Engineering Tripos Part IIA Module Experiment: 4C7

Nonlinear Vibration, South Wing Mechanics Laboratory

Objectives

- To explore nonlinear effects due to high forcing amplitudes about resonance in mechanical structures.
- To construct a model to explain the nonlinear behaviour of the mechanism provided and use this model to simulate the behaviour of the system under specified conditions.
- Demonstrate use of the sonogram (time-varying spectrum) as an analytical tool to distinguish both frequency and temporal characteristics of a transient record.

Introduction

In many instances a linear model is sufficient to approximate the behaviour of physical systems. However, in particular instances the linear model breaks down and a model with further complexity must be used. We will use the principles of nonlinear vibration theory described in the lectures to model the observed behaviour of a clamped-clamped beam subjected to high driving forces about the primary resonance.

This experiment is in two parts:

Part A: Measurement of the vibratory response of a clamped-clamped beam to variable drive forces.

Part B: Modelling and simulation of the nonlinear system.

Report

Your report should describe the key results of the experiment, with plots when appropriate. A number of specific questions need to be addressed as outlined in this handout. The completed report should be handed in as a PDF document through the coursework submission portal on the 4C7 moodle site in advance of the final lecture on Wednesday, December 2.

Apparatus

The apparatus comprises of a thin aluminium beam clamped at two ends to prevent axial displacement. Such a system exhibits a spring hardening effect (comment as to why this is the case in your report) with growing displacement amplitude for sufficiently large force inputs. A shaker is used to apply driving forces to the structure. The shaker derives its input from a PC-generated sine-wave (filtered by a low-pass filter) that can be varied in both frequency and amplitude. A National Instruments Data Acquisition Card (NI-DAQ) converts the specified signal to an analogue output, which is low-pass filtered, passed through channel 1 of an oscilloscope, and power-amplified by a current amplifier which provides the input to the shaker. An accelerometer is placed at the centre of the beam to measure the response of the structure to the applied drive input. The accelerometer output is coupled to an amplifier whose output is captured by the NI-DAQ using a Python package that this notebook uses and is described below. The PC will be used to drive the shaker and capture data from the accelerometer. Measurement data will be saved to your teaching system drive where it will be analysed and used for comparison with simulation. If you choose, you can analyse the data at any of the South Wing lab PCs or your own device (installation instructions at the end of this notebook). Please ensure that you have read through and signed up to the risk assessment for this lab prior to commencing this experiment.

Part A: Experimental procedure

A1: Linear response

Start by double-clicking on the 'Start 4C7.bat' file on the desktop. This copies across a fresh copy of this notebook and a python script file to the folder 'lab_4C7' in your teaching system filespace, then it opens Jupyter Notebooks starting in this folder. You will likely see some error messages in the command prompt: these are normal and are just part of the system looking in different directories for your teaching system username. If you need to (e.g. if you accidentally deleted part of this notebook), then you can get a fresh copy by running 'Start_4C7.bat' again.

Click '4C7_Notebook.ipynb' to open.

The next cells import the necessary modules: pydvma is a python package written for data acquisition at CUED.

Remember:

- to actually run a cell of code, click inside the cell then press 'shift+enter'
- the cell is running while [*] is displayed
- the cell has finished running when it changes to a number
- under the 'view' menu above you can toggle off the header and toolbar, which is useful to give a bit more screen space
- you can control the cell output view by clicking on the left beside the cell: single click to expand/compress, double-click to hide.

```
In [ ]: %gui qt
```

Wait a second before running the imports: it takes a moment for the previous command to take effect behind the scenes. If you get an error, just try it again.

```
In [ ]: import matplotlib
        import numpy as np
        import pydvma as dvma
```

```
In [ ]: matplotlib.use('nbagg')
```

The first stage of the experiment is to measure the linear response due to a sweep input in force. This involves varying the force supplied by the shaker in frequency while fixing the amplitude at a sufficiently low forcing level to measure the linear response while sweeping upwards and downwards in frequency. Now open the logger window by running the next cell (shift+enter), with the following suggested settings:

- channels=2 (number of channels to record)
- fs=1500 (sampling rate in Hz)
- pretrig_samples=100 (starts logging 100 seconds before the signal starts being generated)
- stored_time=10 (time in seconds to store data)
- device_driver = 'nidaq'

```
In [ ]: # acquisition setup
        settings = dvma.MySettings(channels=2,
                                   fs=1500,
                                   stored_time=10,
                                   pretrig_samples=100,
                                   device_driver = 'nidaq')
        logger1 = dvma.Logger(settings)
```

Familiarise yourself with the logger window:

- the centre panel contains the figure - note the zooming / panning tools under the figure.
- the left panel controls what is plotted (you can select which plot to show using the top dropdown)
- the right panel provides data analysis tools (you can select which tools are displayed using the top dropdown)
- the centre top panel is for basic data management (logging / loading / deleting)
- the centre bottom panel is to save data or figures.

Although the logger is set up to record data, we need to choose an output signal to generate. Use the "Tool Selection" dropdown at the top right to choose "Generate Output". Set the following options:

```
Test name:    'up_0.2' (for example, noting that this will appear in the legend)
Type:         'sweep'
Amplitude:    0.2
f1 (Hz):      40
f2 (Hz):      70
Duration (s): 9 (so that it finishes one second before logging finishes)
```

Press "Preview Output" to check it's what you expect to see.

Press either "Log with Output" or simply "Log Data": the generated signal will be sent to the NI-DAQ card to output to the amplifier and the data from the input channels will be logged simultaneously.

You will then see the input and output signals displayed, including data for 1 s after the sweep excitation finished. A precaution: check that the data logging program has recorded the input signal from the accelerometer for the entire duration of the sweep.

Please note three important things:

- The amplitude of the signal generated is a normalised value in the range 0 to 1. Similarly the input logged amplitude is normalised to the range -1 to +1. To get the actual voltages you multiply by 5: but you don't need to do that for the lab because the factor for the input and output are the same, and in any case you will calculate an overall scale factor for comparison with simulations later.
- If the logger becomes unresponsive, or output signal generation has stopped working, then you can restart this notebook by selecting 'Restart & Clear Output' from the 'Kernel' menu of the notebook.
- Within the logger, press "Save Dataset" and "Save Figure" to save your measurements and displayed figures to your directory for this experiment under your teaching system account. Choose a file name that describes this particular sweep: you will have saved several such files by the end of the experiment.

To extract natural frequency and damping parameters for the system, run a fast Fourier transform (FFT) on the captured data by choosing "Standard Tools" from the tool selection dropdown (top right), then pressing "Calculate FFT". Then calculate the transfer function by pressing "Calculate TF'. The window and averaging options here should be set to "None".

Zoom in on the frequency range of interest (40 - 70 Hz) by entering the axis limits in the left panel or by using the zoom controls under the plot. The "Auto Y" button (left panel) can be used to set the amplitude range on the y-axis appropriately.

To reset the figure axes at any stage, press 'Auto X' then 'Auto Y'.

Is the transfer function shaped as you would expect? Zoom in on a frequency range corresponding to amplitudes running from 5-10 dB on either side of the resonant frequency. In the left panel, use the plot type dropdown to choose a "Nyquist" plot. Then choose "Mode Fitting" from the tool selection dropdown (top right) which will fit a circle to the Nyquist plot. Extracted parameters such as the damping factor and resonant frequency will appear in the message area. (This method will be familiar to those also taking 4C6: the rest should just believe it!)

Now repeat the above procedure but sweep downwards in frequency by setting f1 to 70 Hz and the f2 to 40 Hz.

Again save your data and figures in the same folder under a different filename. Do you notice any change in the measured parameters?

Understanding the logger: When you press 'Save Dataset', the logger saves a <DataSet> object which has the following structure:

```
<DataSet> class:
      time_data_list: [<TimeData>, <TimeData>, <TimeData>]
      freq_data_list: [<FreqData>, <FreqData>, <FreqData>]
cross_spec_data_list: []
        tf_data_list: [<TfData>, <TfData>, <TfData>]
     modal_data_list: [<ModalData>]
      sono_data_list: []
      meta_data_list: []
```

Each type of data is arranged in 'sets', e.g. each measurement will add another <TimeData> set to time_data_list. This example has three sets of time, frequency and transfer function data types, and a <ModalData> set (which you will use later). Similarly each time you load data it will add the loaded sets to the appropriate data list.

- When you press 'Calc FFT' then a <FreqData> item is calculated for each <TimeData> set.
- When you press 'Calc TF' then a <TfData> item is calculated for each <TimeData> set.
- When you press 'Calc TF average' then a single <TfData> item is calculated averaging across all <TimeData> items, with an assumption that the time data sets all have the same settings.

A2: Nonlinear response

Repeat the procedure outlined above performing upward and downward frequency sweeps for the forcing input for the same frequency range but for **different values of forcing amplitude**. The force amplitude is directly proportional to the amplitude of the sine wave generated by the computer. Use the "Tool Selection" dropdown at the top right to choose "Generate Output". Set the following options:

```
Test name:    'up_0.2' (choose something systematic and compact: this will appear in the legend!)
Type:         'sweep'
Amplitude:    ... (choose between 0 and 1)
f min (Hz):   40
f max (Hz):   70
Duration (s): 9 (so that it finishes one second before logging finishes)
```

NOTE: if you have closed the logger window, don't worry, you can re-open it using the following at any time:

In [ ]:
```
logger1.show()
```

For each amplitude and sweep direction:

1. Press "Delete All" to clear the previous data.

- Change the output signal settings using the "Generate Output" tool - **don't forget to change the test name** to keep track of the cases.
- Press "Log Data"
- Press "Calculate FFT" and "Calculate TF" (no window, no averaging)
- Use the "Mode Fitting" tool, zoom into the peak, and press "Fit" to obtain the modal estimates (keep a note of these results)
- Press "Save Dataset" and "Save Figure"
- Press "Delete All" and repeat

Run the experiment for sweep amplitudes equal to 0.6 and 1 (and remember to change the test names accordingly to help you later). Especially for the runs with higher amplitude, you should hold the accelerometer wire gently away from the vibrating beam so that it does not create unwanted interference by buzzing. If something is wrong with one of the measurements, then press "Delete Last" or "Delete All" and start again.

Note that each time you press "Log Data", the new measurement is added to the existing set of data: you need to **save each measurement separately**, using "Delete All" before logging the next test case. This lets you estimate the modal parameters for each test case independently (if you had logged all data into a single dataset the mode fitting tool would try to find a single frequency and damping factor for the whole set).

Once you have obtained an estimate of the modal parameters as a function of amplitude, then you can combine the datasets:

1. Press "Delete All"

- Use "Load Data" repeatedly to combine all the sets of data (load the data in a sensible order so you know which set is which)
- Use the "Figure Selection" dropdown (top left) to choose which plot to view.
- Press "Save Dataset" to save the combined data, and "Save Figure" to save the plots you want to keep for your report.

Don't forget to save your data and figures to the teaching system.

Pay particular attention to the shape of the transfer function for each sweep. You can use the buttons in the left panel and legend lines to select particular data to show. Compare the upward and downward sweeps for the same amplitudes. Now compare the plots for varying amplitudes. Comment on your results.

CHECK 1: Have you remembered to save your measurement datasets separately for each test case, with correct test names?

CHECK 2: Have you remembered to save the plots you need to keep for the report?

CHECK 3: Have you remembered to save the combined dataset of all test cases?

Part B: Modelling and Simulation

Python code is provided below that simulates the behaviour of a single-degree-of-freedom vibratory system with a Duffing nonlinearity.

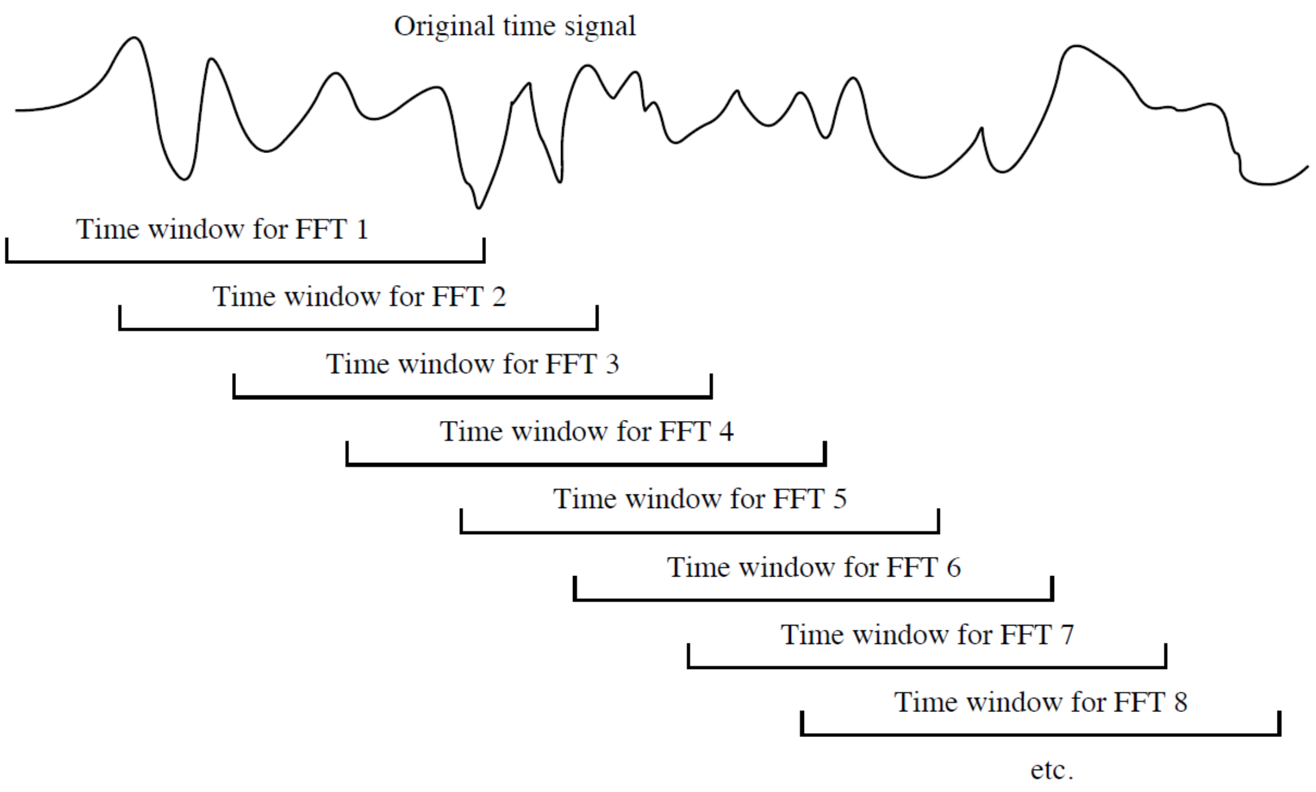
First, a model for the linear case (sweep amplitude = 0.2 in normalised units) will be developed. Let us describe the linear single-degree-of-freedom case by the standard equation:

$$\ddot{y} + 2\zeta p\dot{y} + p^2 y = f(t)$$

In the above equation, the natural frequency p and damping coefficient ζ are given directly by the results of modal-fitting from your first test. Don't forget that p is in radians/s. At a given point in a sweep, the forcing signal takes the form $f(t) = \lambda a\cos(\omega t + \phi)$, where ϕ represents an arbitrary phase of the input signal, a is the chosen amplitude in normalised units, and λ is a scaling factor which describes the ratio of calibration factors for the two channels of measurement (acceleration and force). Using your values for natural frequency p and damping coefficient ζ, estimate the value of the scaling factor λ from the peak output level from the time-series data in your first experiment (for which nonlinear terms can be neglected). Remember that your data is recording acceleration rather than displacement.

In [ ]:
```
# you can use this cell to do any calculations needed
# don't use the variable name 'lambda' as that has it's own special meaning in python!
L = ...
```

Next, to construct the non-linear model, we will examine the experimental data more carefully by conducting a 'sonogram' analysis (also called a 'spectrogram'). A sonogram is a time-varying spectrum, illustrated below. A short section of the time history is copied and Fourier analysed — 'Time window for FFT 1'. Then a section of the same length starting a little later in time is copied and analysed similarly ('FFT 2'), then another and another until a sufficient length of the whole time history has been covered.



By plotting in some suitable way this sequence of frequency spectra, we can now look at the response as a function of BOTH frequency and time (in much the same way that your brain processes sounds which you hear).

For simplicity, close and start a new logger:

```
In [ ]: logger1.close()
        # acquisition setup
        settings = dvma.MySettings(channels=2,
                                   fs=1500,
                                   stored_time=10,
                                   pretrig_samples=100,
                                   device_driver = 'nidaq')
        logger2 = dvma.Logger(settings)
```

Now use the 'Load Data' to load your tests that you have already saved above. Start by loading the 'up' sweeps for the three amplitude levels tested.

Using the 'Tool Selection' dropdown (top-right), choose 'Sonogram'. There are several options to choose:

- 'N frames' is the number of windows used for the calculation
- the slider is linked to 'N frames'
- 'Dynamic Range (dB)' determines the colour scale range: the darkest shade will be the peak log-amplitude; the lightest shade will represent the 'dynamics range' less than the peak. Higher dynamic range shows more data, but also more noise. 60-80 dB is usually a good choice, but feel free to experiment to get the clearest plots.
- 'Set / Chan' chooses the time data that corresponds to the displayed sonogram.

Press 'Calc Sonogram' to do the calculation and show the sonogram: time is on the x-axis and frequency is on the y-axis. Note that the sonogram of all time data is computed, but 'Set / Chan' controls which is displayed. The sonogram will live-update as you change the options. If you have lots of time data channels loaded then this may become slow, in which case use the text entry fields rather than the slider.

You should experiment with the options to obtain the clearest picture. The frequency sweep should be clearly visible. You can adjust the sonogram axes using similar controls to the time display view: choose a range which brings out the important features clearly, and stick to the same ranges for any cases you wish to plot and include in your report, so that comparisons can be made easily.

Construct sonograms for each of the upward and downward sweeps for the measured data, by selecting 'Set / Chan' indices corresponding to the loaded time-data.

How do the sonograms compare? Notice that for a cubic spring nonlinearity as indicated by the Duffing equation, one should observe a third harmonic component on the sonogram plots with increasing amplitude of forcing input. Does the third harmonic component increase with applied input forcing? You will also notice that the sonograms for the measured data indicate the presence of a second harmonic component in the output displacement. Comment upon the possible sources for the appearance of the second harmonic in the measured data. Does the second harmonic component increase with applied forcing amplitudes?

Motivated by the results of the sonogram analysis, the following equation is proposed to model the non-linear single-degree-of-freedom vibratory system:

$$\ddot{y} + 2\zeta p\dot{y} + p^2 y + \alpha y^2 + \mu y^3 = \lambda a \cos(\omega t + \phi)$$

Here, the parameters p, ζ and λ are the same as for the linear model. Form two non-dimensional groups using the variables p, y, α , and μ. Use this to estimate the values of α, and μ for which the effect of quadratic and cubic nonlinearities are expected to be comparable to the linear terms (and remember that you are measuring acceleration, not displacement!). Given that in this experiment the effect of nonlinearity is expected to be relatively weak, make a first guess of their values. Do not be surprised if the numerical values for α and μ seem rather extreme: try them and see.

```
In [ ]: # you can use this cell to do any calculations needed
        alpha = ...
        mu = ...
```

You are now set up to commence computer simulations of the non-linear system. If you are starting this notebook at this point (e.g. if you have restarted the kernel), then re-load the logger package plus the numerical integration package.

```
In [ ]: %gui qt
```

```
In [ ]: import numpy as np
        import pydvma as dvma
        from scipy import integrate
```

Read the commented code bleow. This is provided to you as a starting point for your simulations. You should go through the code carefully to make sure that you fully understand the syntax. The code simulates the force sweep that was conducted in the experiment for a system described by a Duffing non-linearity.

Modify your code to include a quadratic spring term and incorporate the particular values of the extracted parameters into your model.

Run the code by pressing 'shift-s' to execute the cell. You will be able to run upward and downward sweeps as in your experiment. Run simulations for all values of input forcing that correspond to your experimental setup and save the data obtained for each of the cases for further analysis. You will also be able to run a sonogram analysis on the simulated data using the same procedure as for the experimental data.

Compare the simulation results carefully with your range of measurements, and iterate the parameter values of the simulation as necessary to obtain a best match. You only need to compare the responses (not the input force), i.e. channel 1, as your scale factor λ was an overall factor from the selected amplitude a to the output response, so the force amplitudes won't match.

Your report should describe and illustrate how you carried out this process of obtaining best-fitting parameters: do not simply present the final numerical results. Comment on any discrepancies between simulation and experiment.

What other sources of nonlinearity might be operative in this experiment? Discuss briefly what characteristic effects you would expect these would produce, and whether any of these are strong candidates to account for the results of the experiment.

```
In [ ]: # define the equation of motion to integrate, as a function in first-order form dydt=g(y)
        def model(y,t,p,z,L,a,f1,f2,T,alpha,mu):
            '''
            y,t is expected by the odeint solver: y is a vector where y[0] is displacement, y[1] is veloci
        ty
            p = linear natural frequency
            z = linear damping factor
            L = lambda
            a = amplitude
            f1,f2 = frequency start/stop for simulation
            T = time at which f2 is reached
            alpha = quadratic nonlinearity parameter (won't do anything until you add it into equation of
         motion below)
            mu = cubic nonlinearity parameter
            '''

            # define external force
            freq = f1 + (f2-f1)/2/T*t # analytic expression needed for linear sweep
            if t<T:
                # forced response until f2 reached
                force = L*a*np.cos(2*np.pi*freq*t)
            else:
                # unforced response after f2 reached
                force=0

            ### Equations of motion in first-order form ###
            dydt = np.zeros(2)
            dydt[0] = y[1]
            dydt[1] = force - 2*z*p*y[1] - (p**2)*y[0] - mu*y[0]**3

            return dydt
```

```
In [ ]: #####################################
        ### YOU NEED TO FIND THESE VALUES ###
        #####################################
        p = 2*np.pi*...  # natural freq, from linear tests, in rad/s
        z = ...  # damping factor, from linear tests
        L = ...  # lambda, scaling factor identified earlier
        a = ...  # normalised amplitudes for generating output signals

        alpha = ...
        mu = ...
        #####################################

        # setup integration parameters
        fs  = 3000 # match experiments
        T   = 9    # match experiments
        f1 = 40    # match experiments
        f2 = 70    # match experiments

        time_axis = np.arange(0,10,1/fs) # match experiments (logged for 10s)
        y0 = np.zeros(2) # zero initial conditions

        # solve ODE
        y = integrate.odeint(model,y0,time_axis,args=(p,z,L,a,f1,f2,T,alpha,mu))
```

In [ ]:
```
# recalculate force and acceleration
freq = f1 + (f2-f1)/2/T*time_axis # analytic expression needed for linear sweep
force = a*np.cos(2*np.pi*freq*time_axis)
force[time_axis>T]=0
dydt = np.zeros((len(time_axis),2))
for i in range(len(time_axis)):
    dydt[i,:] = model(y[i,:],time_axis[i],p,z,L,a,f1,f2,T,alpha,mu)
acc = dydt[:,1]

# put into format compatible with pydvma
# channel 0 corresponds to input force (as per experiment)
# channel 1 corresponds to acceleration (as per experiment)
data = np.zeros((len(time_axis),2))
data[:,0] = force
data[:,1] = acc
settings = dvma.MySettings(fs=fs,
                           stored_time=10,
                           channels=2)
time_data = dvma.TimeData(time_axis,data,settings)
d = dvma.DataSet(time_data) # create a pydvma dataset
```

When you have run a simulation, save the data using the following:

In [ ]:
```
d.save_data()
```

Then to open it in the logger and compare with experimental data, start a new logger and load the numerical and corresponding experimental data files:

Note: you don't have to start a new logger each time, if preferred you can 'Delete All' and load the next set for comparison.

In [ ]:
```
# acquisition setup
settings = dvma.MySettings(channels=2,
                           fs=1500,
                           stored_time=10,
                           pretrig_samples=100,
                           device_driver = 'nidaq')
logger3 = dvma.Logger(settings)
```

Note that you can re-open any of the logger instances you have previously started, and it will re-open where you left off including with any data so far stored. For example, to restart logger2 use:

In [ ]:
```
logger2.show()
```

AT THE END OF THE EXPERIMENT

- **check you have saved your data and figures to the teaching system**
- share your data with lab-group partners (e.g. using Firefox Send (https://send.firefox.com/) or wetransfer (https://wetransfer.com/))
- sign-out of the pc

NOTE:
- **Please note that the state of the loggers is not saved along with the notebook itself, so when you close the notebook then you can no longer use logger.show().**
- **Therefore, please remember to save data and figures as needed for the report before leaving the lab**
- **The pydvma logger is opensource, so you can install it on your own device (windows/mac/linux), however it is still in development so may not work.**

Installation instructions

1. Install anaconda by downloading from https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)

- Open Anaconda prompt and type:

```
conda install pyaudio
pip install pydvma
```

- If you get an error about not having the correct permissions, then try opening Anaconda prompt by right-clicking and choosing 'run as administrator'
- then open Jupyter notebooks from the windows start menu, and this notebook should work.

```
In [ ]:
```