CS100M Spring 2008 Project 6 due Thursday 5/1 at 6pm

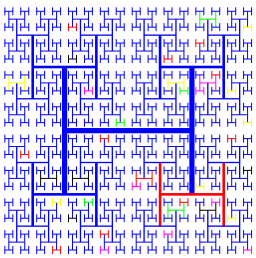
You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

Objectives

There are two questions in this project. You will write a recursive function to produce a fractal pattern in Question 1. In Question 2, you will design and implement a method to "decode" an 11-digit phone number given only the noisy acoustic signals of a touchtone phone. In Question 2, you need to *evaluate* your method to determine how much noise it can withstand.

1 H-Tree

In this problem you will write a recursive function `drawHTree` to draw an "H-tree." Note that you'll use recursion—no loops! The idea is that you start with one 'H', then add four smaller 'H's, one at each end of the previous 'H', then add four more 'H's at each end point, and again, and ...



The H-tree is isn't just an interesting fractal that one can draw; it is the pattern used in laying out an interconnection network on microprocessor chips! Such a network has the feature that at any "level" the ends of the 'H' are the same distance from the center of the network. Consider a 2-level H-tree. The ends

of the smaller 'H's are the same distance from the center of the network. In microprocessor chip design this is important because the delay in routing time signal from the center clock to each part (end point) of the network is then the same. The H-tree is also a space-efficient layout for cramming components onto a chip!

Function `drawHTree` has the following specifications:

```
function drawHTree(x,y,len,w,level)
% Draw recursively an H-tree centered at (x,y).
% Each of the three lines of the 'H' is of length len and width w.
% level is an integer that indicates the level of the recursion.
```

At each level of recursion, the length and width of each line of the 'H' are halved. From Figure 1, you see that most of the 'H's are drawn in blue. Write your function so that there is a *one in ten* probability that an 'H' is drawn in a color that is not blue. In our implementation, one of the following colors are chosen randomly when a "non-blue" color is needed: black, red, yellow, magenta, green. (*Hint*: we used the string `'krymg'` somehow.) You can choose your own colors, but your code should allow for at least four different colors to be used.

Start by reviewing the `drawTriangle` function from Lecture 25 (4/22). Your function `drawHTree` should have a very similar organization. In fact, `drawHTree` is simpler because there's nothing to do when the maximum recursion level is reached. Use a maximum recursion level of four. Use the `plot` function instead of `fill`. You can control the line width by setting the `linewidth` property (see Appendix A of *FVL* or the show random walk example on 3/6).

As usual, decompose the problem! Start by drawing the H-tree in just one color. After you have tested your function to see that it draws the structure correctly, then implement the random color functionality.

Submit your function `drawHTree`. You can assume that `hold` is on and that axes limits are appropriately set. For example, we can call `drawHTree` with the following script:

```
% Show drawHTree
close all
figure
x= 0;
y= 0;
len= 10;
width= 5;  % corresponds to the "linewidth" property in function plot
axis equal off
axis([-1.5*len 1.5*len -1.5*len 1.5*len])
hold on
drawHTree(x,y,len,width,0)
hold off
```

2 Decode a phone number

Some people are able to tell which number has been dialed on a touchtone phone simply by listening to the tone produced when a button is pressed. You will write a MATLAB program to decode the acoustic signal of an 11-digit call made on a touchtone phone. The `findNumber` function we discussed in Lecture 23 (4/15) did part of the job, but user interaction was required in segmenting the acoustic signal and in identifying the two peaks (frequency) for each digit. You will modify function `findNumber` to *automate* the entire decoding process!

Start by reviewing the `makeCall` and `findNumber` functions from Lecture 23 (4/15). Run *and read* the functions to make sure that you really understand the steps involved.

In this project question, we complicate the simulated call signal further by making the duration of a button press random and adding a random duration of "silence" before the first button press. This revised `makeCall` function is posted on the *Projects* page. You can assume that only 11-digit calls will be made.

This is an open question! We expect that students will have different ideas and algorithms for automating the segmentation of the signal and peak finding. You must comment your code clearly *and concisely* to explain your methods. You must also *evaluate your method*. Tell us how much noise it can withstand (20%, 30%, ...?). Can your method handle calls that are not 11 digits? If so, how does the "noise handling ability" change depending on the number of digits? (Try realistic cases: 3-, 7-, 10-, and 11-digit calls). If your method cannot handle a variable number of digits, describe how it would need to be changed to accommodate a variable number of digits. Write and submit a file `eval.*` that contains your evaluation. (`*` means that you can submit a file of any reasonable type that contains your written evaluation—txt, doc, pdf, etc.) The evaluation should be about half a page.

Your score on this question depends on several things:

- Robustness of your method — this has to do with the amount of noise that your method can deal with. Your method should be able to handle at least 20% noise (i.e., `noiseVal` of 0.2 when you generate test signals for the call). If you can get to 50% it is good.
- Your evaluation — is it correct and clearly written? Test your function thoroughly in order to write the evaluation.
- Correctness and good programming style — the general things that we look for, similar to previous projects.

We hope that you will have fun with this project!