CPSC 535
Assignment 1: Introduction to Matlab and Octave

The goal of this assignment is to become familiar with Matlab and Octave as tools for scientific investigation. Matlab, a product of Mathworks, is an interpreted language whose singular feature is that its basic data type is an array. Even a scalar value in Matlab is represented as a 1×1 array. This feature reduces the lines of code required to implement many numerical algorithms and thus makes scientific programming easier (and more fun). Matlab also features an extensive function library of numerical algorithms to solve most common problems and some that are not so common.

Octave is a program that imitates Matlab (at least up to about version 4) and is distributed under the Gnu Public License (see www.gnu.org). This means Octave is free. Octave development is currently slow to non-existent and does not have many of the new features in the latest version of Matlab. Also, Octave libraries are not as extensive as those for Matlab. Still, its price makes it attractive and we can use it to learn about computer vision.

This assignment takes you through some examples that compute fractal dimensions of Koch curves. The examples have little to do with computer vision. They are merely pedagogical exercises. The fractal concepts are taken from Smith [2]. Purves *et al.* [1] describe the Cornsweet effect.

Note that this assignment describes tasks in mathematical terms throughout. It is up to you to translate the mathematics into effecient Octave code.

1 The Koch Curve

The Koch curve, also known as the snowflake curve, appears often in computer science instruction because it makes pretty pictures while demonstrating concepts in recursion. Figure 1(a) shows pictorially how to construct Koch curves. To move from K_n to K_{n+1}, take each straight line segement in K_n, divide it into thirds, and replace the middle section with two segments one third of the length of the original segment.

To aid the implementation of Koch curve computations consider Figure 1(b). Suppose X_0 and X_1 are segment end points on K_n. Then X_0, X_1, X_2, X_3, and X_4 are all segment end points on K_{n+1}. If $X_i = (x_i, y_i)$, then

$$\begin{aligned} X_2 &= \frac{2}{3}X_0 + \frac{1}{3}X_1 \\ X_3 &= \frac{1}{2}X_0 + \frac{1}{2}X_1 + \frac{\sqrt{3}}{6}(y_0 - y_1, x_1 - x_0) \\ X_4 &= \frac{1}{3}X_0 + \frac{2}{3}X_1 \end{aligned}$$

Write an Octave function called **koch** that computes K_{n+1} from K_n. I recommend that you represent the curves as an array with two columns. Each row in the array will correspond to a segment end point with the x-coordinate in the first column and y-coordinate in the second column. If you order the end point correctly, this representation will allow you to plot the Koch curves easily using the gplot command.

Write a script that generates K_{10} and plots it. For K_0 use a line segment from $(0, 0)$ to $(1, 0)$.

For this section hand in your code for **koch**, the script, and your plot of K_{10}.

2 The Fractal Dimension of K_∞

In an imaginary mathematical universe we can continue to divide Koch curves forever. The limiting curve is K_∞ which is a fractal. There are different definitions of *fractal dimension*, but we will work with just one, the *box* dimension.

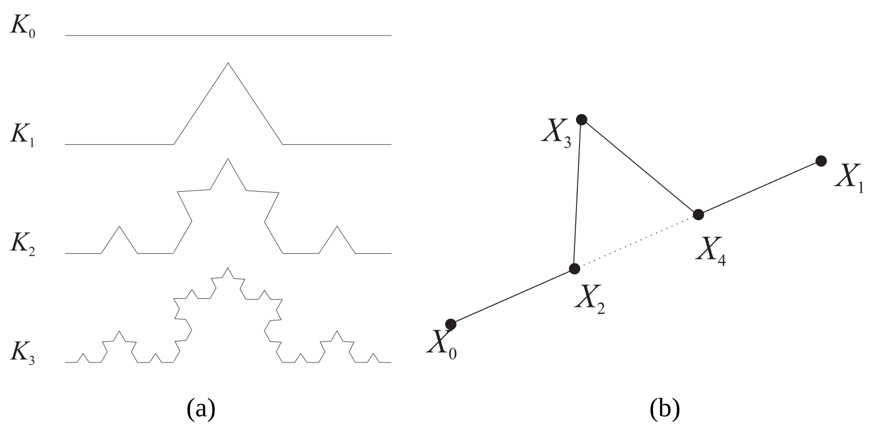

Figure 1: Construction of Koch (or snowflake) curves.

Suppose we carve up the space surrounding K_∞ into boxes of size δ by δ. We can check the boxes to see how many of them contain a part of the curve[1]. This gives us a measurement of the curve for box size δ. If the boxes are smaller, the count goes up. If the shape is not a fractal (e.g. a straigt line) then reducing box size to $\delta/2$ should double the box count. In general for fractals we have

$$\text{proportional increase in the number of occupied boxes} = \text{reduction factor in box size}^d$$

where d is the fractal dimension of the curve.

This suggests the following algorithm to numerically estimate d for K_∞ (note, the algorithm uses 3 for the reduction fator in box size).

```
$\delta = 1$
**for** $r$ = 0 **to** $r_{max}$ **do**
    $n_r = 0$
    **for** all $\frac{\delta}{3^r} \times \frac{\delta}{3^r}$ boxes **do**
        **if** Koch curve in box **then**
            $n_r = n_r + 1$
```

This gives a set of box counts, $\{n_r\}$. To apply the formula we need to know the proportional increases, n_1/n_0, n_2/n_1, and so on. You can average them all together as follows

$$\text{mean proportional increase in box count} = \frac{1}{r_{max}} \sum_{r=0}^{r_{max}-1} \frac{n_{r+1}}{n_r}$$

Once you have the mean proportional increase you can apply the equation for the fractal dimension.

Write an Octave script to estimate the fractal dimension of K_∞. I suggest you use K_5 as your sample curve and $r_{max} = 5$.

Hand in your script and the estimate of the fractal dimension.

3 So What About Images?

Since this is a computer vision course it is necessary to acquire and display images. In this part of the assignment you will create an image that illustrates the *Cornsweet* effect.

Write an Octave script to do the following:

[1]For your code, it is sufficient to test the enpoints of curve segments only.

1. create a row vector of 50 elements using the following equation

$$x_i = \begin{cases} 0 & i \leq 25 \\ (\frac{i-25}{25})^2 & i > 25 \end{cases}$$

2. from the previous step, build a 100-element row vector in which the first 50 elements are the x_i and the second 50 elements are the x_i negated and in reverse order.

3. create an array that consists of 50 duplicate rows, each identical to the 100-element row vector.

Display this array as an image using a 256-value gray scale.
Hand in your script and a copy of the resulting image.

Fast Octave Code

The Octave manual contains several tips for writing fast Octave programs. I recommend that you read it.

The biggest problem that I have observed when students learn Octave is that they tend to rely on their C/C++ knowledge too much. If you write in Octave by translating from C/C++ directly into Octave, it is likely that your program will be painfully slow. When marking assignments in the past I have been able to tell with a second or two of starting a students program whether or not they have grasped Octave. The difference in execution speed is very obvious.

Hand In

1. Cover sheet with your name, course number, and assignment number only.

2. Name and student ID number inside the cover sheet.

3. All plots, images, and the fractal dimension estimate. Make sure you label the plots to indicate what they are.

4. E-mail your **.m** file of Octave code to the TA.

Marking

Assignment grades will be based on

1. the correctness of the plots and images, and

2. the quality of your Octave code.

Your code should be correct, readable, well-documented, and modular. Octave and Matlab are not very readable languages, but you should still be able to produce good-quality code that is maintainable.

Collaboration

The assignment must be done individually so everything that you hand in must be your original work, except for the code copied from a cited source or that supplied by your instructor. When someone else's code is used like this, you must acknowledge the source explicitly. Copying work that is not your own without acknowledgement is academic misconduct. Contact your instructor if you have problems.

References

[1] D. Purves, R. B. Lotto, and S. Nundy. Why we see what we do. *American Scientist*, 90(3):236–243, 2002.

[2] P. Smith. *Explaining Chaos*. Cambridge University Press, Cambridge, UK, 1998.