Nathan Tung
004-059-195
Discussion 1A

CS170A – Homework 3

1. Fourier Music

```
[y,MysteryFs] = wavread('mystery.wav');
n=length(y);
```

a. We are told that `MysteryFs` is an incorrect samping frequency. In order to determine the correct sampling frequency `Fs`, we experiment by plugging in different values into `sound(y,Fs)`.

   By comparing our sound file with "Beethoven's Fifth Symphony," we see that the frequency is much larger than `MysteryFs=11025`. The real sampling frequency is approximately `Fs=47000`.

b. The factors of length `n` as well as `(n-27)` are:

```
factor(n) = 2 2 2 2 2 2 2 2 2 7 223

factor(n-27) = 799259
```

c. To calculate the average time required for `fft(y(1:n));` and `fft(y(1:(n-27)));` we record the cpu time, run a for-loop of the specified operation 100 times, then subtract current cpu time from the previous one.

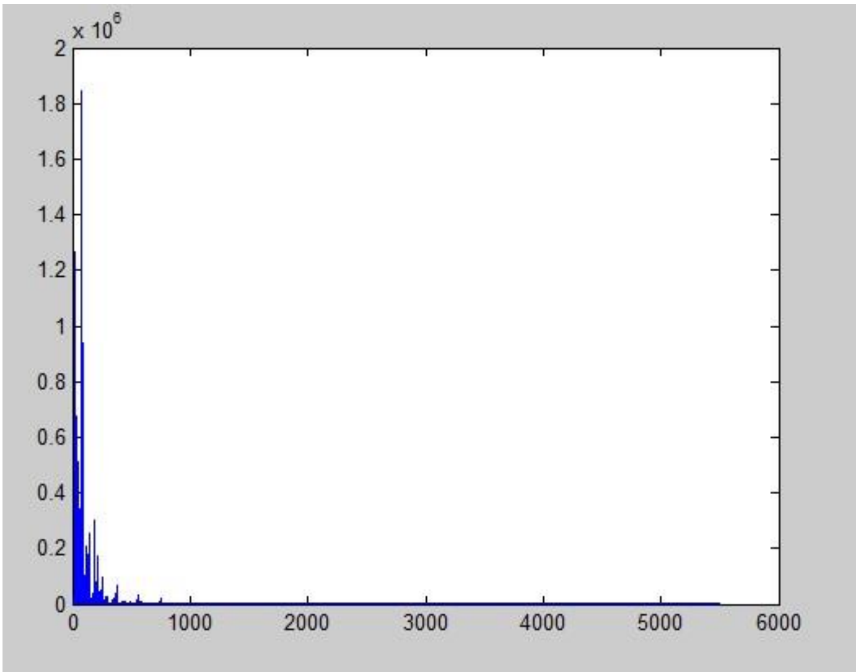
```
t = cputime;
for i=1:100
   fft(y(1:n));
end
t1 = (cputime - t)/100;

t = cputime;
for i=1:100
   fft(y(1:(n-27)));
end
t2 = (cputime - t)/100;
```

   Here we find that `t1=0.2897 s` and `t2=0.0679 s`. In other words, Fast Fourier Transform works faster on the second operation since `n-27` has more factors into which it can be decomposed.

d. In order to plot the spectrum of frequency against the Fourier transform power, we follow a series of steps.

   First, we let `Y=fft(y)`, or the Fourier transform of little y. Next, we need to define the power of Y to be `power = abs(Y(2:(n/2+1))).^2`, such that we only look at the first half of the vector. Note that Nyquist frequency is `NyquistFrequency = Fs/2`. Lastly, we determine `frequencies = linspace(1, NyquistFrequency, n/2)`. Finally, we can plot the frequencies against the power.

```
plot(frequencies, power);
```



Power vs. Frequency (Hz)

e. Our function can be written as follows:

```
function [spike_freq,spike_power] = largest_spike(power,freq_range)

position = find(power==max(power));
spike_freq = freq_range(position);
spike_power = power(position); %same as max(power)
```

We find the max power then determine the index of that largest value. Lastly, we set our output to the frequency and power at that index.

In order to find the four frequencies of the four largest spikes, we modify this function a bit such that the largest value in power is set to 0 each time we find the max. Then the next search for the max value will find the second-largest value and so on.

```
function four_spikes(power,freq_range)

pos1 = find(power==max(power));
spike_freq1 = freq_range(pos1)
spike_power1 = power(pos1);
power(pos1)=0;

pos2 = find(power==max(power));
spike_freq2 = freq_range(pos2)
spike_power2 = power(pos2);
```

Nathan Tung
004-059-195
Discussion 1A

```
power(pos2)=0;

pos3 = find(power==max(power));
spike_freq3 = freq_range(pos3)
spike_power3 = power(pos3);
power(pos3)=0;

pos4 = find(power==max(power));
spike_freq4 = freq_range(pos4)
spike_power4 = power(pos4);
```

The output for the four frequencies looks like this:

```
spike_freq1 = 70.4958
spike_freq2 = 74.1367
spike_freq3 = 70.5371
spike_freq4 = 24.4457
```

f. The function I wrote below takes in a frequency list. For each frequency in the list, we determine which note it is closest to, as well as how many octaves apart it is from the "original" set of notes. We output an array of notes, which are actually the k-values of the notes. Notice that k=0 is a C, k=11 is a B, and so on. An octave array is also produced, although not stored as an output.

```
function notes = musical_notes(frequencies)

n=length(frequencies);
notes = [n,1];
octaves = [n,1];

C = 261.63; %frequency of C

    function c0 = c(z)
        c0 = 2^(z/12);
    end

for i=1:n
    freq=frequencies(i);
    octave_diff=0;

    while(freq>508.56) %this value is between B and a high C
        freq=freq/2; %check for lower octave
        octave_diff=octave_diff+1; %increment octave difference
    end

    while(freq<254.29) %this value is between C and a low B
        freq=freq*2; %check for higher octave
        octave_diff=octave_diff+1; %increment octave difference
    end

    %By now, freq is somewhere in range of first octave
        k=0;
```

Nathan Tung
004-059-195
Discussion 1A

```
    while(freq>c(k)*C)
        k=k+1;
    end

    %freq is between the note represented by k and (k-1)
    if(k==0 || k==11)
        notes(i)=k;
    else
        small=freq-C*c(k-1);
        large=C*c(k)-freq;

        if(small<large)
            notes(i)=k-1;
        else
            notes(i)=k;
        end
    end

    octaves(i)=octave_diff;
end
octaves %print out the octave difference for each note
end
```

Plugging in our four spike frequencies from before, we get this output.

```
musical_notes([spike_freq1;spike_freq2;spike_freq3;spike_freq4])

octaves =      2      2      2      4
ans =      1      2      1      7
```

notes = C#, D, C#, G (octave difference of 2 away from the other three notes)

In other words, our first and third largest spike frequencies were C# notes, while the second spike frequency was a note of D. The last spike frequency was a note of G and two octaves different from the other notes. None of our frequency pairs seem to be in the 5/4 harmony ratio. We have been using `Fs=47000` to answer the previous questions until now.

g. Here, we use similar code from before in order to find the key of an input song. I ran the function with `'mystery.wav'` as an input.

```
function note = key(wavfile)

[y,Fs]=wavread(wavfile);
n=length(y);

Y=fft(y);
power=abs(Y(2:(n/2+1))).^2;
NyquistFrequency = Fs/2;
frequencies=linspace(1, NyquistFrequency, n/2);

freq = frequencies(find(power==max(power)));
```

Nathan Tung
004-059-195
Discussion 1A

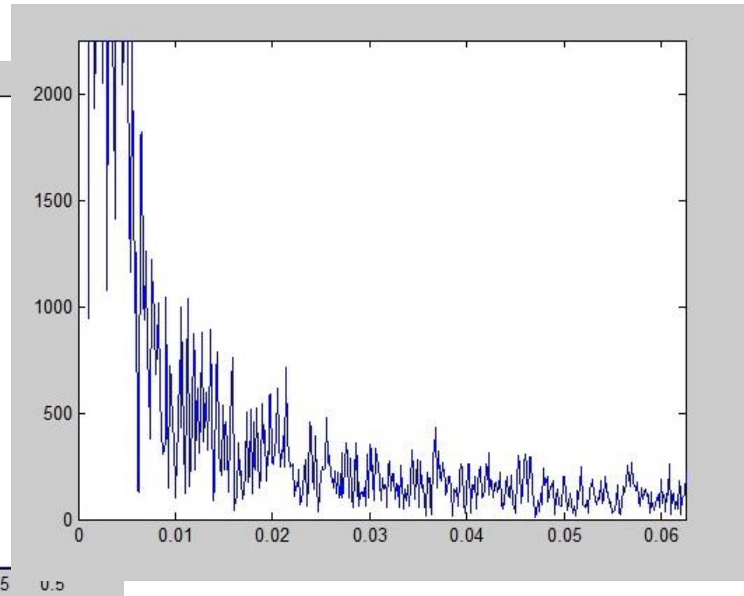
```
C = 261.63; %frequency of C

    function c0 = c(z)
        c0 = 2^(z/12);
    end

while(freq>508.56) %this value is between B and a high C
    freq=freq/2; %check for lower octave
end

while(freq<254.29) %this value is between C and a low B
    freq=freq*2; %check for higher octave
end

%By now, freq is somewhere in range of first octave

k=0;
while(freq>c(k)*C)
    k=k+1;
end

%freq is between the note represented by k and (k-1)

if(k==0 || k==11)
    note=k;
else
    small=freq-C*c(k-1);
    large=C*c(k)-freq;

    if(small<large)
        note=k-1;
    else
        note=k;
    end
end

end
```

Running this piece of code shows us that `mystery.wav` is in a key of k=1, which corresponds to C#. That is, the note played at highest power is a C#.

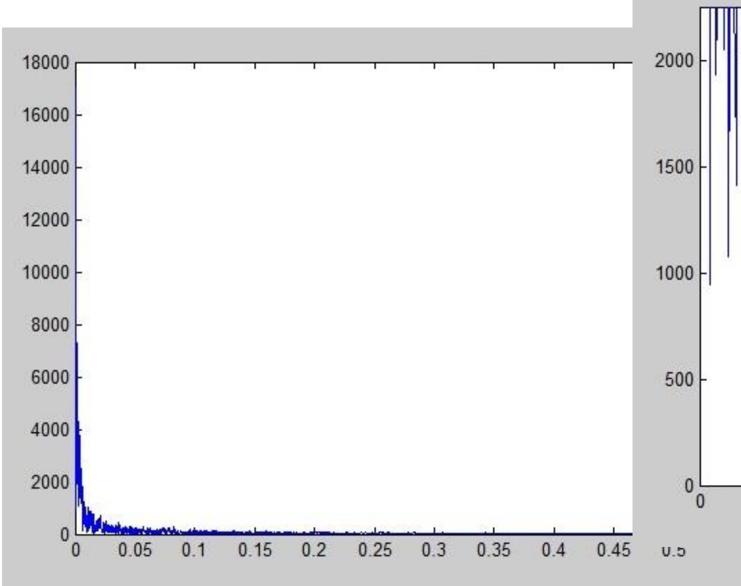
Nathan Tung
004-059-195
Discussion 1A

2. **Stock Analysis**

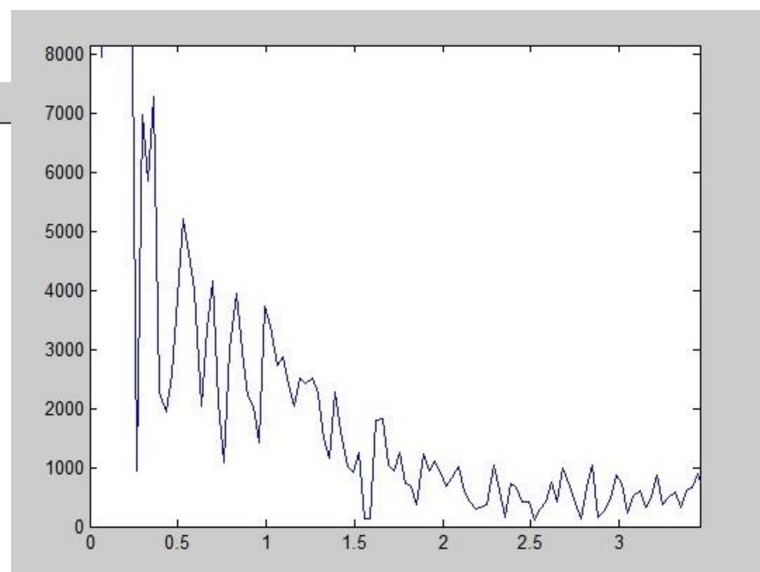
We will be analyzing stock from Advanced Micro Devices, Inc. (AMD), which has its origins in 1983. Therefore, we have decades of stock data. Running the given code on AMD.csv and fixing the `power` to read `power_spectrum`, we obtain the following plots.



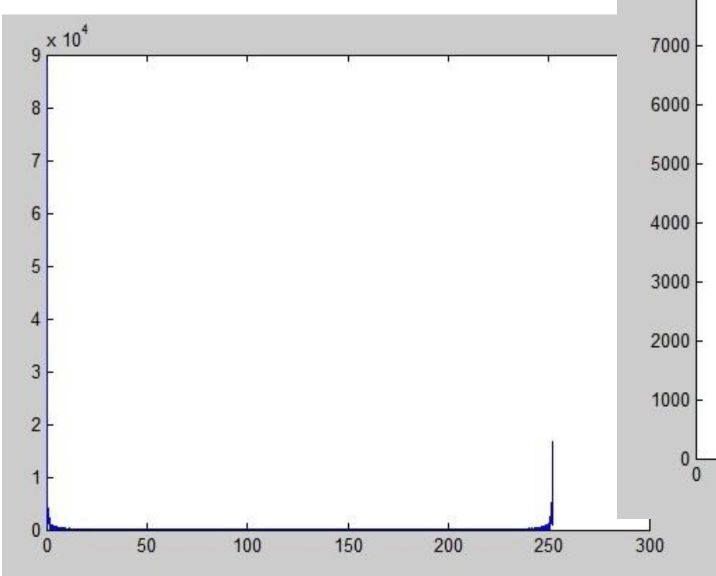
Stock Data (1/day) enlarged



Stock Data (1/day)



Stock Data (1/year) enlarged



Stock Data (1/year)

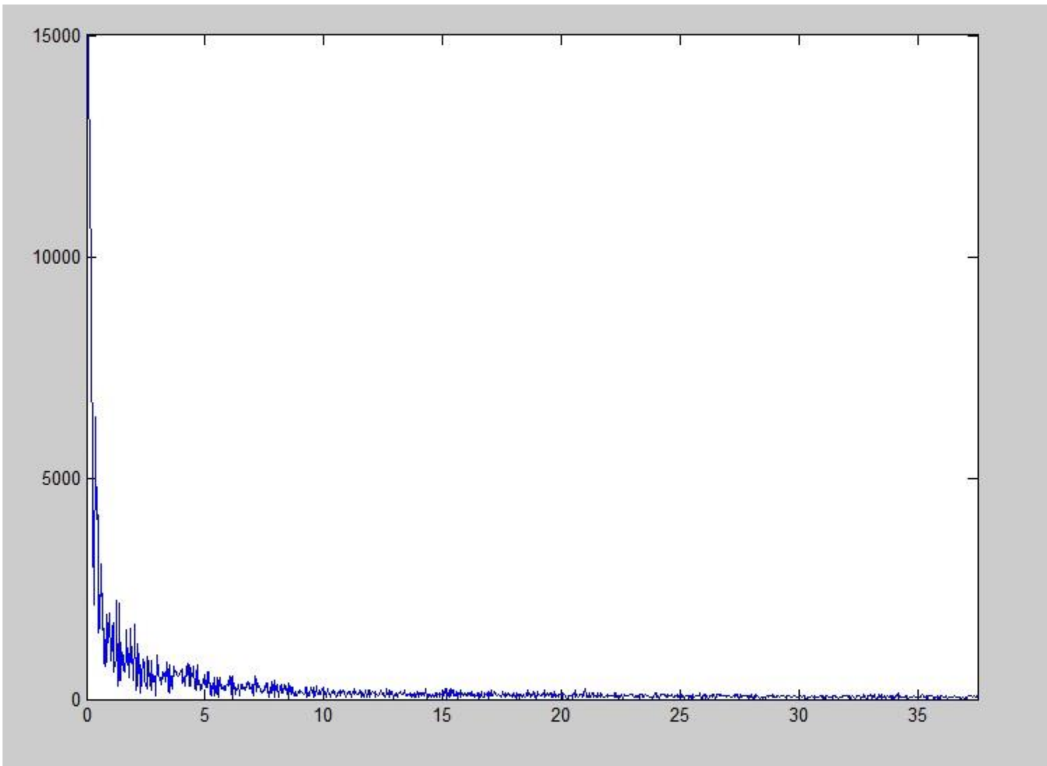
Nathan Tung
004-059-195
Discussion 1A

a. As we can see, the power spectrum indeed spikes at frequencies around 0.002/day and 0.004/day. When multiplied by 252 days in 1 year, we see that these frequencies become 0.504/year and 1.008/year. Likewise, the yearly stock data has peaks corresponding to these frequencies as well.

b. This time, we look at Microsoft (MSFT) stock and look at the power spectrum produced by `read_stock()`.

   We can run our code from before – for example, `four_spikes` – to see at which frequencies the power peaks. The largest spikes are found at these annual frequencies:

```
spike_freq1 = 0
spike_freq2 = 0.0368
spike_freq3 = 0.0735
spike_freq4 = 0.110
```



MSFT Power Spectrum (1/year)

c. In order to write our function, we use basic Matlab functions like `fopen()` and `fprintf()` to save our .csv file. We also use `strcat()` to combine strings, particularly for forming the filename. Lastly, I use my `four_spikes` function from before to give the largest peaking frequencies.

Nathan Tung
004-059-195
Discussion 1A

```
function find_freq(fileName)

%copy down the information into 'fileName.csv'
s=urlread(strcat('http://ichart.finance.yahoo.com/table.csv?s=',fileName));
fid=fopen(strcat(fileName,'.csv'),'wt');
fprintf(fid,s);

[time, quotes] = read_stock(strcat(fileName,'.csv'));
n = length(quotes);
power_spectrum = abs(fft(quotes)).^2;
frequencies = linspace(0, 1.0, n);
plot(frequencies(2:floor(n/2)), power_spectrum(2:floor(n/2)));
freqs = linspace(0, 252, length(power_spectrum));
figure; plot( freqs, power_spectrum );

%return frequencies of highest peaks
four_spikes(power_spectrum,freqs);
```

Using `fileName='AAPL'` we get the normal plots and these peak frequencies:
`spike_freq1 = 0; spike_freq2 = 0.0348; spike_freq3 = 0.0696; spike_freq4 = 0.1044`

3. A Simple 'Fake Photo' Detector

a. We write a function called `fakePhoto` that will produce some grayscale image. This image's brightness can tell us whether parts of the original input image were altered. Our code is as follows.

```
function reshaped = fakePhoto(photoName)

%save photo as m x n RGB image and obtain dimensions
A=imread(photoName);
sizeOfA=size(A);
m=sizeOfA(1);
n=sizeOfA(2);

%reshape the image into (m*n) x 3
R=reshape(A,m*n,3);

%find covariance of R along with its PCs
CovR=cov(double(R));
[U,S,V]=svd(CovR);
PrincipalComponents=U(:,1:3);
SecondPC=PrincipalComponents(:,2);

%project image into its second PC; reshape to m x n
reshaped=double(R)*double(SecondPC);
reshaped=reshape(reshaped,m,n);

%display grayscaled image
imshow(reshaped);
```

Nathan Tung
004-059-195
Discussion 1A

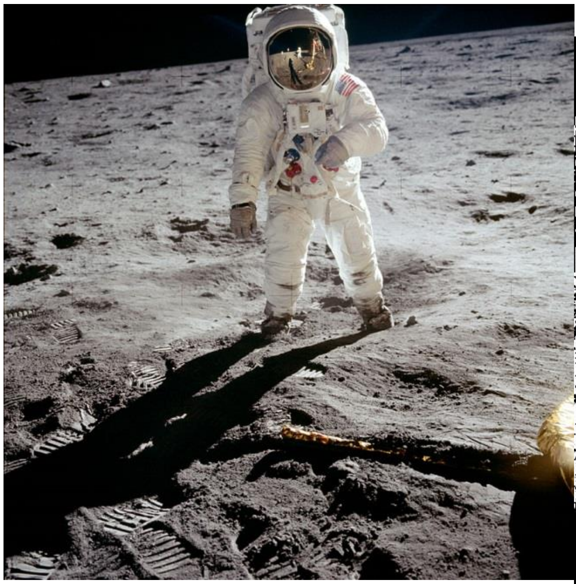
Using our code, we can compare images with their grayscaled, color-distributed alternatives. For example, we use a colored image of the fake iPhone 6 (rather than the black and white image suggested). The results are quite clear.

Original iPhone 6 Image

Testing iPhone 6 Image

We can tell that some of the app icons were obviously retouched, along with the bottom curve of the phone. The part of the fake image that was most difficult to produce – for example, the curvy surface up top – shows many white spots that look out of place. On the other hand, every other part is pitch black. We can tell that the image is just a rumor.

b. Now we can try it on the Apollo 11 landing picture.



Original Apollo 11 Image

Testing Apollo 11 Image

Here, the black and white color distribution is quite evenly distributed. There does not seem to be any obscure spots or very dark areas. I would consider this image to be unaltered and not so much the controversy that many consider it to be.

c. Finally, we look at a random image that is without a doubt altered drastically.

Original Apple/Mouth Image

Testing Apple/Mouth Image

The hand grasping the apple is the most realistic part of the image. We know that the teeth, mouth, and tongue on the apple are undoubtedly photoshopped. Our grayscale image agrees – we have many out-of-place dark and white spots where the image was edited.

Overall, our fake photo detector appears to be a useful tool in verifying the authenticity of questionable images.