Dynamic Measurement and Signal Processing
Lab 6: Order Tracking

Guilherme Aramizo Ribeiro
PhD Student

Patcharapol Gorgitrattanagul
Teaching Assistant

Dr. Jason R Blough
Instructor

December 10, 2015

Abstract

Hello! ... Good! Yes!

Some comment

1 Background and Objectives

The purpose of this lab is to understand and apply order tracking algorithms in real data. A Fast Fourier Transform (FFT) based and a Time Varying Discrete Fourier Transform (TVDFT) are used to visualize the order of a test bench motor. This section will overview important aspects of rotating machine testing and the two order tracking methods.

As presented in [1], Order Tracking analyses the non-stationary frequency components associated to a rotating machine in operation. It differs from standard Digital Signal Processing (DSP) techniques due to the non-stationary nature of the frequency of interest. In other words:

$$X(t) = A(k,t)\sin(2\pi i(k/p)t + \Phi_k) \tag{1}$$

where A is the amplitude
Φ is the phase angle of order k
p is the period of primary order .
t is time
k is the tracked order

rotating machine overview

Order refers to the signal component whose frequency is a factor of the primary frequency. It can vary in amplitude and frequency over time. The amplitude variation causes errors in its tracking because all the order tracking methods consider a semi-constant amplitude over time.

The FFT order tracking .

background of FFT OT

Algorithm 1 FFT Based Order Tracking

1: $acc,\ tach \leftarrow sample(blocksize)$
2: Another line
3: Do this
4: Do that $a = 2$

$$R = \frac{V_{high} - V_{low}}{2^Q} \tag{2}$$

Show FFT equation

The TVDFT order tracking .

$$R = \frac{V_{high} - V_{low}}{2^Q} \tag{3}$$

background of TVDFT OT

Show TVDFT equation

2 Apparatus

The material used in the lab activity is listed bellow:

- Speed adjustable DC motor with an unbalanced mass;
- Tachometer;
- Accelerometer;

name
name
name

name

- Signal conditioner;

name

- Accelerometer Hand Calibrator;
- Data Acquisition NI cDAQ 9172;
- Data Acquisition NI 9234;
- Desktop computer with MATLAB, LabVIEW and device drivers installed ...

3 Experimental Procedures

A test bench motor with speed adjust is instrument with a tachometer sensor and mounted on a base. The interface between the base and the ground has a layer of foam that allows the platform to oscillate vertically. An 1-axis accelerometer is glued into the center of the platform.

insert calibrator name here

Initially, the accelerometer is calibrated with the . At 149.2 Hz the calibrator oscillates by $1gRMS$. The square rooted auto-power at 149.2 Hz, $A_{f=149.2\ Hz}$, was measured at X V. The calibration factor, c was then calculated as

fill value

$$c = \frac{1g}{A_{f=149.2\ Hz}} = \frac{X}{X} = X \tag{4}$$

fill value

The acquisition parameters were selected as to capture 32 blocks of 476 samples, measured at 1.651 kHz, calculate the auto-power with a Flat Top window. The window size is such that the vibrating frequency falls within the center of a frequency bin to avoid leakage. The sampling frequency is higher than the Nyquist rate.

On the center of the base, a 1-axis accelerometer The lab activity contains two components: an in-lab experimental procedure and a simulation performed with the MATLAB software. Both activities are described in detail in the lab manual, so this section describes the exceptions found in the activity.

It wasn't possible to decrease the signal generator output amplitude such that it would be mixed in the noise floor. Because the amplitude value is adjusted in discrete steps and the minimal value is well above the noise floor.

4 Data Summary

A sinusoidal signals with frequency of 356 Hz and amplitude of 1.24 V_{RMS} were acquired during the laboratory section. The time history of the experimental and MATLAB generated data was added to the Appendix, Figure 2, 3 and 4. The experiment run using a range of ± 5 V is presented in the frequency domain in the Figure 1.

Figure 1: Overlayed plot of experimental and simulated data in the frequency domain for ± 5 V of range and 8, 12 and 24 number of bits.

5 Interpretation and Analysis

Equation 3 demonstrates that a broad input range or low number of bits decrease the resolution. The time history of the simulation data confirms this statement: Figure 2 and 3 both show that the samples are closer to the nominal curve as the number of bits increase, and this distance is smaller in the experiment with the smaller range, ± 5 V. Also, voltage saturation was observed in the Figure 4, where the simulated signals saturates on ± 0.75 V.

Another point from Figure 1 is that the noise floor of the experimental data is higher than the simulated 24 bits counterpart, even thought the acquisition system also has a 24 bits precision. The experimental data also accounts with noises in the environment and in the DAQ electronics.

The oscilloscope's experimental SNR analysis was not possible because the signal generator couldn't reduce the sine amplitude low enough. But according to their respective data-sheet, the oscilloscope has a 8 bit resolution, while the acquisition board has 24 bits. And both systems have $\pm 5\ V$ as the narrower acquisition range. So according to Equation 3, the minimum signal amplitude detectable is 19 mV and 298 ηV, respectively. This follows that the NI system has higher SNR.

Finally, in respect to the critical thinking section, for a sine wave with period of $T = 56\mu s$ and RMS voltage of $A_{RMS} = 0.35\ V$, the desired sampling rate, F_s, voltage range, L, and number of bits, Q, for the requirements described in the manual are

$$F_s = 524288\ Hz \approx 2^{nextpow2(20/T)} \tag{5}$$

$$L = \pm 1\ V \approx \pm 2\ (A_{RMS}\ \sqrt{2}) \tag{6}$$

$$Q = 8 > log_2 20 \tag{7}$$

being $n = nextpow2(x)$ a function that returns the next power of 2 of x, or in other words, an integer n, $2^{n-1} < x \leq 2^n$.

6 Conclusions

On this assignment a physical signal was measured thought a data acquisition system in order to observe the effects of quantization. This data was also compared to simulated signals on MATLAB. Finally, the studies of acquisition parameters were explored with an example case.

From the frequency analysis of the discretized sine waves, higher SNR was observed in signals with higher number of bits and higher input voltage band usage. But experimental data has an inherent process noise floor that might be more significant than the quantization error, thus limiting the SNR to a maximum value. Check [2]. Testing saving capabilities.

References

[1] Jason R. Blough. "A Survey of DSP Methods for Rotating Machinery Analysis, What is Needed, What is Available". In: *Symposium on Emerging Trends in Vibration and Noise Engineering* (2001).

[2] Albert Einstein. "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]". In: *Annalen der Physik* 322.10 (1905), pp. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.

Appendices

A Effects of Number of Bits and Voltage Range

Figure 2: Time history of the experimental data and quantized simulated data for a ADC input range of $\pm 5\ V$

Figure 3: Time history of the experimental data and quantized simulated data for a ADC input range of $\pm 300\ V$

Figure 4: Time history of the experimental data and quantized simulated data for a ADC input range of $\pm 0.75\ V$

B Main Script

```
%% DSP - Laboratory #6
% Guilherme Aramizo Ribeiro
%% Definitions
% TODO replace with experimental data
rpm_nominal_ = 15000; % motor operating speed [rpm]
rpm_min_ = rpm_nominal_ / 5;
acc_sens_ = 3/(9.81*sqrt(2)); % V/(m/s^2)
rot_unbalance_ = [1e-3 1e-4 3.5e-4 2e-5];
orders_ = [1 2 5 7];
motor_func_ = @(t,w) sum(bsxfun(@times, orders_.*sqrt(rot_unbalance_), w).^2 .* ...
    sin(bsxfun(@plus, 2*pi*bsxfun(@times, orders_, w.*t), rand(size(orders_)))), 2);

%% Getting calibration factor
% Calibrating a signal
%     Input frequency range: [Fmin Fmax] = [149.2 149.2] Hz
%     Input voltage range: [Vmin Vmax] = ?
%
%     Fs = 13.1072e6/(256*31) > 300 Hz = 2*Fmax
%     winsize = 476, minimize leakage
%     noavg = 32

% acquisition parameters
Fs = 13.1072e6/(256*31);
winsize = 476;
noavg = 32;
range = 5;
bits = 24;

% TODO replace with experimental data
% Simulate signal
% time = linspace(0, (winsize*(noavg+2))/Fs, winsize*(noavg+2))';
% y = myDSP.discretize( ...
%     (9.81*sqrt(2))*acc_sens_*sin(2*pi*149.2*time), bits, range);
data = dlmread('../report_lab06/data/calib.csv', '\t');
time = data(:,1);
y = data(:,2);

%{
plot(time*1e3, y, 'o-')
xlim([0 30])
xlabel('time [ms]'); ylabel('acc voltage [V]')
%}
```

```
% Visualize auto-power
yreshaped = myDSP.reshape(y, winsize, 0);

win = window(@flattopwin, winsize);
gain = [1; 2*ones(winsize-1, 1)] / (mean(win)*winsize);

GY = bsxfun(@times, fft(bsxfun(@times, yreshaped, win)), gain);
Gyy = squeeze(mean(conj(GY) .* GY, 2));

f = linspace(0, Fs, winsize);

figure
subplot(211); semilogy(f, sqrt(Gyy)); grid on; grid minor
xlim([0 Fs/2]); xlabel('frequency [Hz]')
subplot(212); plot(time*1e3, y, 'o-'); xlabel('time [s]'); xlim([0 30])

calib_factor = (9.81*sqrt(2))/0.03417; % Calibration factor [(m/s^2)/V]

%% Set #1

% Steady state 0.5*rpm speed
%     Input frequency range: 10th order of the nominal speed
%     Input voltage range: [Vmin Vmax] = ?
%
%     Fs = 10240 = 13.1072e6/(256*5) > 2*Fmax = 2*(10*(rpm_nominal_/60))
%     winsize = 40960 = Fs/0.25, df = Fs/winsize = 0.25
%     noavg = 50

% acquisition parameters
Fs = 13.1072e6/(256*5);
winsize = 40960;
noavg = 50;
range = 5;
bits = 24;

% TODO replace with experimental data
% time = linspace(0, (noavg+2)*winsize/Fs, (noavg+2)*winsize).';
% rpm = rpm_nominal_ * ones(size(time)) / 2;
% tach = square(2*pi*cumsum(rpm/60).*diff([-1/Fs; time])) + randn(size(time))/10;
% acc = calib_factor * acc_sens_ * motor_func_(time, rpm/60);

data = dlmread('../report_lab06/data/ss.csv', '\t', 1);
time = data(:,1) - data(1,1);
tach = data(:,4);
acc = data(:,2) * calib_factor;
```

```
%{
figure
subplot(311); stairs(time, rpm); xlim([0 30e-3])
subplot(312); stairs(time, tach); xlim([0 30e-3])
subplot(313); stairs(time, acc); xlim([0 30e-3])
%}

yreshaped = myDSP.reshape(acc, winsize, 0);
yreshaped(:,[1 end]) = []; % remove head and tail

win = window(@hann, winsize);

gain = [1; 2*ones(winsize-1, 1)] / (mean(win)*winsize);

GY = bsxfun(@times, fft(bsxfun(@times, yreshaped, win)), gain);
Gyy = squeeze(mean(conj(GY) .* GY, 2));

f = linspace(0, Fs, winsize);

figure
semilogy(f, sqrt(Gyy)); grid on; grid minor
xlim([0 Fs/2]); xlabel('frequency [Hz]'); ylabel('base acceleration [m/s^2]')

%% 1)
avg_rpm = mean(myDSP.speed_from_tach(time, tach)); % Average RPM

%% Set #2, #3, #4

% Transient: 0->45s, 0->20s, 0->10s, ranging full rpm bounds
%     Input frequency range: 10th order of nominal speed
%     Input voltage range: [Vmin Vmax] = ?
%
%     Fs = 10240 = 13.1072e6/(256*5) > 2*Fmax = 2*(10*(rpm_nominal_/60))
%     winsize = 40960 = Fs/0.25, df = Fs/winsize = 0.25
%     noavg = 50

% acquisition parameters
Fs = 13.1072e6/(256*5);
len = 45*Fs;
% winsize = 40960;
% noavg = 50;
range = 5;
bits = 24;

% time = linspace(0, round(len*1.1)/Fs, round(len*1.1)).';
%
```

```
% rpm_ = ones(size(time, 1), 3) * rpm_nominal_;
% rpm_(time < 45, 1) = rpm_min_ + time(time < 45)*(rpm_nominal_-rpm_min_)/45;
% rpm_(time < 20, 2) = rpm_min_ + time(time < 20)*(rpm_nominal_-rpm_min_)/20;
% rpm_(time < 10, 3) = rpm_min_ + time(time < 10)*(rpm_nominal_-rpm_min_)/10;
%
% tach = square(2*pi*bsxfun(@times, cumsum(rpm_/60), diff([-1/Fs; time])));
%
% acc = calib_factor * acc_sens_ * [motor_func_(time, rpm_(:,1)/60), ...
%     motor_func_(time, rpm_(:,2)/60), motor_func_(time, rpm_(:,3)/60)];

data1 = dlmread('../report_lab06/data/slow3.csv', '\t', 1);
data2 = dlmread('../report_lab06/data/mid2.csv', '\t', 1);
data3 = dlmread('../report_lab06/data/fast2.csv', '\t', 1);

len = max([size(data1,1), size(data2,1), size(data3,1)]);
time = (1:len).'/Fs;

tach = zeros(len, 3);
acc = zeros(len, 3);

tach(1:size(data1,1),1) = data1(:,4);
tach(1:size(data2,1),2) = data2(:,4);
tach(1:size(data3,1),3) = data3(:,4);

acc(1:size(data1,1),1) = data1(:,2) * calib_factor;
acc(1:size(data2,1),2) = data2(:,2) * calib_factor;
acc(1:size(data3,1),3) = data3(:,2) * calib_factor;

plot(time, tach)



%% 2) Tachometer
rpm = [myDSP.speed_from_tach(time, tach(:,1)), myDSP.speed_from_tach(time, tach(:,2)), ...
    myDSP.speed_from_tach(time, tach(:,3))];

plot(time, rpm); grid on; grid minor
xlabel('frequency [Hz]'); ylabel('Shaft speed [RPM]'); ylim([0 5000])


%% 3) Color map
noblocks = 100;

time_range = [45 20 10];
for k = 1 : 3
    winsize = round((Fs*time_range(k))/noblocks); % 100 evenly spaced RPM bins
```

```
    y = acc(:, k);

    yreshaped = myDSP.reshape(y, winsize, 0);
    yreshaped = yreshaped(:,(1:noblocks) + 1); % remove head and tail

    win = window(@hann, winsize);

    gain = [1; 2*ones(winsize-1, 1)] / (mean(win)*winsize);

    GY = bsxfun(@times, fft(bsxfun(@times, yreshaped, win)), gain);
    Gyy = filter(ones(1,5)/5, 1, conj(GY) .* GY, [], 2);

    f = linspace(0, Fs, winsize);
    [freqs, rpms] = meshgrid(f, linspace(rpm_min_, rpm_nominal_, 100));
    accs = sqrt(Gyy).';

    figure;
    surf(freqs, rpms, accs, 'EdgeColor','None');
    view(2); xlim([0 Fs/2]); ylim([rpm_min_, rpm_nominal_])
    colorbar; xlabel('Frequency [Hz]'); ylabel('shaft speed [RPM]')
end

%% 4) Order tracking
%% FFT, constant order bandwidth
noblocks = 100;

time_range = [45 20 10];
for k = 1 : 3
    winsize = round((Fs*time_range(k))/noblocks); % 100 evenly spaced RPM bins

    angle = cumsum(rpm(time<time_range(k),k)*(2*pi/60));
    R = angle(end) / noblocks;

    angle = cumsum(rpm(:,k)*(2*pi/60));
    RR = angle(end) / noblocks;

    xx = cumsum(rpm(:,k));
    xx = xx * t(end)/xx(end);
    y = interp1(time, acc(:, k), xx);

    yreshaped = myDSP.reshape(y, winsize, 0);
    yreshaped = yreshaped(:,(1:noblocks) + 1); % remove head and tail

    win = window(@hann, winsize);

    gain = [1; 2*ones(winsize-1, 1)] / (mean(win)*winsize);
```

```
    GY = bsxfun(@times, fft(bsxfun(@times, yreshaped, win)), gain);
    Gyy = filter(ones(1,5)/5, 1, conj(GY) .* GY, [], 2);

    f = linspace(0, winsize/R, winsize);
    [freqs, rpms] = meshgrid(f, linspace(rpm_min_, rpm_nominal_, 100));
    accs = sqrt(Gyy).';

    figure;
    surf(freqs, rpms, accs, 'EdgeColor','None');
    view(2); ylim([rpm_min_, rpm_nominal_])
    colorbar; xlabel('Frequency [Hz]'); ylabel('shaft speed [RPM]')
end
```

C Auxiliary DSP Functions

```
classdef myDSP
methods(Static)

function [SIGNAL, nowin] = reshape(signal, win_size, overlap_p)
%RESHAPE Reshape vector in window blocks, with overlaps
%     Example:
%     % Generate data
%     % Define dynamic system
%     sys = rss(3, 2, 1);
%     Fs = -100*min(real(eig(sys.A)));
%     N = 10000;
%
%     time = linspace(0, N/Fs, N).';
%     u = reshape(repmat(randn(round(N/20),1), [1 20]).', [N 1]);
%     y = lsim(sys, u, time);
%
%     winsize = 1000;
%     overlap = 0.3;
%     yreshaped = myDSP.reshape(y, winsize, overlap);
%     ureshaped = myDSP.reshape(u, winsize, overlap);
%
%     win = window(@hann, winsize);
%     gain = [1; 2*ones(winsize-1, 1)] / (mean(win)*winsize);
%
%     GY = bsxfun(@times, fft(bsxfun(@times, yreshaped, win)), gain);
%     Gyy = squeeze(mean(conj(GY) .* GY, 2));
%
%     GU = bsxfun(@times, fft(bsxfun(@times, ureshaped, win)), gain);
%     Guy = squeeze(mean(bsxfun(@times, conj(GU), GY), 2));
%
%     f = linspace(0, Fs, winsize);
%
%     figure
%     subplot(311); semilogy(f, sqrt(Gyy)); grid on; grid minor
%     xlim([0 Fs/2]); xlabel('frequency [Hz]')
%     subplot(312); semilogy(f, sqrt(abs(Guy))); grid on; grid minor
%     xlim([0 Fs/2]); xlabel('frequency [Hz]')
%     subplot(313); plot(time, y); xlabel('time [s]')

    len = size(signal, 1); % signal length
    nocha = size(signal, 2); % number of channels

    if win_size > len
```

```
        error('Window size is greater than signal size')
    end
    if overlap_p > 1 || overlap_p < 0
        error('Overlap should range between 0 and 1')
    end

    overlap = round(overlap_p*win_size);
    nowin = floor((len-win_size)/(win_size-overlap) + 1); % # of windows

    SIGNAL = zeros(win_size, nowin, nocha);
    for k = 1 : nocha
        % make overlap multiple of win_size, percentual to absolute
        idx = repmat((1:win_size).', [1 nowin]) + ...
            repmat((0:nowin-1)*(win_size-overlap), [win_size 1]);
        SIGNAL(:,:,k) = reshape(signal(idx,k), [win_size, nowin]);
    end
end


function Y = discretize( U, N, V )
    % Discretize vector U on the range +-V into N bits

    U = uencode( U, N, V, 'signed');

    U = cast( U, 'double' );
    N = cast( N, 'double' );
    V = cast( V, 'double' );

    Y = interp1( [ -2^(N-1) 2^(N-1)-1 ], [-V V], U );
end

function [blocksize, noc] = optimize_blocksize(F0, Fs, noc_range, Fmin)

    min_time = 2/Fmin;
    min_blocksize = min_time/Fs;

    nocs = noc_range(1) : noc_range(2);
    approx_blocksizes = nocs * Fs/F0;
    % disregard too small block sizes
    approx_blocksizes(approx_blocksizes < min_blocksize) = [];
    if isempty(approx_blocksizes)
        error('Slower component do not fit window choices')
    end
    blocksizes = round(approx_blocksizes);

    [err, idx] = min(abs(approx_blocksizes - blocksizes));
```

```
    blocksize = blocksizes(idx(1));
    noc = blocksize * F0/Fs;

    if err > 1e-3
        warning(['Bad block size. Round error: ' num2str(err)])
    end
end

function rpm = speed_from_tach(time, tach)

    len = length(tach);
    tach = reshape(tach, [len 1]);

    thres = (prctile(tach, 75) - prctile(tach, 25))*0.6;
    trigger = [false; tach(2:end) - tach(1:end-1) > thres];
    time_rising = time(trigger);

    bad = zeros(size(time_rising)) > 0;
    avg_dt = time_rising(2) - time_rising(1);
    for k = 2 : length(time_rising)
        if time_rising(k)-time_rising(k-1) < 0.5*avg_dt
            bad(k) = true;
        else
            avg_dt = avg_dt*0.5 + 0.5*(time_rising(k)-time_rising(k-1));
        end
    end
    triggerb = find(trigger);
    time_rising = time(triggerb(~bad));

    w = 2*pi./(time_rising(2:end) - time_rising(1:end-1));
    w = [w; w(end)];

%     plot(time, tach, time(trigger), tach(trigger), 'o', time(triggerb(bad)), tach(trigge
    rpm = interp1(time(triggerb(~bad)), w, time, 'linear', 'extrap') * 60/(2*pi);
end

end
end
```