Rachel King

ECE 3512: Stochastic Processes in Signals and Systems

Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA 1912

# Problem Statement

All parts of this lab involve computing the variance of the signal in some way; first, the variance for the entire duration of the signal. Then, for increasing values, starting from 10. And then, using the window-frame approach. The purpose of this lab is to demonstrate the different ways to calculate variance when the signal being processed is happening in real time.

# Approach and Results

*Parts 1 and 2 – Increasing Samples Approach*

The dotted lines in the following figures was computed using all the points in the signal to calculate a single value, which is extended across the whole plot. The red line is the variance calculated using an increasing number of points (x is the length of the file):

%starting at 10, incrememnt

for i = 10:x

 %segment file

 seg = speech(1:i);

 %compute and store

 var = nanvar(seg);

 n(i) = var;

end

 Each iteration, another point is added to the segmented array. The variance was calculated and then stored in a new array, which we later plot. This method took quite a bit of time, and seems extremely efficient, especially considering the variance of the whole file may not necessarily be information that we need.



Figure – Raw Speech File



Figure – Google Stock Price

*Part 3- Frame/Window Approach*

For this part, we computed the variance in a specific region of the signal. We kept the frame size small so that we could get an accurate reading of the signal with many points. Using the frame window code from previous labs, we added the following code:

X(n\_center) = nanvar(sig\_wbuf);

Where n\_center is the center point of the window, and sig\_wbuf is the actual segmented window of the file. Because this left many zeroes in between indices where values were stored (indices that there existed an n\_center value), we had to connect the points in between by making them ‘NaN’, and then assuming some points for those NaN values:

for i = 2:length(X)

 if X(i) == 0

 X(i) = X(i-1);

 end

end

Since this is an estimation of the variance, this ‘cheating’ shouldn’t be much of a problem (I hope). The black line signifies the Frame/Window estimation

Speech File



Figure – Speech File



Figure – Google Stock Prices

# MATLAB Code

*Parts 1 and 2*

%% plot audio file

function ca\_3\_1\_sp

h = fopen('rec\_01\_speech.raw');

speech = fread(h,inf,'int16');

[z, q] = size(speech);

vr = nanvar(speech);

y = linspace(0, z, z);

%call varience plot for speech file

real\_time(speech, z);

hold on

hline = refline(0, vr);

set(hline,'LineStyle',':')

end

function real\_time(speech, x)

%allocate size of array to store varience values

y = 1:1:x;

y = y';

%starting at 10, incrememnt

for i = 10:x

 seg = speech(1:i);

 %compute and store

 var = nanvar(seg);

 n(i) = var;

end

figure(1)

plot(y, n, 'r');

title('Variance Estimate, Increasing Samples', 'fontweight', 'bold');

xlabel('Sample Index');

ylabel('Varience Amplitude');

%

end

*Part 3*

function X = ca\_3\_final\_ish

 clear; clc;

% close open sessions

%

close all;

% define two key parameters:

% M: frame duration in samples - how often we compute an output

% N: window duration in samples - how much data we use in each computation

%

M = [ 80];

N = [ 240];

%[sig, txt, raw] = xlsread('google\_v00.xlsx', 1);

fp = fopen('rec\_01\_speech.raw','r');

sig = fread(fp,inf,'int16');

% create a matrix to store the output

X = compute\_rms\_1(sig, M, N);

plot(X, 'k');

end

 %{

% function: compute\_rms

%

% arguments:

% sig\_a: the input signal (input)

% fdur\_a: the frame duration in samples (input)

% wdur\_a: the window duration in samples (input)

%

% return:

% rms: a vector of rms values (output)

%

% Note that this function returns the rms counter as a sampled data

% signal that is the same length as the input signal. This is wasteful

% of memory, but makes it easy to produce a time-aligned plot.

%

% This algorithm computes the sum of squares for wdur\_a samples.

%}

function X = compute\_rms\_1(sig\_a, fdur\_a, wdur\_a)

% declare local variables

%

k = 1;

sig\_wbuf = zeros(1, wdur\_a);

num\_samples = length(sig\_a);

num\_frames = 1+round(num\_samples / fdur\_a);

rms\_full = zeros(length(sig\_a),1);

% loop over the entire signal

%

for i = 1:num\_frames

 %for indexing

 n\_center = (i - 1) \* fdur\_a + (fdur\_a / 2);

 n\_left = n\_center - (wdur\_a / 2);

 n\_right = n\_left + wdur\_a - 1;

 %make sure we're not using points outside the signal

 if( (n\_left < 0) || (n\_right > num\_samples) )

 sig\_wbuf = zeros(1, wdur\_a);

 end

 % transfer the data to this buffer:

 % note that this is really expensive computationally

 for j = 1:wdur\_a

 index = n\_left + (j - 1);

 if ((index > 0) && (index <= num\_samples))

 sig\_wbuf(j) = sig\_a(index);

 end

 end

 % square the signal. divide it by the number of samples used and sum

 % the result to build the value for that frame

 %

 rms = sqrt( (1 / wdur\_a) \* sum(sig\_wbuf.^2));

 for j = 1:fdur\_a

 index = n\_center + (j - 1) - (fdur\_a/2);

 if ((index > 0) && (index <= num\_samples))

 rms\_full(index) = rms;

 end

 end

 X(n\_center) = nanvar(sig\_wbuf);

end

for i = 2:length(X)

 if X(i) == 0

 X(i) = X(i-1);

 end

end

end

# Conclusions

Observing figures 3 and 4, we can see that the two approaches give us very different results, and in fact should be used for two different cases. The variance estimated in Part 2 (the red line) in both cases eventually coincides with the dotted blue line, or the variance of the entire signal. Yes, this approach may be useful if I wanted to see how the variance of the whole signal changed over time. However, in a real-time situation, we may not be interested in values from the beginning of the file; if we are predicting the price of Google stock tomorrow, we do not care much how much it cost two years ago. The frame/window approach provides a solution to this; it only looks at the most recent window of time and computes the variance, which is very useful if we are analyzing a signal as it is happening. In both figures 3 and 4, the black line (window/frame approach) oscillates, but generally stays centered on a value. That is not the case of the red line; in figure 2, you can see that it is increasing the entire time. Quelling the oscillations may be possible by choosing a smaller window, but they may be cause by the varying nature of the signal itself. Ultimately, the window/frame approach makes more sense, and was much faster to compute especially for the decently large speech file, which contained many points.