Fast Fourier Transform, Curve Fitting, and Ordinary Differential Equations
Project #3

Due on Friday, March 20, 2015 (*extended)*

Inst. Pavel Snopok, Spring 2015

Michael Lee

March 20, 2015

Abstract

The Fourier transform is an indispensible tool used to retrieve pertinent information from signals. Its ability to utilize the frequency domain allows us to filter, encode, decode, and recreate information from various types of sources. It is particularly useful in signal processing where Fourier transforms can clean signals by reducing contaminating noises. Filtering, in signal processing, essentially works like this: given an input from an information processing system, the electronic component or software will use a form of the Fast Fourier Transform (FFT) to convert the input signal into the phase domain, retrieve the frequency, amplitude, and phase information, and then reconstruct the signal with the original parameters in the time domain. In addition to the Fourier analysis, we also examine how we can use programs to solve a system of ordinary differential equations. One of the simplest systems that gives us valuable insights in computational physics is the system of a mass connected to a spring driven by a external force. We shall observe how different driving force strengths and frequencies can result in completely different solutions to the equation of motion.

1 Introduction

1.1 Noisy Signal

In this heuristic simulation, we obtain a noisy signal sample and will reconstruct the original signal by resolving the original frequency, amplitude, and phase. We then compare the two signals.

We expect three things in this exercise:

1. The peaks of the Fast Fourier Transform should be evident.

2. If there are multiple peaks, it means the signal is a linear combination of multiple signals of respective frequencies.

3. The amplitude(s) of the original signal(s) should be directly related to the number of samples of the peak frequency divided by the total number of samples.

The Fast Fourier Transform is an algorithm that computes a fourier transform in $O(nlog(n))$ as opposed to the Discrete Fourier Transform (DFT) which has a time complexity of $O(n^2)$. Due to its common use in communications and 100-fold improvement over the DFT, the FFT is considered to one of the ten most important algorithms in computer science. It utilizes a divide and conquer algorithm that breaks the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{2\pi nk\frac{n}{N}}$$

into pieces where $N = N_1 N_2$ so that

$$X_k = \sum_{n=0}^{N_1-1}\sum_{n=0}^{N_2-1} x_n e^{2\pi nk\frac{n}{N_1 N_2}}$$

The algorithm divides the DFT into two groups, transforms the first group leaving the second group untransformed, and then recursively divides the second group, continuing the same steps until all the data is transformed. This process leads to a reduction in the number of computational steps because of the periodicity of $e^{2\pi nk\frac{n}{N}}$, meaning that terms can be collected from each evaluation of a group because of repeated values. This leads to the *log(n)* term due to the symmetry, and as such, the FFT can reduce each multiplicative term since they differ by either adding or subtracting appropriate x_i and x_j for N_i and N_j, making the overall multiplicative steps smaller than $\sum_{n=0}^{N-1} x_n e^{2\pi nk\frac{n}{N}}$ without dividing and conquering.

1.2 Signal from a Mass and Spring

Through Newton's second law, we have for the mass and spring system:

$$m\frac{d^2x}{dt^2} = F_k + F_{ext} \tag{1}$$

where F_k is the restoring force of the spring and F_{ext} is an external driving force.

The goal is to solve the second order differential equation when there is no external driving force. This is equivalent to finding the equation of motion of a simple harmonic oscillator while neglecting any dampening forces. For our restoring force, we have

$$F_k = -kx, \quad k = m\omega^2.$$

Our differential equation becomes

$$\frac{d^2x}{dt^2} = -\frac{kx}{m} = -\omega^2 x$$

which has the solution

$$x(t) = Csin(\omega t)$$

$$x(0) = C \Rightarrow x(t) = x_0 sin(\omega t).$$

Similarly

$$\frac{dx(t)}{dt} = x_0 \omega cos(\omega t) = v_0 cos(\omega t).$$

The initial conditions given are that x_0 equals the amplitude of the signal in part one and $v_0 = 0$. The solution we expect to see for the position as a function of time is a sine curve. If we add a driving force, the solution will be a linear combination of the solutions to the functions, F_k and F_{ext}. Depending on the magnitude and frequency of the external force, the solution may have extremely different results.

If we have for a driving force

$$F_{ext} = F_0 sin(\omega_1 t)$$

the solution for the equation of motion will be

$$\frac{d^2x}{dt^2} = F_0 sin(\omega_1 t) - \omega_0^2 x$$

$$x(t) = x_0 sin(\omega_0 t) - \frac{F_0}{\omega_1^2} sin(\omega_1 t)$$
$$= x_0 sin(\omega_0 t) - F_1 sin(\omega_1 t).$$

When $F_1 sin(\omega_1 t) \gg x_0 sin(\omega_0 t)$, the system experiences mode locking meaning that because of a large driving force, the system will first experience close to no movement at small values of t but then at large t, $F_1 sin(\omega_1 t)$ will dominate and the system will oscillate with frequency w_1. At different frequencies when $w_0 \neq w_1$, the system will experience constructive and destructive interference. At the same frequencies, the function will grow constructively until it reaches the same amplitude as the driving force.

1.3 Beat Frequency

Beat frequency occurs when you have two signals of similar amplitudes where $\omega_0 \approx \omega_1$. The sum of the two signals create a singal that has a frequency

$$\frac{\mid \omega_1 - \omega_0 \mid}{2\pi}.$$

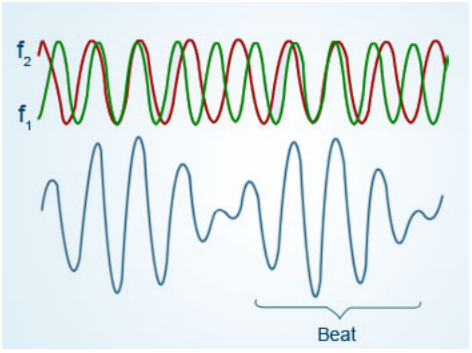

Figure 1: Beat frequency. Credit: http://www.wwk.in/resources

In addition to simulating beat frequency, we examine how the peak amplitude of the beat frequency changes in relation to the driving frequency in the range

$$\frac{\omega_0}{10} \leq \omega_1 \leq 10\omega_0.$$

2 FFT Analysis and Results

Through our FFT program, we found that the frequency of the noisy signal peaked at 440 Hz.

We found that the number of samples at the peak frequency was 1386. To obtain the amplitude of the original signal we use

$$Amplitude = \frac{N_{peak}}{N_{total}/2}$$

We divide by total sample by two since we are looking only at the positive frequencies, or equivalently, we are summing up the total number of peak frequencies in the time domain so we add both the -440Hz and 440Hz magnitudes.

Since the total number of samples is 11025, our amplitude becomes:

$$Amplitude = \frac{1386}{11025/2} = .2514$$

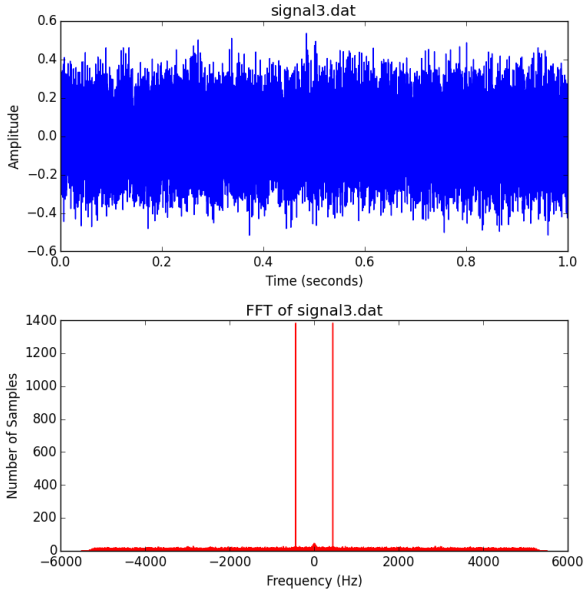

Figure 2: 1) Graph of the noisy signal data. 2) Graph of FFT of the signal.

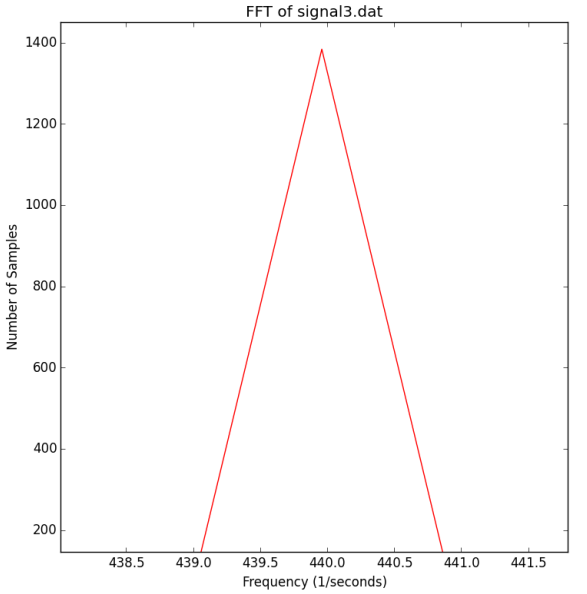

Figure 3: FFT peak at 440Hz.

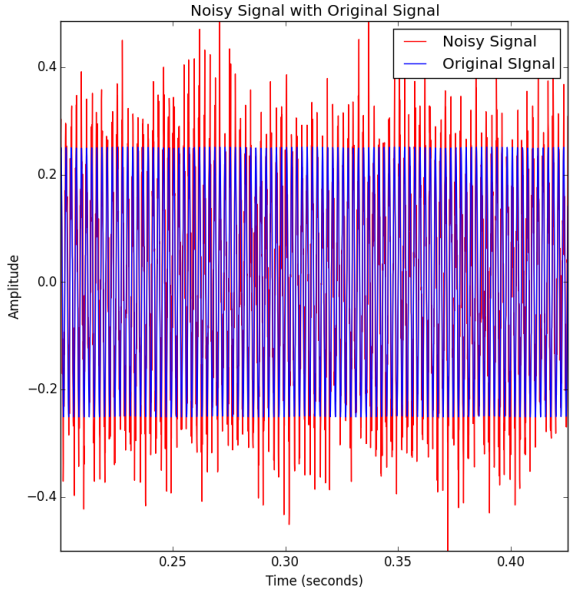

Figure 4: The original signal juxaposed with the noisy data.

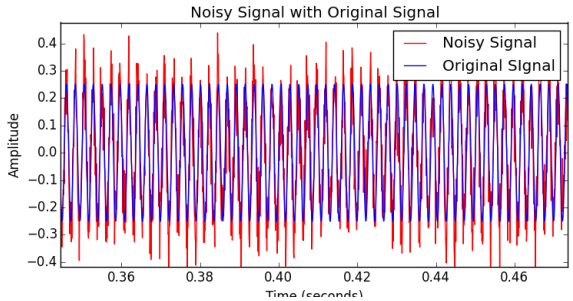

Figure 5: A closer look at figure 3.

3 Curve Fitting Analysis and Results

When we overlay the generated signal over our noisy signal, we find that the two signals coincides. From this, we can say that the phase is zero or $2\pi n$ where $n = 0, 1, 2, ...$

There were no other peak frequencies associated with the signal meaning that the noise was random and that there were no other signals mixing with it. We also see that the amplitude of the noisy signal rarely goes below the original. This is something we should expect since random noise often just add fuzz to a signal and is usually not subtractive.

4 ODE Solution with No External Forces

We solve the system of ordinary differential equations with parameters

$$F_k = -kx, \quad \frac{k}{m} = \omega^2, \quad \omega = 2\pi f.$$

$f = .44$KHz in this simulation. We use KHz because Hz will give us unneccesarily large values. The initial conditions for $x(0)$ is the amplitude of the signal in part one and $v(0)$ is zero so that we have

$$x(0) = .2514, \quad v(0) = 0.$$

Using the ordinary differentiation equation solver given to us, we plot the solution to obtain the graph:

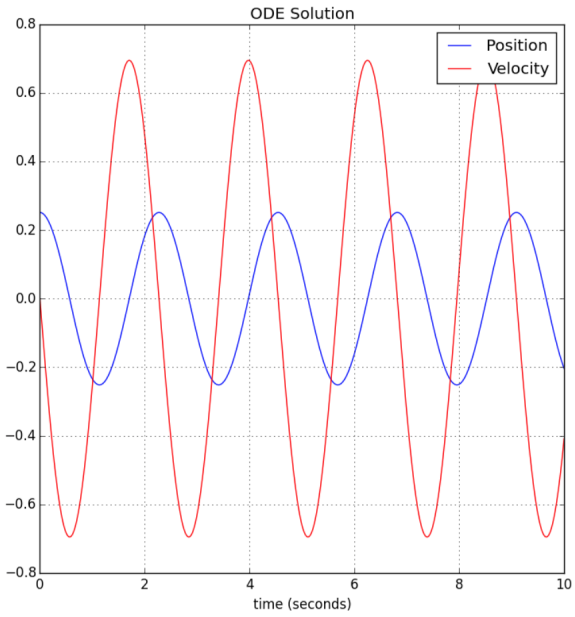

Figure 6: The solution to the mass and spring system with an external driving force.

5 Comparison of ODE Solution with External Forces

If we add an external driving force to the system, our solution looks very different. In the case of a very large driving force with $f_1 = 1$KHz, this leads to the mode locking effect where the system is overwhelmed by the external force and oscillates in phase with it regardless of its frequency.

If the driving force is the same frequency as the restoring spring force, the amplitude of the wave will be initially stuck in place until the system normalizes at some time t. Then, once it starts going, the amplitude grows with time until it reaches the driving force's amplitude.

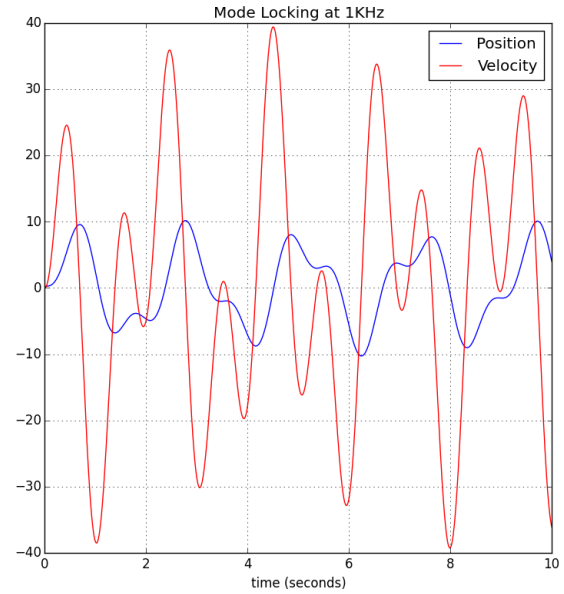

Figure 7: A mass and spring system driven by a disproportionate 1KHz external force.

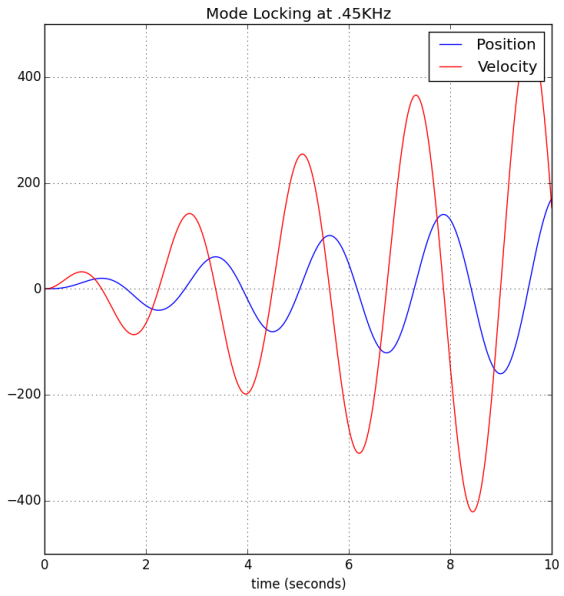

Figure 8: A mass and spring system driven by a disproportionate .44KHz external force.

6 Reflection on Resonances and Beats

We superimpose two test functions and graph their beat frequencies. Graphing a sine wave with f equal .06KHz, we can clearly see that the beat frequency is indeed

$$\frac{\mid \omega_1 - \omega_0 \mid}{2\pi}$$

$$= \mid .5 - .44 \mid = .06KHz$$

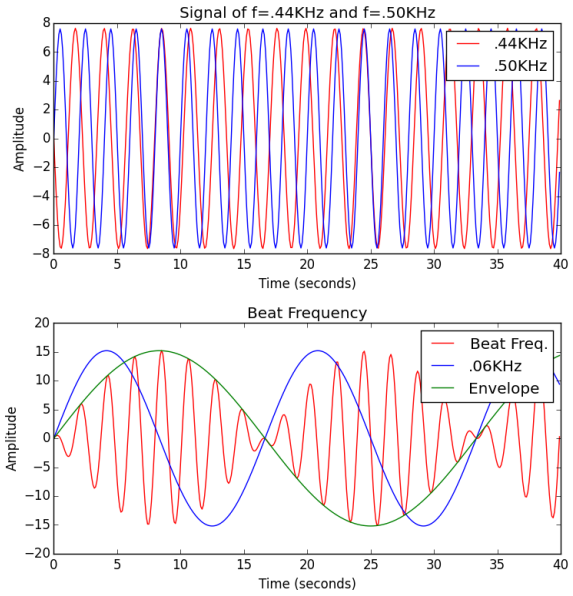

Figure 9: 1) Two signals, .44KHz and .50KHz, superimposed. 2) The sum of the two signals, a .06KHz signal showing that the beat frequency is the difference between the two original frequencies.

We have also that

$$f = 1/T = \frac{1}{16.6677sec} = .06KHz$$

as expected.

In our maximum amplitude vs frequency graph, we see that there is a huge dip when f is .44KHz. This is because the two signals completely destructively interfere and thus generate no signal at all. There are other large dips in the graph and they correspond to (nf) where $n = 2, 3, 4, ...$

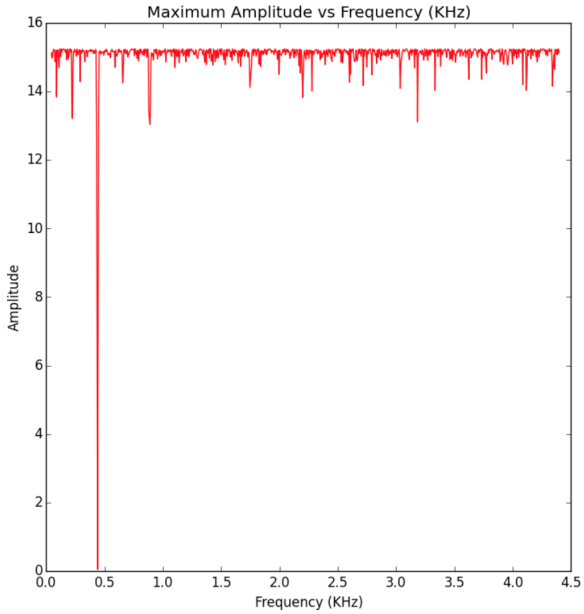

Figure 10: Peak amplitude of beat frequency as a function of frequency.

7 Summary

In conclusion, we explored a few ways of analyzing signals. Although these exercises seem banal, they are important problems in signal processing in that they have many useful applications. We used the Fourier Fast Transform to transform a noisy signal into the frequency domain to make the data more accessible to humans. This allowed us to extract important information such as the frequency, amplitude, and phase of the signal. In the real world, these values could correspond to data in digital bit stream or other kinds of signals used in communications. It is important to understand the power of the theoretical knowledge behind the applications.

Solving ordinary differential equations gives us a feel for using computational science to simulate physical systems. We saw how by just changing parameters in a ODE solver, we received completely different results for the solution to the ordinary differential equation. This is exactly the reason why we use computers for these sorts of problems; our human intuition is fallible when it comes to nonlinear dynamics but once we obtain graphs and figures of a linear nature, it is easy for us to extrapolate information from our data. Discussing beat frequency, we proved that the beat frequency is the difference

between two signal's frequencies. Graphing the maximum amplitude versus the driving force in the range $\frac{\omega_0}{10} \leq \omega_1 \leq 10\omega_0$, we obtain the expected result that when the frequencies of the two signals where equal, the peak amplitude becomes zero. This is because the driving force and the restoring force are 180° out of phase. This makes physical sense for us but (maybe not) surprisingly, the virtual program agrees with the physical system. All is well in the ordered universe.

References

[1] R. Landau, Manual J. Paez, Cristian C. Bordeianu. *A Survey of Computational Physics*, 2010: Princeton University Press.

[2] D. Beazley and Brian K. Jones. *The Python Cookbook*, 2013: O'Reilly.

[3] Newman, M. E. J. *Computational Physics*, 2013: Createspace.

8 Appendix

Noisy Signal and FFT

```
"""
Written by Michael Lee

PHYS440: Computational Physics
Project 3, Fast Fourier Transform, Curve Fitting, and Ordinary Differential Equations

"""

import numpy as np
import scipy
import scipy.fftpack
import matplotlib.pyplot as plt
import matplotlib.figure

time, amplitude = np.loadtxt("signal3.dat", unpack=True)
N = len(time)

normalized_time = np.linspace(0.0,1.0,N)
fast_fourier_transform_of_amplitude = scipy.fft(amplitude)
freq = scipy.fftpack.fftfreq(amplitude.size, normalized_time[1]-normalized_time[0])

plt.subplot(211)
plt.subplots_adjust(hspace=.3)
plt.plot(time2, amplitude, 'b')
plt.title("signal3.dat")
plt.xlabel("Time (seconds) ")
plt.ylabel("Amplitude")

plt.subplot(212)
plt.title("FFT of signal3.dat")
plt.ylabel("Number of Samples")
plt.xlabel("Frequency (Hz)")
plt.plot(freq, abs((fast_fourier_transform_of_amplitude)), 'r')

plt.show()
```

Curve Fitting Noisy Signal with Original Signal

```
"""
Written by Michael Lee.

PHYS440: Computational Physics
Project 3, Fast Fourier Transform, Curve Fitting, and Ordinary Differential Equations

"""

import numpy as np
import math
import matplotlib.pyplot as plt

time, amplitude = np.loadtxt("signal3.dat", unpack=True)
```

```
N = len(time)
signal_fit = np.sin(440*2*np.pi*time)*.25142

plt.plot(time, amplitude, 'r', label="Noisy Signal")
plt.plot(time, signal_fit, 'b', label="Original SIgnal")

plt.title('Noisy Signal with Original Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.legend()

plt.show()
```

ODE Solver by Prof. Snopok

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 04 21:06:33 2013
Modified on Fri Mar 06 2015

@author: snopok
"""

from pylab import *

def f(t,y,omega=0.44*2*pi): # function returns RHS
    fReturn=zeros(2)
    fReturn[0] = y[1] # 1st deriv
    # 2nd derivative: spring, friction, external
    fReturn[1] = -omega**2*y[0]+100*sin(.45*2*pi*t)
    return fReturn

def rk4(t,y,h,f,omega): # rk4
    n_dim = y.shape[0]
    k1 = zeros(n_dim)
    k2 = zeros(n_dim)
    k3 = zeros(n_dim)
    k4 = zeros(n_dim)
    # calculate k1 = h*f(t_n,y_n)
    k1 = h*f(t,y,omega)
    # calculate k2 = h*f(t_n+h/2,y_n+k_1/2)
    k2 = h*f(t+h/2.0,y+k1/2.0,omega)
    # calculate k3 = h*f(t_n+h/2,y_n+k_2/2)
    k3 = h*f(t+h/2.0,y+k2/2)
    # calculate k4 = h*f(t_n+h,y_n+k_3)
    k4 = h*f(t+h,y+k3)
    y = y+(k1+2.0*(k2+k3)+k4)/6.0
    return y

if __name__ == "__main__":

  #   Initialization
  a = 0.0 # start time
  b = 10.0 # end time
  t = a # initialize time
  n = 10000 # number of steps
  h = (b-a)/n # step size for rk4
#  ydumb = zeros((2),float)
```

```
  y = zeros(2)
  y[0] = 0.2514 # initial position
  y[1] = 0.0 # initial velocity
  omega = 0.44*2*pi # frequency

#  tarray=zeros(n+1)
  tarray=[]
  xarray=[]
  varray=[]
#  i=0
  while (t<b): # Time loop
    if ((t+h)>b): h=b-t # Last step
    y=rk4(t,y,h,f,omega) # rk4 iteration
    t=t+h # increase time
    # save data for plotting:
#    tarray[i]=t
    tarray.append(t)
    xarray.append(y[0])
    varray.append(y[1])

  figure(1)
  plot(tarray, xarray, 'b', label="Position")
  xlabel('time (seconds)')
  plot(tarray, varray, 'r', label="Velocity")
  title('Mode Locking at .45KHz')
  legend()
  grid('on')

show()
```

Beat Frequency

```
"""
Written by Michael Lee.

PHYS440: Computational Physics
Project 3, Fast Fourier Transform, Curve Fitting, and Ordinary Differential Equations

"""

import numpy as np
import matplotlib.pyplot as plt

time = np.arange(0,40,.1)
omega = .44*2*np.pi
signal1 = -omega**2*np.sin(omega*time)
signal2 = 7.6*np.sin(.5*2*np.pi*time)
beat_frequency = signal1 + signal2
sixty_hertz_signal = 15.2*np.sin(.06*2*np.pi*time)
envelope = 15.2*np.sin(.06*np.pi*time)

plt.subplot(211)
plt.subplots_adjust(hspace=.3)
plt.plot(time, signal1, 'r', label=".44KHz")
plt.plot(time, signal2, 'b', label=".50KHz")
plt.legend()
plt.title("Signal of f=.44KHz and f=.50KHz")
plt.xlabel("Time (seconds)")
```

```
plt.ylabel("Amplitude")

plt.subplot(212)
plt.plot(time, beat_frequency, 'r', label="Beat Freq.")
plt.plot(time, sixty_hertz_signal, 'b', label=".06KHz")
plt.plot(time, envelope, 'g', label="Envelope")
plt.legend()
plt.title("Beat Frequency")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")

plt.show()
```

Peak Amplitude vs Driving Frequency

```
"""
Written by Michael Lee

PHYS440: Computational Physics
Project 3, Fast Fourier Transform, Curve Fitting, and Ordinary Differential Equations
"""

from pylab import *
import numpy as np
import matplotlib.pyplot as plt

omega = .44*2*np.pi
time = np.arange(0, 40, .1)

def max_amplitude_of_beat_frequency(signal1, signal2):
    beat_function = signal1 + signal2
    max_amp = max(beat_function)
    return max_amp

if __name__ == '__main__':

    list_of_max_amps = []
    frequency_range = np.arange(.044, 4.4, .0044)

    function_1 = -omega**2*sin(omega*time)

    for i in frequency_range:
        function_2 = 7.6*sin(i*2*np.pi*time)
        list_of_max_amps.append(max_amplitude_of_beat_frequency(function_1, function_2))

    plt.plot(frequency_range, list_of_max_amps, 'r')
    plt.title("Maximum Amplitude vs Frequency (KHz)")
    plt.xlabel("Frequency (KHz)")
    plt.ylabel("Amplitude")
    plt.show()
```