HW3: Tone Encoding and Decoding
CSE1010 Fall 2012
Jeffrey A. Meunier
University of Connecticut

1. Introduction
In this assignment you will write a Matlab program to generate (or *synthesize*) and analyze DTMF sounds, otherwise known as Touch Tone® sounds. You will use Matlab vectors to hold the sound data, plot some of it, and then use the Fourier Transform to take a sound signal and extract its frequency components.

There is a lot of information in this project document. I have written a lot of explanation, and I also give you quite a few example graph images which take up extra space. The project does not require a lot of programming, but it does require a lot of understanding. You will undoubtedly see some of this information again on the midterm exam. I give you a lot of examples and ways to test your program as you develop it. Please do not skip these parts of the assignment.

2. Value
This program is worth a maximum of 20 points. See the grading rubric at the end for a breakdown of the values of different parts of this project.

3. Due Date
This project is due by 11:59PM on Sunday, September 30, 2012.

4. Objectives
The purpose of this assignment is to give you familiarity with:

- defining functions
- calling functions
- using vectors to store data
- processing sound
- finding peaks in a graph
- plotting / graphing

Keywords: function, vector, peak, plot, graph, DTMF, Touch Tone, sound, encode, decode,

5. Background

In this assignment you will be generating and analyzing DTMF sound signals, otherwise known as Touch Tone® sounds. Have a look at this web link to be sure you understand the background:

http://en.wikipedia.org/wiki/Dual-tone_multi-frequency_signaling

You don't need to memorize anything or use any of that information in your program, but it's an interesting read nevertheless.

Interestingly, in that article under the **Keypad** heading there is a table of the keys, where each key is a link to a sound file. Clicking the link will play the sound. You may find this useful later.

5.1. if statements

You will need to use a few if statements in this project, but I haven't explained them to you yet. Thankfully they're not that difficult to learn. Have a look at section 5.2.2 in the book on pages 139 – 141.

5.2. Sound Waves

In class you have seen me use the **sin** function to generate a sine wave. A sine wave is not just a mathematical abstraction, it is the wave that represents any pure tone. Matlab is quite good at generating and playing tones.

The tone of middle-C is 261.626 Hz (Hz = cycles per second), so if we were able to generate a sine wave that oscillates at that rate and have Matlab play it, you would hear middle-C. Let's do that.

First you would need to tell Matlab how many samples per second you desire (this is how tones are generated by digital computers). Let's choose 8192 samples per second, because it's both a power of 2 (that will be important later) and it's a reasonable number for what we need. We also need to determine how long we want the tone to play, so let's say 2 seconds. So the time domain is the range 0 to 2 seconds chopped into 8192 segments per second. This is the expression in Matlab for the time domain:

```
seconds = 2;
samplesPerSecond = 8192;
timeDomain = linspace(0, seconds, seconds * samplesPerSecond);
```

Let's have a look at the first 10 elements of the **timeDomain** vector:

```
>> timeDomain(1:10)
ans =
  Columns 1 through 6
    0    0.0001    0.0002    0.0004    0.0005    0.0006
  Columns 7 through 10
    0.0007    0.0009    0.0010    0.0011
```

This means that the sound we generate will be sampled at 0 seconds, then again at 0.0001 seconds, then at 0.0002 seconds, and so on.

We won't just take the **sin** of the **timeDomain** vector to generate a wave. Its units are seconds, but the sin function expects numbers in an angle as some (possibly fractional) multiple of 2π. We already know that middle-C oscillates 261.626 times per second, meaning it goes through 261.626 x 2π radians every second. Thus, if we multiply the **timeDomain** vector by this amount, we will get a frequency vector to which we can apply the **sin** function.

```
freq = 261.626;
tone = sin(2 * pi * freq * timeDomain);
```

This signal is much too dense to plot, but it is perfect for playing. The **sound** function can do this if you give it the tone vector and the number of samples per second of the tone vector. However, the amplitude of the sound is quite large. Let's turn the volume down by half first:

```
tone = tone * 0.5;
```

Now play the tone:

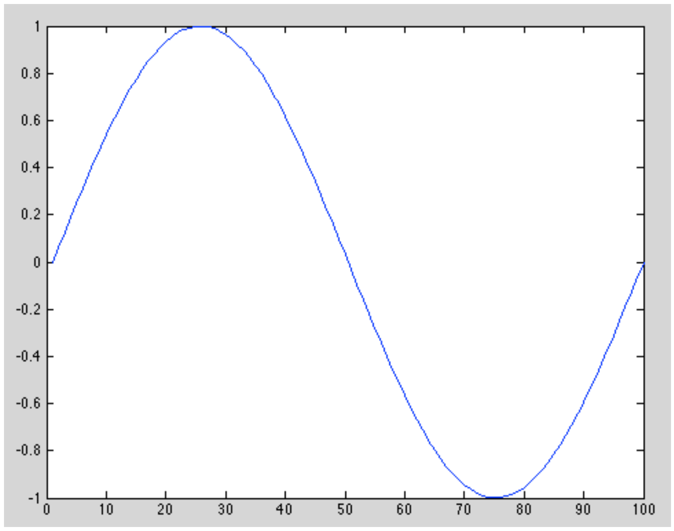
```
sound(tone, 8192)
```

If you're using a computer in the lab to do this, you will need to plug in earphones, otherwise Matlab complains that there is no sound hardware installed.

5.3. Combining Sound Waves

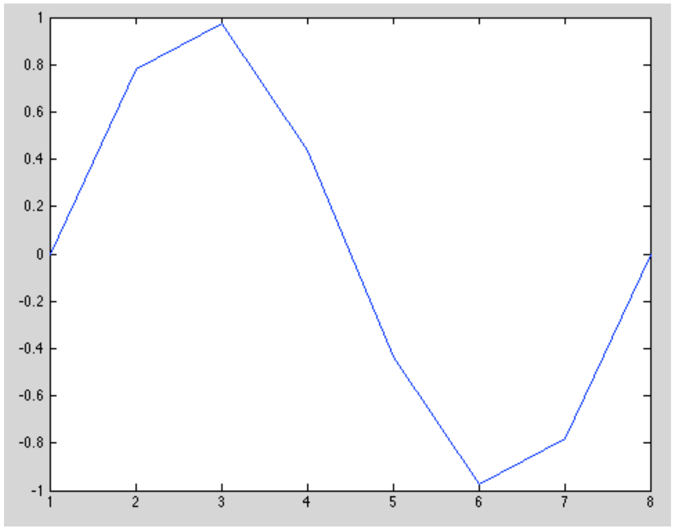
Let's look at a simple sound wave of 1 Hz (or 1 cycle per second) over 1 second, with 100 samples per second.

```
timeDomain = linspace(0, 1, 100);
wave1 = sin(2 * pi * 1 * timeDomain);
plot(wave1)
```



The plot shows one complete cycle of the wave. The wave is quite smooth at 100 samples per second (or simply *100 samples*, since it's exactly 1 second). Just for fun, let's try it at 8 samples instead.

```
plot(sin(2*pi*linspace(0,1,8)))
```

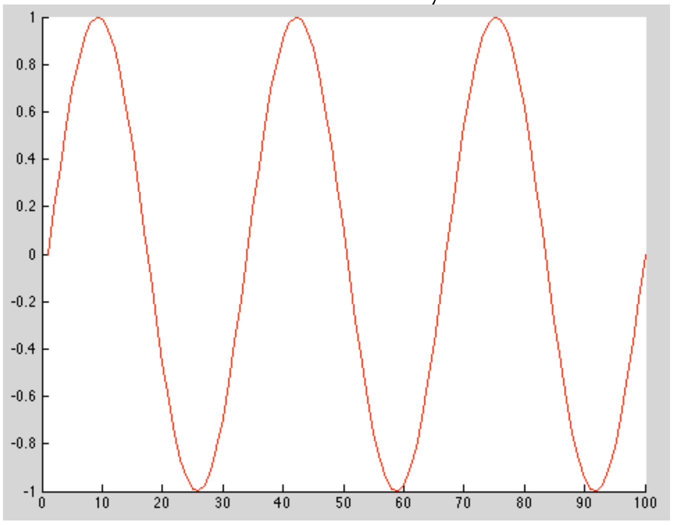


You can see it's much more jagged.

Say we wanted to combine two tones, a 1Hz wave and a 3Hz wave. The way to generate the vector of these two waves would be to add the two separate wave vectors together. Let's have a look at the 3Hz wave.

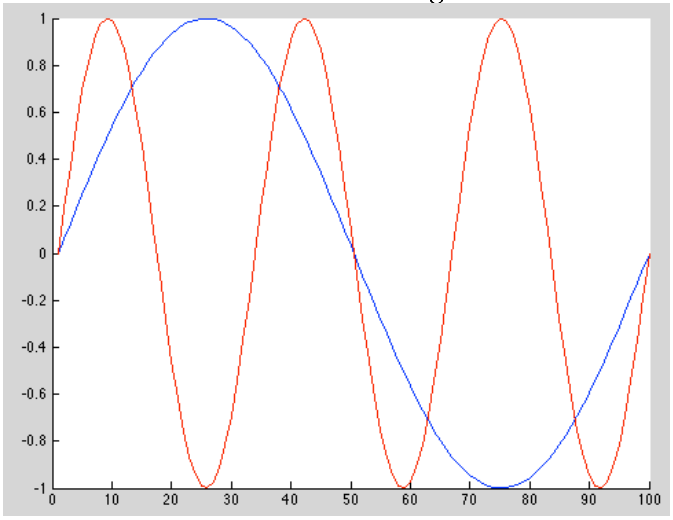
```
  wave2 = sin(2 * pi * 3 * timeDomain);
```

This is what wave2 looks like by itself:

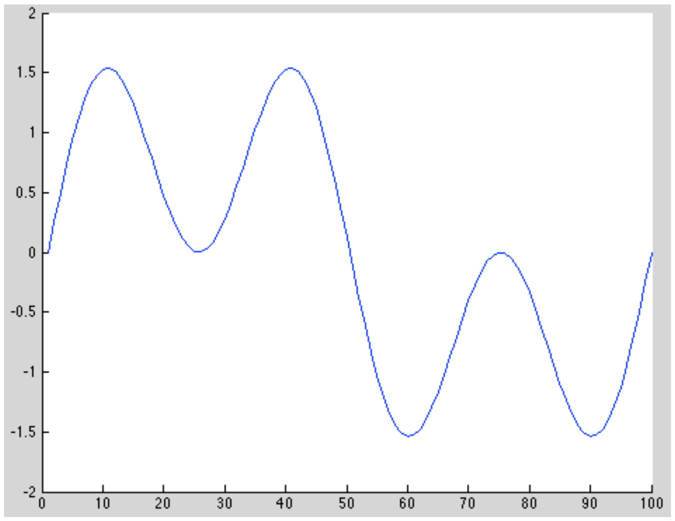


You can see that this plot contains three complete cycles of the wave.

You would think that combining two waves causes them to somehow look like this:



But in fact this is not how waves combine. The amplitudes (heights) simply add together, yielding a single wave that is the sum of the separate waves. This is what 1Hz + 3Hz looks like:



Notice how the amplitude now exceeds 1: it goes as high as 1.5 and as low as –1.5, so this wave is louder than either of the two separate waves. Also observe in the previous plot at the 25th sample the blue 1Hz wave is at its maximum of 1 and the red 3Hz wave is at its minimum of –1. At that point the sum of the red and blue waves is 0. That's what the value of this combined wave is at the 25th sample.

Consider going the opposite direction: Given a complex wave like this one, is it possible to determine what fundamental waves make up this wave? In other words, can we perform some analysis that tells us that in fact this wave is made of a 1Hz wave and a 3Hz wave? It turns out that a mathematician named Joseph Fourier discovered that it is possible, and developed an algorithm to do it. This is known as the Fourier Transform, and you'll use it in this project.

6. Assignment

Read through all of the subsections in this section before you start, in order to understand how you must proceed with the assignment. This assignment has two parts: in the first part, you will write functions that synthesize DTMF sounds. In the second part, you will write functions that analyze DTMF sounds to determine what their frequency components are.

6.1. Sound Synthesis

In this first part you will write functions to synthesize DTMF sounds, first by writing a function that creates single pure tones, then combining the tones into a dual-tone sound.

6.1.1. Function generateTone

This function will generate a single sinusoidal pure tone of a specified frequency.

Write a function called **generateTone** that has three parameters:

1. a frequency
2. the number of samples per second
3. the number of seconds of the tone duration

Name the parameters well. I would suggest **freq** or **frequency**, **sampsPerSec** or **samplesPerSec** or **samplesPerSecond**, and **seconds** or **duration** or even **toneDurationSeconds**.

This function must return one value: it will be the generated tone. Name this return variable well also.

This is the outline of the function:

```
function ____ = generateTone(___ , ___ , ___)
   ___
end
```

You have to fill in the blanks.

In this function use the **linspace** function to create a time domain that goes from 0 to the duration, having the required number of samples per second. I showed you how to do that in the **Background** section.

Then apply the **sin** function to the time samples. Be sure to multiply the time domain by the frequency and 2π.

Also multiply the resulting tone vector by 0.5 so that the amplitude (volume level) is reduced a bit. Otherwise the sound will be quite loud.

After you have written the function, test it. In the background section I showed you how to generate the tone for middle-C. Make sure this function generates the same tone:

```
sound(generateTone(261.626, 8192, 1))
```

6.1.2. Function synthDtmf

This function will generate a dual tone by using the **generateTone** function you just wrote.

Write a function called **synthDtmf** that has two parameters that are two frequency numbers. This function generates two tones by calling the **generateTone** function twice, once for each frequency number. Finally, the two tone vectors are added together and returned from this function. Note that as long as the vectors have the same size (and they will) they can be added together directly, something like this:

```
sumVector = vector1 + vector2;
```

When you call the **generateTone** function you must specify the frequency (you know that: it's in the first parameter variable for the first tone, the second parameter variable for the second tone), and use 8192 as the number of samples per second, and a duration of 0.3 seconds.

The function will look like this:

```
function ___ = synthDtmf(___ , ___)
  tone1 = ___;
  tone2 = ___;
  ___ = ___ + ___;
end
```

Here's an example. The number 0 on the telephone keypad generates a sound that's the sum of two frequencies: 941Hz and 1336Hz. This function call generates and plays a DTMF sound for the number 0 on the telephone keypad:

```
sound(synthDtmf(941, 1336))
```

Compare it to the reference sound found here:
http://upload.wikimedia.org/wikipedia/commons/2/2d/Dtmf0.ogg

6.1.3. Function nameToFreq

This function will take the name of any one of the 16 keys on the extended telephone keypad (0-9, A-D, *, #) and return the two frequencies that make up the DTMF sound for that key.

Create a function called **nameToFreq.** This function has one parameter: the name of the key on the keypad.

This function returns two values. To do this, specify a vector of two variables as the function's return variables.

```
function [___ , ___] = nameToFreq(___)
```

You must be sure to assign values to both return variables before the function ends.

In this function check if the name is equal to the character string '0'. If it is, assign the two frequencies 941 and 1336 to the two return variables in any order. It starts like this:

```
if ___ == '0'
  ___ = 941;
  ___ = 1336;
```

Otherwise, if the name is equal to '1', the function will return 697 and 1209. It continues like this:

```
elseif ___ == '1'
  ___ = 697;
  ___ = 1209;
elseif . . .
  . . .
end
```

Here is the complete table of frequencies for the keys:

Key name	**Frequency 1**	**Frequency 2**
'0'	941	1336
'1'	697	1209
'2'	697	1336
'3'	697	1477
'4'	770	1209
'5'	770	1336
'6'	770	1477
'7'	852	1209
'8'	852	1336
'9'	852	1477
'A'	697	1633
'B'	770	1633
'C'	852	1633

'D'	941	1633
'*'	941	1477
'#'	941	1477

Add the remaining key names to your function so that the function handles all 16 keys.

Test the function.

```
>> [f1 f2] = nameToFreq('0');
f1 =
    941
f2 =
    1336
>> sound(synthDtmf(f1, f2))
```

The sound should be the same as before. Test some of the other sounds to be sure they work.

6.1.4. Function touchTone

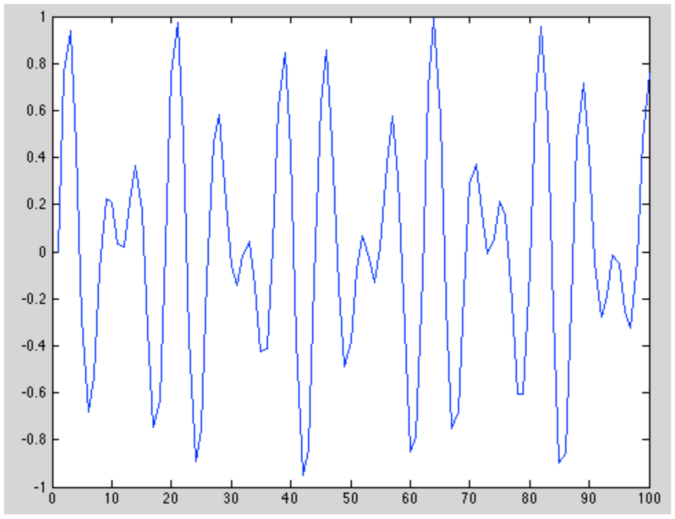
This function will take the name of any one of the 16 keys on the extended telephone keypad (0-9, A-D, *, #) and return the sound vector of the two tones that make up the DTMF sound.

Create a new function called **touchTone**. Inside this function you will need to call the **nameToFreq** function (remember there's a difference between defining a function and calling a function, and inside a function that you're writing you can write a function call to any other function), save the return values in two variables, and then call the **synthDtmf** function using those two variables as arguments. The return value of the **synthDtmf** function must be returned from this function.

```
inside touchTone:
    call nameToFreq, save in two variables
    call synthDtmf with those two variables
    make sure the return value from synthDtmf is returned from touchTone
```

This is a plot of the first 100 samples of the sound generated for '0':

```
t0 = touchTone('0');
plot(t0(1:100))
```



You are now finished with the sound synthesis portion of this assignment.

6.2. Sound analysis

Given a vector containing sound data, we can analyze it to determine what its frequency components are by using the Fourier Transform. In general, the Fourier Transform takes a signal in the time domain and converts it to a signal in the frequency domain. Matlab has a *Fast Fourier Transform*, or FFT, function built-in already, so all we need to do is use it. I will *not* discuss it here. You'll learn about it when you take an electrical engineering course.

The hard part is that the vector returned from Matlab's FFT function requires some manipulation in order to be made useful to us. I'll give you a lot of the Matlab code needed to do that.

Although the sampling frequency we are using is 8192 samples per second, which in theory lets us detect tones that go up to 8192 Hz, there are some issues with how sampling works and how the Fourier Transform treats the data, making frequencies above 4096 (half of 8192) unimportant to us. Thus, we shall consider only those frequencies up to half of the sampling frequency.

You can read about the Nyquist-Shannon sampling theorem here, but only if you feel like it: http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

6.2.1. Function timeToFreq

Create a new function called **timeToFreq**. This function will convert a sound signal vector (which is by definition in the time domain) into a vector of power levels in the frequency domain. You don't have to understand what this all means, you just have to understand how to write a function.

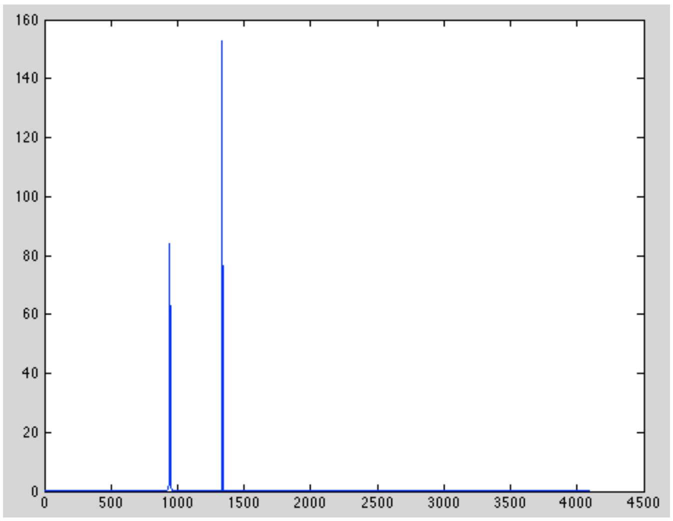
This function will have two parameters, the sound signal and the number of samples per second, and it must return two values, the frequency vector and the power vector.

I have written the body of the function for you already, shown below. All you need to do is write the function header line and put the word **end** at the end. The variables I use here give you an indication as to what you need to call the parameters and return variables.

```
  count = length(signal);
  y = fft(signal, count);
  power = y.*conj(y)/count;
  power = power(1:floor(count/2));
  freq = samplesPerSec/count*(0:(floor(count/2)-1));
```

Test the function to be sure it works correctly.

```
  sig = touchTone('0');
  [f p] = timeToFreq(sig, 8192);
  plot(f, p)
```



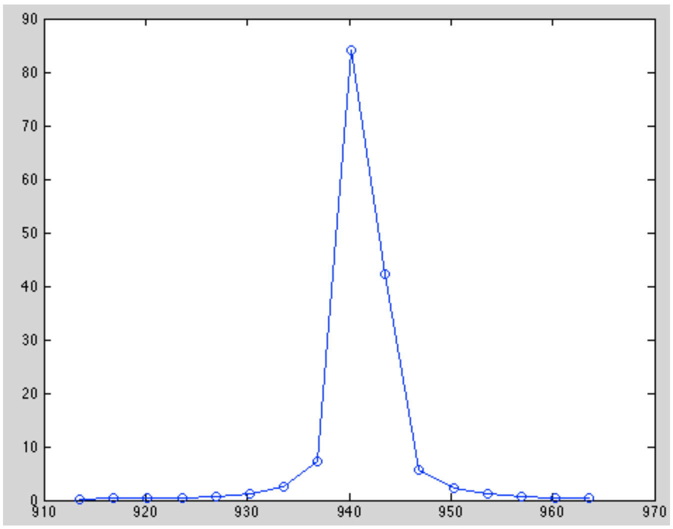
This plot shows two peaks at right about 941 and 1336, which are the two frequencies that make up the tones for the 0 key.

6.2.2. Function peaks

This function takes two vectors: the frequency vector and the power vector, and returns a vector of numbers that indicate what the frequencies are at the peaks of the power graph.

It seems like it could be simple: just figure out all the locations where the power graph is non-zero. However, this problem is made somewhat difficult by the nature of the power data, which is hard to see on the previous plot. Let's zoom in on the first peak:

```
  plot(f(275:290), p(275:290), '-o')
```



From this plot you can see that there are several non-zero data points surrounding the 941Hz location, and it even seems that the peak might be at 940 and not 941. This complicates things a bit.

However, notice that the peak occurs at that location where the graph changes from rising (the left slope going up to the peak) to falling (the right slope going down from the peak). This is the definition of a peak in a graph. We need to ask Matlab: At what locations in the vector is the point at that location greater than the point to its left and the points to its right? It turns out that it's not hard to ask that question

If **P** contains all the points, then we need to construct two other vectors: one that contains all the values to the left of **P** (element-by-element), and one that contains all the values to the right of **P**. Here's an example:

```
  P = [1 2 3 3 4 3 2 1]
```
Let's create the vector of left elements:
```
  Left = [0 P(1:end-1)]
```
which is the same as:
```
  Left = [0 1 2 3 3 4 3 2]
```

And the right elements:

```
  Right = [P(2:end) 0]
```
Which is the same as:
```
  Right = [2 3 3 4 3 2 1 0]
```

Consider location 3. **Vect(3)** contains the value 3. To its left is location 2, which contains 2, and to its right is location 4, which contains 3. These same numbers are also found in **Left(3)** and **Right(3)**. But now look at location 5: **Vect(5)** is greater than **Left(5)** *and* **Right (5)**.

```
  >> P > Left & P > Right
  ans =
      0  0  0  0  1  0  0  0
```

(The ampersand means *and also*.) This is the logical mask showing where the elements in **P** are greater than the elements in **Left**, *and* the elements in **P** are greater than the elements in **Right**. This expression is true in only one location. If there were multiple peaks it would be true in multiple locations. Let's ask for the location of the peak:

```
  >> find(P > Left & P > Right)
  ans =
      5
```

Let's get the value at that location:

```
  >> P(P > Left & P > Right)
  ans =
      4
```

The value 4 is the peak value. This works correctly. Notice that I did not use the **find** function here, instead I used a vector of logical values. This is known as logical indexing. The vector of logical values is used as the vector index, and Matlab returns the vector elements where the logical values are true.

Write a function called **peaks** that takes a vector of frequencies and a vector of power values. This function must return the frequencies at which the peaks occur. This must be done in several steps:

1. Determine the locations in the power vector where the peaks occur.
2. Return the values from the frequencies vector at those locations.
3. Call the round function on the vector as the last thing you do. This keeps the output looking nicer. You can do it like this if you want to:
```
        returnVariable = round(returnVariable);
```

Now test the **peaks** function:

```
  >> sig = touchTone('0');
  >> [f p] = timeToFreq(sig, 8192);
  >> peaks(f, p)
  ans =
    940  1337
```

Hm, these numbers are close to the original frequencies of 941 and 1336, but not exactly equal. We can deal with this discrepancy in the next few functions.

6.2.3. Function closeTo

I'm tired of writing this assignment already. ;) Create a new function called **closeTo**, and copy and paste the function below.

This determines if a number **a** is within **epsilon** units of another number **b**, where **epsilon** is defined to be 5.

```
function result = closeTo(a, b)
  epsilon = 5;
  result = abs(a - b) <= epsilon;
end
```

This allows then inexact comparison of two numbers. Test it:

```
  >> closeTo(940, 941)
  ans =
      1
  >> closeTo(940, 946)
  ans =
      0
```

6.2.4. Function freqToName

I'm still tired of writing this assignment, and this is stuff you don't quite know yet. Copy and paste this as a new function into your project, but see if you can understand how it works.

This function converts a pair of frequencies into the name of a key on the telephone keypad. It is the opposite of the **nameToFreq** function.

```
function name = freqToName(f1, f2)
  if closeTo(f1, 697)
    if closeTo(f2, 1209)
```

```
      name = '1';
    elseif closeTo(f2, 1336)
      name = '2';
    elseif closeTo(f2, 1477)
      name = '3';
    elseif closeTo(f2, 1633)
      name = 'A';
    end
  elseif closeTo(f1, 770)
    if closeTo(f2, 1209)
      name = '4';
    elseif closeTo(f2, 1336)
      name = '5';
    elseif closeTo(f2, 1477)
      name = '6';
    elseif closeTo(f2, 1633)
      name = 'B';
    end
  elseif closeTo(f1, 852)
    if closeTo(f2, 1209)
      name = '7';
    elseif closeTo(f2, 1336)
      name = '8';
    elseif closeTo(f2, 1477)
      name = '9';
    elseif closeTo(f2, 1633)
      name = 'C';
    end
  elseif closeTo(f1, 941)
    if closeTo(f2, 1209)
      name = '*';
    elseif closeTo(f2, 1336)
      name = '0';
    elseif closeTo(f2, 1477)
      name = '#';
    elseif closeTo(f2, 1633)
      name = 'D';
    end
  end
end
```

Test it:

```
  >> freqToName(940,1337)
  ans =
  0
```

This looks like the value for the number 0 or even the logical value **false**, but in fact this is the character string '0'. Notice how the answer is up against the left margin. That indicates that the answer is a character string and not a number. Try different frequencies:

```
  >> freqToName(943, 1630)
```

```
ans =
D
```

6.3. Tie Them Together

The last function you will write will simply connect the sound synthesis portion of your project to the sound analysis portion of your project. Thankfully, it's not too difficult.

Create a new function called **encodeAndDecode**. It has one parameter: this will be the name of the tone to encode. This function also returns one value: the name of the decoded tone. Inside the function do this:

1. Generate a touch tone signal having the given name by calling the **touchTone** function.
2. Convert the signal to its frequency and power vectors by calling the **timeToFreq** function.
3. Generate the vector of peaks by calling the **peaks** function. The function call will return a vector having two elements in it.
4. Convert the two vector elements back into a name by calling the **freqToName** function.

Test it:

```
>> encodeAndDecode('0')
ans =
0
>> encodeAndDecode('1')
ans =
1
>> encodeAndDecode('*')
ans =
*
>> encodeAndDecode('A')
ans =
A
```

7. Report

Create a Microsoft Word or Libre Word file (or some other format that your TA agrees on -- ask him or her if you are not sure). Save the file with the name **Project3** with a .doc or .docx format.

At the beginning of the document include this information:

Tone Encoding and Decoding
CSE1010 Project 3, Fall 2012
Your name goes here
The current date goes here
TA: *Your TA's name goes here*
Section: *Your section number goes here*
Instructor: Jeffrey A. Meunier

Be sure to replace the parts that are underlined above.

Now create the following three sections in your document.

1. Introduction

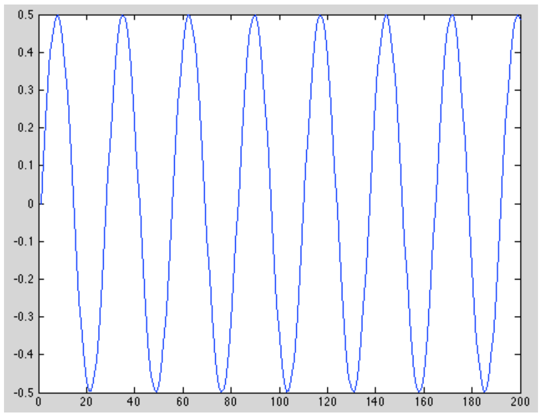
In this section copy & paste the text from the first paragraph of the introduction section of this assignment. (It's not plagiarism if you have permission to copy something. I give you permission.)

2. Functions

Create one subsection for each function that you wrote (even the ones I gave you). In each section, summarize in one sentence what the function does, and show how to use the function as best you can, and a subset of the return value or values. If the result is a vector, insert a plot of either the whole vector or part of the vector (if a part of the vector is more interesting than the whole vector). Here's an example:

The generateTone function generates a sinusoidal tone given some frequency, number of samples per second, and the number of seconds the tone should last.

```
>> t = generateTone(300, 8192, 1);
>> t(1:10)
ans =
  Columns 1 through 8
    0  0.1140  0.2221  0.3184  0.3980  0.4565  0.4910  0.4996
  Columns 9 through 10
    0.4819  0.4387
>> plot(t(1:200))
```



3. Source code

Copy & paste the contents of all your functions here. Insert a blank line between functions. You do not need to write anything.

8. Submission

Submit the following two things things on HuskyCT:

1. All the .m files for this project stored in a Zip file. Do not upload each separate .m file on HuskyCT. I have made tutorial videos available on HuskyCT describing how to make a Zip file. These are the files you must include in the Zip file:
   - closeTo.m
   - encodeAndDecode.m
   - freqToName.m
   - generateTone.m
   - nameToFreq.m
   - peaks.m
   - synthDtmf.m
   - timeToFreq.m
   - touchTone.m
2. The MS Word document.

If for some reason you are not able to submit your files on HuskyCT, email your TA before the deadline. Attach your files to the email.

9. Notes

Here are some notes about working on this project:

- Type all the examples that I provide. They are intended to teach you a lot of material you will need to know.

10. Grading Rubric

Your TA will grade your assignment based on these criteria:

- (2 points) The comment block at the beginning of the program file is correct.
- (6 points) The program displays the correct answers.
- (4 points) The program uses the correct calculations to generate the answers.
- (4 points) The program is formatted neatly. Follow any example in the book, or see the web site here: https://sites.google.com/site/jeffscourses/cse1010/matlab-program-formatting
- (4 points) The document contains all the correct information and is formatted neatly.

11. Getting help

Start your project early, because you will probably not be able to get help in the last few hours before the project is due.

- If you need help, send e-mail to your TA immediately.
- Go to the office hours for (preferably) your TA or any other TA. I suggest you seeing your own TA first because your TA will know you better. However, don't let that stop you from seeing any TA for help.
- Send e-mail to Jeff.
- Go to Jeff's office hours.