S(k)W_N^k] \begin{bmatrix} p \\ \lambda \end{bmatrix}$$

$$\begin{bmatrix} S(k_1) \\ S(k_2) \end{bmatrix} = \begin{bmatrix} 1 & S(k_1)W_N^{k_1} \\ 1 & S(k_2)W_N^{k_2} \end{bmatrix} \begin{bmatrix} p \\ \lambda \end{bmatrix}$$

$$\text{For } N \geq 2, \text{ it is possible to select two indices } k_1 \text{ and } k_2$$
$$\text{to build a linear system of equations}$$
$$\text{from which it is easy to solve for } p \text{ and } \lambda, \text{which give}$$
$$w \text{ (desired discrete} - \text{time frequency) and its strength}$$
$$\text{We get exact frequency if } k_1 \text{ is chosen to be sample at}$$
$$\text{the highest peak.}$$

Since for our practical case we are dealing with sinusoids, we need to extend the above results to 2 complex exponentials of equal and opposite frequencies. This sounds simple, but gets cumbersome mathematically. We encourage the reader to read resource [3] where the author discusses M tones instead of just one or two.

This algorithm has not been implemented in Matlab. It would be ideal if future EEC 193AB/134AB students coded the algorithm in Matlab after thoroughly understanding resource [3].

Matlab Functions:

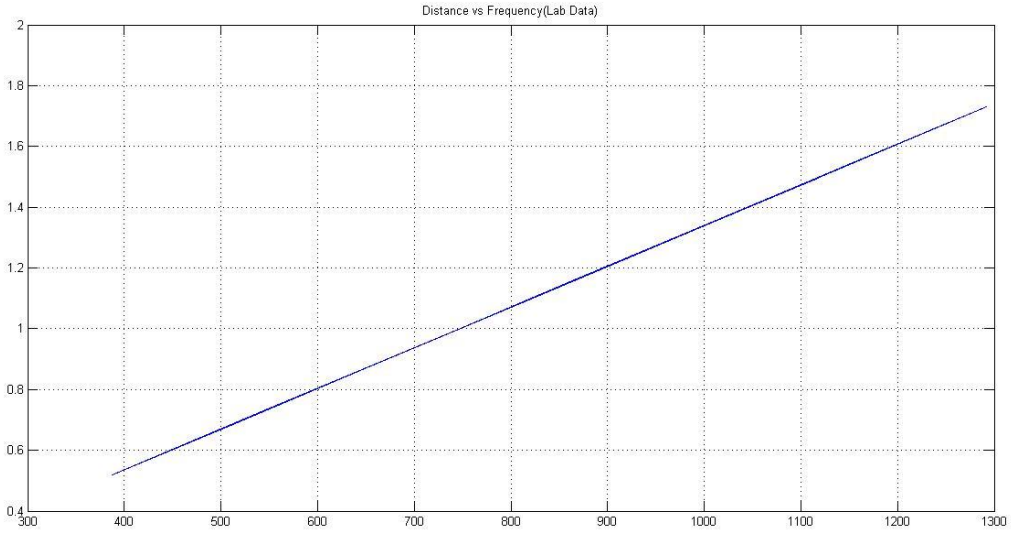
We now demonstrate how algorithm 1 can be applied to a real piece of data. The data used represents a student walking away from the radar up to 2 meters. Since recorded in the lab, the data contains lots of noise. Here is the Matlab function that can be used on any signal with any appropriate sampling rate.

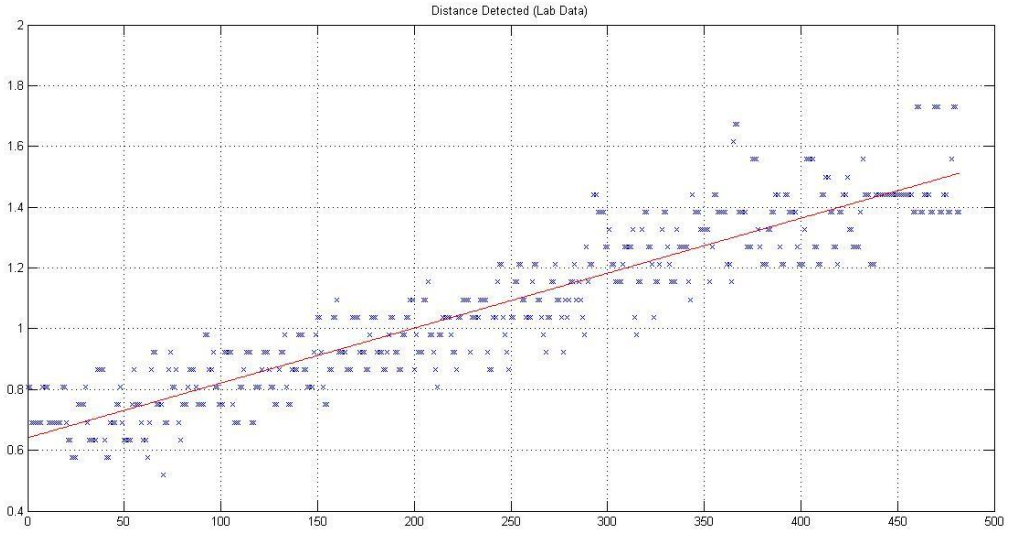
```
function [dist freq] = arduino(f,Fs)

clc
c = 3*10^8;
Tp = (25/2)*10^-3;
BW = 700*10^6;
N = 1024;
index_1 = 1;
index_2 = N;
freq_calc = [];      % calculated frequency
dis_calc = [];       % calculated distance
fc = 5*10^3;         % cutoof frequency
wn = (2/Fs)*fc;      % normalized cutoff frequency
b = fir1(20,wn,'low',kaiser(21,3));
f = filter(b,1,f);
j = 1;

while index_2 <= length(f)
    x = f(index_1:index_2).*hamming(N);     % window (use hamming)
    s = x;
    [S W] = dtft(s,N);       % dtft is not a standard Matlab function
    T{j} = W;
    F{j} = S;
    k = find(W==0);          % find index of DC
    S(k) = 0;                % set DC to zero
    [~, i] = max(abs(S));
    w = abs(W(i));
    delta_F = w*Fs/(2*pi);
    freq_calc = [freq_calc delta_F];
    dis_calc = [dis_calc (delta_F*c*Tp)/(4*BW)];
    index_1 = index_1 + N;
    index_2 = index_2 + N;
    j = j + 1;

end

dist = dis_calc;
n = 0:length(dist)-1;
coeff = polyfit(n,dist,1);  % linear fit through the data
p = polyval(coeff,n);
freq = freq_calc;

figure(1)
plot(freq,dist),title('Distance vs Frequency(Lab Data)'),grid
figure(2)
plot(n,dist,'x',n,p,'r'),title('Distance Detected (Lab Data)'),grid
end
```

NOTE: In the above function, f is the signal and Fs is the sampling rate. To obtain the signal array and sampling rate, place the recorded WAV file in the same directory as the function and then type in the following in Matlab environment.

```
[y Fs] = wavread('filename.wav');
f = y(:,2);        % the signal is on the second channel
```

Here is the result of the data described above.





The points on the last graph represent actual data. The red line is the Least Squares regression. This is an important step since any data has ± error. Luckily, Matlab has a built in least squares function. We conclude our results are quite decent since the line fits the description of the data. The results are not perfect due to inaccuracies in frequency estimation and noise.

Conclusion:

To understand Radar DSP, we must start with fundamentals of DSP one of which is the sampling theorem. It is critical to be able to convert between discrete-time and continuous-time frequencies. Matlab (dtft or fft functions) provides discrete-time frequency, but for distance estimation we require continuous-time frequency. Multiple algorithms are at the service of a DSP engineer when estimating the frequency of a sinusoid. It is up to the engineer to pick an algorithm that is accurate, robust, and computationally efficient.

Resources:

[1] *Discrete-Time Signal Processing,* Alan Oppenheim and Ronald Schaffer

[2] *Estimation of Complex Singe-Tone Parameters in the DFT Domain,* Serge Provencher, IEEE Transactions on Signal Processing, July 2010

[3] *Parameter estimation of Complex Multitone Signal in the DFT Domain,* Serge Provencher, IEEE Transactions on Signal Processing, July 2011