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 and is distributed under the Gnu Public License (see www.gnu.org). This means Octave is free. Octave development is generally a few versions behind Matlab 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 several examples that show how Matlab/Octave style programming can really save you a great deal of time and effort for certain kinds of programming: time that you will hopefully save on the other assignments for this course. As such, code-efficiency is a large component of this assignment. Parallelise your code and do array operations whenever possible. **AVOID FOR LOOPS AT ALL COSTS**. The results are quite astounding: for example, in previous years, some part-2 submissions took more than 20 minutes to compute, while the TA's version takes less than 0.5s.

There are three pieces to this assignment: the cornsweet effect, a fractal Koch curve, Haar wavelets. The fractal concepts are taken from Smith [4]. Haar wavelets are described in detail in [1, 2]. Purves *et al.* [3] describe the Cornsweet effect. Note that this assignment describes tasks in mathematical terms throughout. It is up to you to translate the mathematics into efficient Octave code. Furthermore, you can treat some of this as a mathematical review. The Haar wavelets in particular leverage linear algebra, skills you will need for the rest of the course.

1 Warming up with 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. I highly recommend researching this effect on the internet, as in the past students often hand-in incorrect results that are easy to fix.

Write an Octave script to do the following:

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. Use the `colormap` and `image` commands to do this. Hand in your script and a copy of the resulting image.

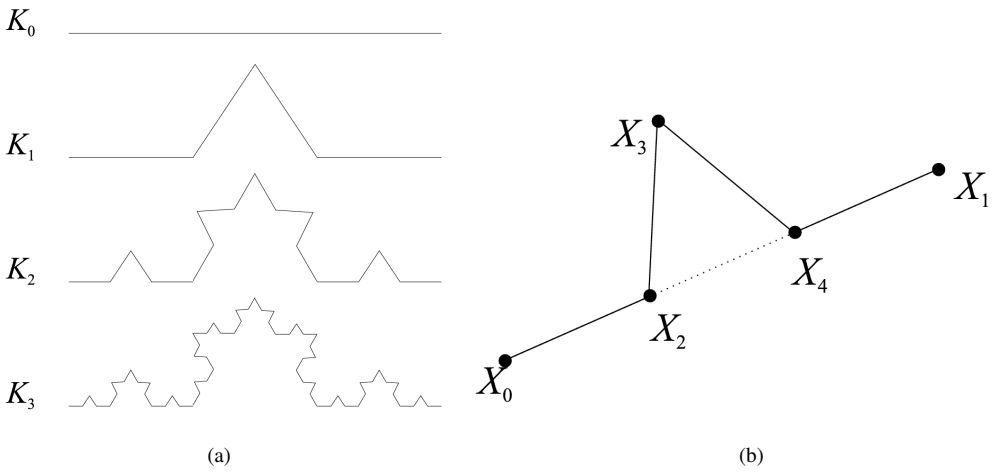


Figure 1: Construction of Koch (or snowflake) curves.

2 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. Put this in its own `koch.m` file; you can find details on this on-line in the octave or Matlab documentation. 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 `plot` command. You can save this plot to disk using the `print` command. e.g., `print -dpng foo.png` after showing a plot to screen.

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

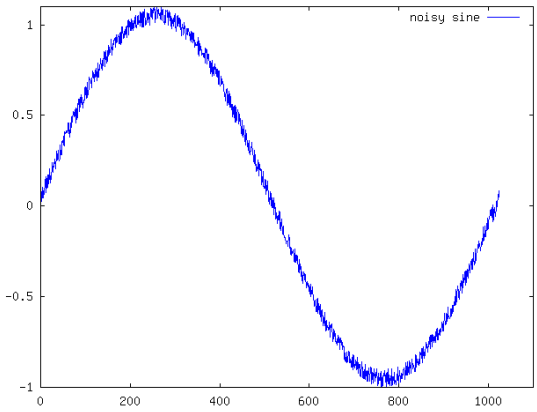
For this section hand in your code for **koch**, the script, and your plot of K_{10}.

3 Haar Wavelets

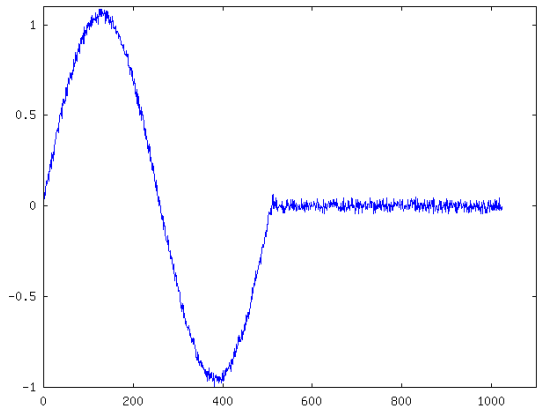
Part 1

Wavelets are tools that aid in frequency analysis by dividing a given function or signal into different frequency components. You will learn more about frequency analysis later in the course. For now, given a set of data, visualise the high-frequency data as the rapid changes, or detail, of the data. The low-frequency data is the slow changes, or the general trend, of the data. In Figure 2(a), the low-frequency data is the general shape of the sine curve. The high-frequency detail is the noise, or the *fuzz* on the sine curve.

The Haar wavelet is the simplest possible wavelet and splits data into its high and low frequency components. A single application of Haar decomposition does a frequency cut at $f/2$ where f is the frequency of the original



(a) A noisy sine curve



(b) First Haar decomposition. Resulting curve is on the left, the detail data is on the right.

Figure 2: Haar decomposition of a noisy sine curve.

data. Using Figure 2, notice that there are 1024 samples of data in Figure 2(a). After one Haar decomposition, the sine curve in Figure 2(b) has half the data points (half the frequency) at 512, and also has 512 points of the removed high-frequency data. k successive applications, then, does a frequency cut at $f/2^k$.

The Haar wavelet is a very simple operation that works on pairs of the input discrete data (for continuous data, this works on windows, or regions). The low-frequency component is the average of the two samples and the high-frequency component is half the difference. That is:

$$\begin{aligned} \psi(t) &= \frac{f(2t-1)+f(2t)}{2} \\ \phi(t) &= \frac{f(2t-1)-f(2t)}{2} \end{aligned}$$

Where $\psi(t)$ is the low-frequency data and $\phi(t)$ is the high-frequency data. To reconstruct the data,

$$f(t) = \begin{cases} \psi(\frac{t+1}{2}) + \phi(\frac{t+1}{2}) & t \text{ odd} \\ \psi(\frac{t}{2}) - \phi(\frac{t}{2}) & t \text{ even} \end{cases}$$

In this assignment, you will be removing the high-frequency noise from the sine curve and will add the noise to a cosine curve. First, generate a noisy sine curve with 1024 samples using the octave command

```
sin(0:2*pi/(1024-1):2*pi)+randn(1,1024)/10
```

then do five consecutive Haar decompositions, resulting in a high-frequency cutoff at $f/2^5$, meaning the low-frequency data will only have 32 data points. For each consecutive iteration only operate on the remaining low-frequency data.

Plot the resulting data in a similar fashion to Figure 2(b) (32-data-point sine + 992 data points of detail) and attach it to your assignment. Take special notice how the detail changes on each iteration.

Following, apply this high-frequency data to a cosine curve. Start with a 32-data-point cosine curve and do a five-level haar composition using the detail extracted from your noisy sine. Plot the result and hand it in with your assignment.

HINT: This entire question (Haar) can be done using linear algebra. Setup the pair-wise decomposition as a matrix operation, with the data being a vector. Multiplying the data vector by this matrix will perform a haar decomposition of the entire dataset in a single operation. For successive decompositions, work on a subset of the data, leaving the previously-removed detail untouched. If you manage this, then your five Haar decomposition matrices can be multiplied together, and then applied to your data in one step! Further, you can do the inverse (recomposition) by using the inverse of this matrix (`inv(matrix)`) for free. If you instead do this with for loops, it is much more work and computation will take much longer.

For this section hand in all plots as specified above and all code.

Part 2: Moving to Images

Haar decomposition on images provides several gains including progressive resolution zoom and advantages in compression techniques. Moving from the 1D case in Part 1 to the 2D case is actually quite simple:

1. Do 1D decompositions on each row of the image data.
2. Following, do 1D decompositions on each column of the resulting data. Note that this step and the previous one can be reversed.

The result of this is that you get a low-frequency version of the image separated from large amounts of detail. Please do a 3-level haar decomposition on the attached `cat.png` file (you can choose your own image, but be careful that the image size has a factor of 2^3 to allow for even decomposition).

For this section, hand in your code and the resulting image, and answer the following questions:

- What does the removed high-frequency data look like? What is the range, min, max, and how does this affect how it looks when you draw it with `image`?
- Discuss how Haar decomposition can be useful for image compression.
- What are the limits of Haar decomposition? Why and when would you decompose to this limit?

HINT: Don't over-complicate this. Part 2 is a trivial (one line) extension over Part 1 once the image is loaded. Again, think of this in terms of linear algebra operations. Further, use the `pngload.m` script provided on the professor's website to get the image data loaded and use the `double` function to cast the data to the double datatype for use in your operations.

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, and your resulting mark will reflect this.

Hand In

Hard copy and electronic copy (email to TA):

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 source code. Make sure you label the plots to indicate what they are.

Marking

Assignment grades will be based on

1. the correctness of the plots and images, and
2. the quality and depth of your written answers, and
3. 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 or questions regarding this.

References

[1] Charles K. Chui. *An Introduction to Wavelets*. Academic Press, San Diego, US, 1992.

[2] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69:331–371, 1910.

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

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