Motivating Student Learning Through Applications

Steven Schlicker
schlicks@gvsu.edu

Grand Valley State University

JMM: Thursday, January 17, 2019

Overview

- Background
- Examples
- Big Picture

Background

- My background

Background

- My background
- What do my students need?

Examples of Applications

- Linear Combinations: Analyzing Knight Moves

Application: The Knight's Tour

Chess is a game played on an 8×8 grid which utilizes a variety of different pieces. One piece, the knight, is different from the other pieces in that it can jump over other pieces. However, the knight is limited in how far it can move in a given turn. For these reasons, the knight is a powerful, but often under-utilized, piece.

A knight can move two units either horizontally or vertically, and one unit perpendicular to that. Four knight moves are as illustrated in Figure 1, and the other four moves are the opposites of these.

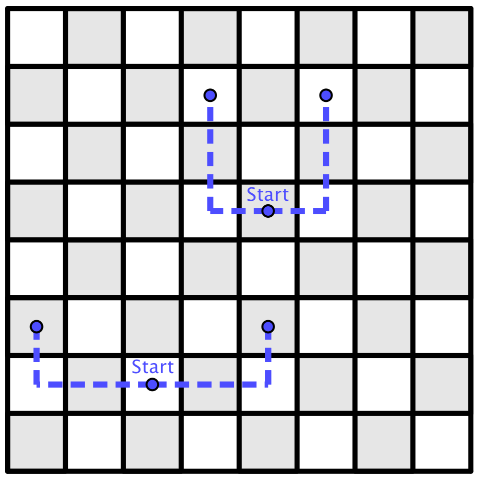

Figure 1: Moves a knight can make.

The knight's tour problem is the mathematical problem of finding a knight's tour, that is a sequence of knight moves so the the knight visits each square exactly once. While we won't consider a knight's tour in this text, we will see using linear combinations of spans of vectors that a knight can move from its initial position to any other position on the board, and that it is possible to determine an sequence of moves to make that happen.

Project: Analyzing Knight Moves

To understand where a knight can move in a chess game, we need to know the initial setup. A chess board is an 8×8 grid. To be able to refer to the individual positions on the board, we will place the board so that its lower left corner is at the origin, make each square in the grid have side length 1, and label each square with the point at the lower left corner. This is illustrated at left in Figure 2.

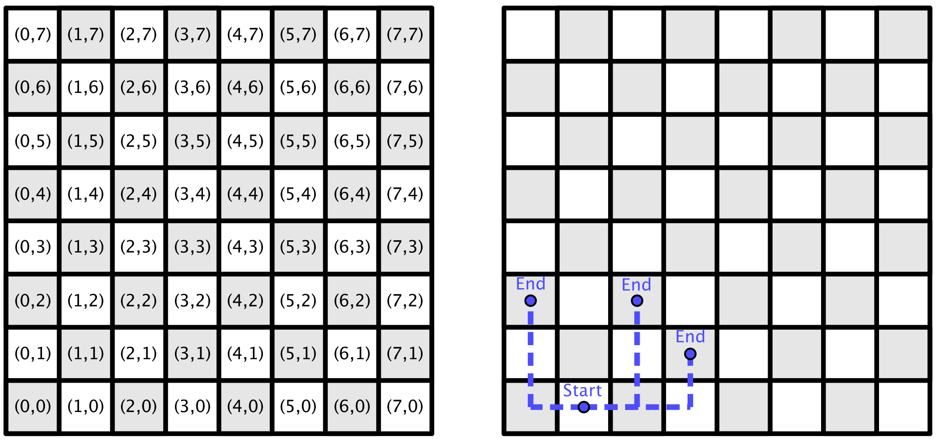

Figure 2: Initial knight placement and moves.

Each player has two knights to start the game, for one player the knights would begin in positions $(1, 0)$ and $(6, 0)$. Because of the symmetry of the knight's moves, we will only analyze the moves of the knight that begins at position $(1, 0)$. This knight has only three allowable moves from its starting point (assuming that the board is empty), as shown at right in Figure 2. The questions we will ask are: given any position on the board, can the knight move from its start position to that position using only knight moves and, what sequence of moves will make that happen. To answer these questions we will use linear combinations of knight moves described as vectors.

Each knight move can be described by a vector. A move one position to the right and two up can be represented at $\mathbf{n}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Three other moves are $\mathbf{n}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$, $\mathbf{n}_3 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, and $\mathbf{n}_4 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$. The other four knight moves are the opposites of these four. Any sequence of moves by the knight is given by the linear combination

$$x_1\mathbf{n}_1 + x_2\mathbf{n}_2 + x_3\mathbf{n}_3 + x_4\mathbf{n}_4.$$

A word of caution: the knight can only make complete moves, so we are restricted to integer (either positive or negative) values for x_1, x_2, x_3, and x_4. You can use the GeoGebra app at `https://www.geogebra.org/m/dfwtskrj` to see the affects the weights have on the knight moves. We should note here that since addition of vectors is commutative, the order in which we apply our moves does not matter. However, we may need to be careful with the order so that our knight does not leave the chess board.

Project Activity 1.

(a) Explain why the vector equation

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} + x_1\mathbf{n}_1 + x_2\mathbf{n}_2 + x_3\mathbf{n}_3 + x_4\mathbf{n}_4 = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

will tell us if it is possible for the knight to move from its initial position at $(1,0)$ to the position (5,2).

(b) Find all solutions, if any, to the system from part (a). If a solution exists, find weights x_1, x_2, x_3, and x_4 to accomplish this move. Is there more than one sequence of possible moves? (Hint: Be careful – we must have solutions in which x_1, x_2, x_3, and x_4 are integers.) You can check your solution with the GeoGebra app at `https://www.geogebra.org/m/dfwtskrj`.

Project Activity 1 shows that it is possible for our knight to move to position $(5,2)$ on the board. We would like to know if it is possible to move to any position on the board. That is, we would like to know if the span of the four moves $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$, and $\mathbf{n}_4$ will allow our knight to cover the entire board. This takes a bit more work.

Project Activity 2. Given any position (a,b), we want to know if our knight can move from its start position $(1,0)$ to position (a,b).

(a) Write a vector equation whose solution will tell us if it is possible for our knight to move from its start position $(1,0)$ to position (a,b).

(b) Show that the solution to part (a) can be written in the form

$$x_1 = \frac{1}{4}\left(-5x_3 + 3x_4 + b + 2(a-1)\right) \tag{1}$$

$$x_2 = \frac{1}{4}\left(3x_3 - 5x_4 + b - 2(a-1)\right) \tag{2}$$

x_3 is free

x_4 is free.

To answer our question if our knight can reach any position, we now need to determine if we can always find integer values of x_3 and x_4 to make equations (1) and (2) have integer solutions. In other words, we need to find values of x_3 and x_4 so that $-5x_3 + 3x_4 + b + 2(a-1)$ and $3x_3 - 5x_4 + b - 2(a-1)$ are multiples of 4. How we do this could depend on the parity (even or odd) of a and b. For example, if a is odd and b is even, say $a = 2r + 1$ and $b = 2s$ for some integers r and s, then

$$\begin{aligned} x_1 &= \frac{1}{4}\left(-5x_3 + 3x_4 + 2s + 4r\right) \\ x_2 &= \frac{1}{4}\left(3x_3 - 5x_4 + 2s - 4r\right). \end{aligned}$$

With a little trial and error we can see that if we let $x_3 = x_4 = s$, then $x_1 = r$ and $x_2 = -r$ is a solution with integer weights. For example, when $a = 5$ and $b = 2$ we have $r = 2$ and $s = 1$. This makes $x_1 = 2$, $x_2 = -2$, $x_3 = 1 = x_4$. Compare this to the solution(s) you found in Project Activity 1. This analysis shows us how to move our knight to any position (a,b) where a is odd and b is even.

Project Activity 3. Complete the analysis as above to determine if there are integer solutions to our knight's move system in the following cases.

(a) a odd and b odd

(b) a even and b even

(c) a even and b odd.

Project Activity 3 shows that for any position on the chess board, using linear combinations of move vectors, we can find a sequence of moves that takes our knight to that position. (We actually haven't shown that these moves can be made so that our knight always stays on the board – we leave that question to you.)

Examples of Applications

- Bases for Vector Spaces: Wavelets

Application: Image Compression

> If you painted a picture with a sky, clouds, trees, and flowers, you would use a different size brush depending on the size of the features. Wavelets are like those brushes.

-Ingrid Daubechies

The advent of the digital age has presented many new opportunities for the collection, analysis, and dissemination of information. Along with these opportunities come new difficulties as well. All of this digital information must be stored in some way and be retrievable in an efficient manner. One collection of tools that is used to deal with these problems is wavelets. For example, The FBI fingerprint files contain millions of cards, each of which contains 10 rolled fingerprint impressions. Each card produces about 10 megabytes of data. To store all of these cards would require an enormous amount of space, and transmitting one full card over existing data lines is slow and inefficient. Without some sort of image compression, a sortable and searchable electronic fingerprint database would be next to impossible. To deal with this problem, the FBI adopted standards for fingerprint digitization using a wavelet compression standard.

Another problem with electronics is noise. Noise can be a big problem when collecting and transmitting data. Wavelet decomposition filters data by averaging and detailing. The detailing coefficients indicate where the details are in the original data set. If some details are very small in relation to others, eliminating them may not substantially alter the original data set. Similar ideas may be used to restore damaged audio,[1], video, photographs, and medical information.[2]

We will consider wavelets as a tool for image compression. The basic idea behind using wavelets to compress images is that we start with a digital image, made up of pixels. Each pixel can be assigned a number or a vector (depending on the makeup of the image). The image can then be represented as a matrix (or a set of matrices) M, where each entry in M represents a pixel in the image. As a simple example, consider the 16×16 image of a flower as shown at left in Figure 1. (We will work with small images like this to make the calculations more manageable, but the ideas work for any size image. We could also extend our methods to consider color images, but for the sake of simplicity we focus on grayscale.) This flower image is a gray-scale image, so each pixel has a numeric representation between 0 and 255, where 0 is black, 255 is white, and numbers between 0 and 255 represent shades of gray. The matrix for this flower image is

$$\begin{bmatrix}
240 & 240 & 240 & 240 & 130 & 130 & 240 & 130 & 130 & 240 & 240 & 240 & 240 & 240 & 240 & 240 \\
240 & 240 & 240 & 130 & 175 & 175 & 130 & 175 & 175 & 130 & 240 & 240 & 240 & 240 & 240 & 240 \\
240 & 240 & 130 & 130 & 175 & 175 & 130 & 175 & 175 & 130 & 130 & 240 & 240 & 240 & 240 & 240 \\
240 & 130 & 175 & 175 & 130 & 175 & 175 & 175 & 130 & 175 & 175 & 130 & 240 & 240 & 240 & 240 \\
240 & 240 & 130 & 175 & 175 & 130 & 175 & 130 & 175 & 175 & 130 & 240 & 240 & 240 & 240 & 240 \\
255 & 240 & 240 & 130 & 130 & 175 & 175 & 175 & 130 & 130 & 240 & 240 & 225 & 240 & 240 & 240 \\
240 & 240 & 130 & 175 & 175 & 130 & 130 & 130 & 175 & 175 & 130 & 240 & 225 & 255 & 240 & 240 \\
240 & 240 & 130 & 175 & 130 & 240 & 130 & 240 & 130 & 175 & 130 & 240 & 255 & 255 & 255 & 240 \\
240 & 240 & 240 & 130 & 240 & 240 & 75 & 240 & 240 & 130 & 240 & 255 & 255 & 255 & 255 & 255 \\
240 & 240 & 240 & 240 & 240 & 240 & 75 & 240 & 240 & 240 & 240 & 240 & 240 & 240 & 240 & 240 \\
240 & 240 & 240 & 75 & 75 & 240 & 75 & 240 & 75 & 75 & 240 & 240 & 240 & 240 & 240 & 240 \\
50 & 240 & 240 & 240 & 75 & 240 & 75 & 240 & 75 & 240 & 240 & 240 & 240 & 50 & 240 & 240 \\
240 & 75 & 240 & 240 & 240 & 75 & 75 & 75 & 240 & 240 & 50 & 240 & 50 & 240 & 240 & 50 \\
240 & 240 & 75 & 240 & 240 & 240 & 75 & 240 & 240 & 50 & 240 & 50 & 240 & 240 & 50 & 240 \\
75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 \\
75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75 & 75
\end{bmatrix}. \tag{1}$$

Now we can apply wavelets to the image and compress it. Essentially, wavelets act by averaging and differencing. The averaging creates smaller versions of the image and the differencing keeps track of how far

[1] see https://ccrma.stanford.edu/groups/edison/brahms/brahms.html for a discussion of the denoising of a Brahms recording

[2] Denoising of Heart Sound Signal Using Wavelet Transform, Gyanaprava Mishra, Kumar Biswal, Asit Kumar Mishra, *International Journal of Research in Engineering and Technology*, ISSN: 2319-1163, Volume: 02 Issue: 04, Apr. 2013.

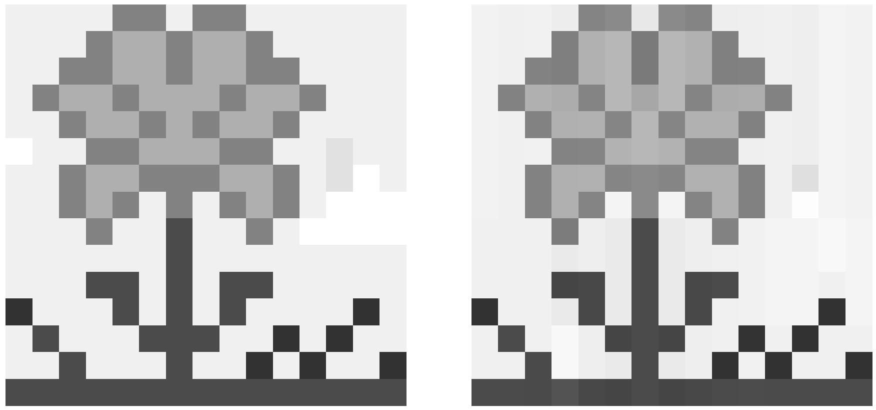

Figure 1: Left: A 16 by 16 pixel image. Right: The image compressed.

the smaller version is from a previous copy. The differencing often produces many small (close to 0) entries, and so replacing these entries with 0 doesn't have much effect on the image (this is called *thresholding*). By introducing long strings of zeros into our data, we are able to store a (compressed) copy of the image in a smaller amount of space. For example, using a threshold value of 10 produces the flower image shown at right in Figure 1.

The averaging and differencing is done with special vectors (wavelets) that form a basis for a suitable function space. More details of this process can be found at the end of this section.

Project: Image Compression with Wavelets

We return to the problem of image compression introduced at the beginning of this section. The first step in the wavelet compression process is to digitize an image. There are two important ideas about digitalization to understand here: intensity levels and resolution. In grayscale image processing, it is common to think of 256 different intensity levels, or scales, of gray ranging from 0 (black) to 255 (white). A digital image can be created by taking a small grid of squares (called pixels) and coloring each pixel with some shade of gray. The resolution of this grid is a measure of how many pixels are used per square inch. An example of a 16 by 16 pixel picture of a flower was shown in Figure 1.

An image can be thought of in several ways: as a two-dimensional array; as one long vector by stringing the columns together one after another; or as a collection of column vectors. For the sake of simplicity, we will use the latter approach in this exercise. We call each column vector in a picture a *signal*. Wavelets are used to process signals. After processing we can apply some technique to compress the processed signals.

To process a signal we select a family of wavelets. There are many different families of wavelets – which family to use depends on the problem to be addressed. The simplest family of wavelets is the Haar family. More complicated families of wavelets are usually used in applications, but the basic ideas in wavelets can be seen through working with the Haar wavelets, and their relative simplicity will make the details easier to follow. Each family of wavelets has a father wavelet (usually denoted ϕ) and a mother wavelet (ψ).

Wavelets are generated from the mother wavelet by scalings and translations. To further simplify our work we will restrict ourselves to wavelets on [0,1], although this is not necessary. The advantage the wavelets have over other methods of data analysis (Fourier analysis for example) is that with the scalings and translations we are able to analyze both frequency on large intervals and isolate signal discontinuities on very small intervals. The way this is done is by using a large collection (infinite, in fact) of basis functions with which to transform the data. We'll begin by looking at how these basis functions arise.

If we sample data at various points, we can consider our data to represent a piecewise constant function obtained by partitioning [0,1] into n equal sized subintervals, where n represents the number of sample points. For the purposes of this project we will always choose n to be a power of 2. So we can consider all of our data to represent functions. For us, then, it is natural to look at these functions in the vector space of all functions from $\mathbb{R}$ to $\mathbb{R}$. Since our data is piecewise constant, we can really restrict ourselves to a subspace of this larger vector space – subspaces of piecewise constant functions. The most basic piecewise constant function on the interval $[0, 1]$ is the one whose value is 1 on the entire interval. We define ϕ to be this constant function (called the characteristic function of the unit interval). That is

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0, & \text{otherwise.} \end{cases}$$

This function ϕ is the Father Haar wavelet.

This function ϕ may seem to be a very simple function but it has properties that will be important to us. One property is that ϕ satisfies a scaling equation. For example, Figure 2 shows that

$$\phi(x) = \phi(2x) + \phi(2x - 1)$$

while Figure 3 shows that

$$\phi(x) = \phi(2^2x) + \phi(2^2x - 1) + \phi(2^2x - 2) + \phi(2^2x - 3).$$

So ϕ is a sum of scalings and translations of itself. In general, for each positive integer n and integers k

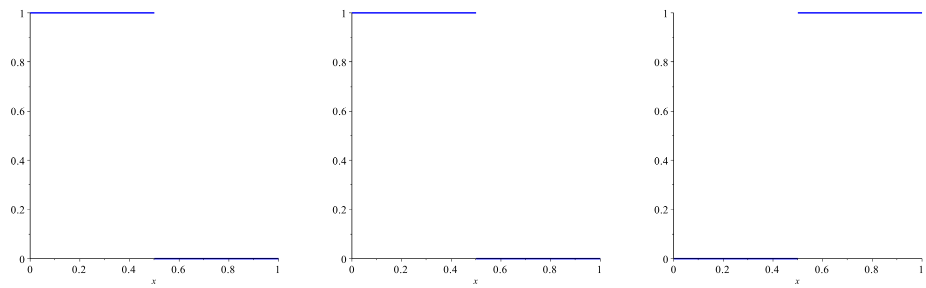

Figure 2: Graphs of $\phi(x)$, $\phi(2x)$, and $\phi(2x-1)$ from left to right.

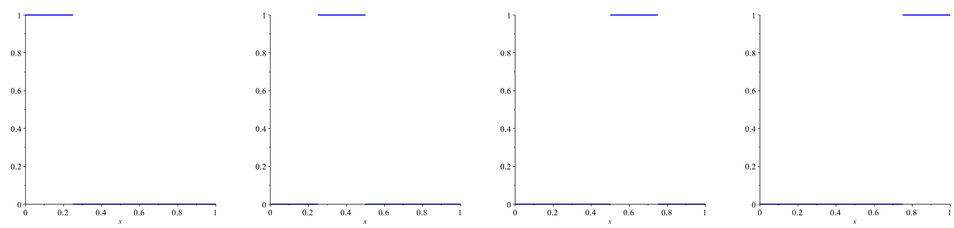

Figure 3: Graphs of $\phi(2^2x)$, $\phi(2^2x-1)$, $\phi(2^2x-2)$, and $\phi(2^2x-3)$, from left to right.

between 0 and $2^n - 1$ we define

$$\phi_{n,k}(x) = \phi\left(2^n x - k\right).$$

Then $\phi(x) = \sum_{k=0}^{2^n-1} \phi_{n,k}(x)$ for each n.

These functions $\phi_{n,k}$ are useful in that they form a basis for the vector space V_n of all piecewise constant functions on $[0,1]$ that have possible breaks at the points $\frac{1}{2^n}$, $\frac{2}{2^n}$, $\frac{3}{2^n}$, $\ldots$, $\frac{2^n-1}{2^n}$. This is exactly the kind of space in which digital signals live, especially if we sample signals at 2^n evenly spaced points on $[0,1]$. Let $\mathcal{B}_n = \{\phi_{n,k} : 0 \le k \le 2^n - 1\}$. You may assume without proof that $\mathcal{B}_n$ is a basis of V_n.

Project Activity 1.

(a) Draw the linear combination $2\phi_{2,0} - 3\phi_{2,1} + 17\phi_{2,2} + 30\phi_{2,3}$. What does this linear combination look like? Explain the statement made previously "Notice that these 2^n functions $\phi_{n,k}$ form a basis for the vector space of all piecewise constant functions on $[0,1]$ that have possible breaks at the points $\frac{1}{2^n}$, $\frac{2}{2^n}$, $\frac{3}{2^n}$, $\ldots$, $\frac{2^n-1}{2^n}$".

(b) Remember that we can consider our data to represent a piecewise constant function obtained by partitioning $[0,1]$ into n subintervals, where n represents the number of sample points. Suppose we collect the following data: 10, 13, 21, 55, 3, 12, 4, 18. Explain how we can use this data to define a piecewise constant function f on $[0,1]$. Express f as a linear combination of suitable functions $\phi_{n,k}$. Plot this linear combination of $\phi_{n,k}$ to verify.

Working with functions can be more cumbersome than working with vectors in $\mathbb{R}^n$, but the digital nature of our data makes it possible to view our piecewise constant functions as vectors in $\mathbb{R}^n$ for suitable n. More specifically, if f is a function in V_n, then f is piecewise constant functions on $[0,1]$ with possible breaks at the points $\frac{1}{2^n}$, $\frac{2}{2^n}$, $\frac{3}{2^n}$, $\ldots$, $\frac{2^n-1}{2^n}$. If f has the value of y_i on the interval between $\frac{i-1}{2^n}$ and $\frac{i}{2^n}$, then we can identify f with the vector $[y_1\ y_1\ \ldots\ y_{2^n}]^{\mathsf{T}}$.

Project Activity 2.

(a) Determine the vector in $\mathbb{R}^8$ that is identified with ϕ.

(b) Determine the value of m and the vectors in $\mathbb{R}^m$ that are identified with $\phi_{2,0}$, $\phi_{2,1}$, $\phi_{2,2}$, and $\phi_{2,3}$.

We can use the functions $\phi_{n,k}$ to represent digital signals, but to manipulate the data in useful ways we need a different perspective. A different basis for V_n (a *wavelet basis*) will allow us to identify the pieces of the data that are most important. We illustrate in the next activity with the spaces V_1 and V_2.

Project Activity 3. The space V_1 consists of all functions that are piecewise constant on $[0, 1]$ with a possible break at $x = \frac{1}{2}$. The functions $\phi = \phi_{n,k}$ are used to records the values of a signal, and by summing these values we can calculate their average. Wavelets act by averaging and differencing, and so ϕ does the averaging. We need functions that will perform the differencing.

(a) Define $\{\psi_{0,0}\}$ as

$$\psi_{0,0}(x) = \begin{cases} 1 & \text{if } 0 \leq x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq x < 1 \\ 0 & \text{otherwise} \end{cases}.$$

A picture of $\psi_{0,0}$ is shown in Figure 4. Since $\psi_{0,0}$ assumes values of 1 and -1, we can use $\psi_{0,0}$ to perform differencing. The function $\psi = \psi_{0,0}$ is the Mother Haar wavelet.[3] This Haar wavelet is nice in that it has what is called *compact support* (it is 0 outside of a small interval). Show that $\{\phi, \psi\}$ is a basis for V_1.

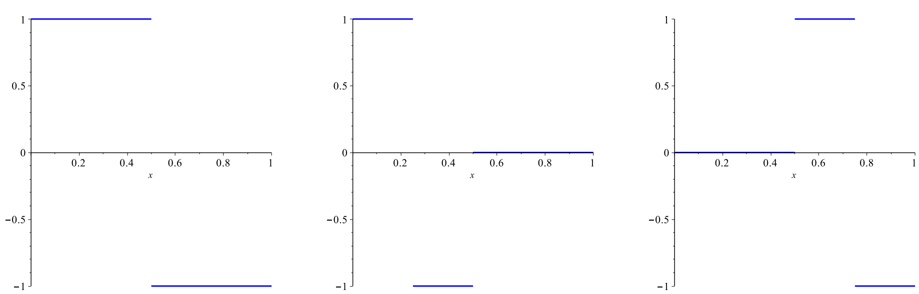

Figure 4: The graphs of $\psi_{0,0}$, $\psi_{1,0}$ and $\psi_{1,1}$ from left to right.

(b) We continue in a manner similar to the one in which we constructed bases for V_n. For $k = 0$ and $k = 1$, let $\psi_{1,k} = \psi\left(2^1 x - k\right)$. Graphs of $\psi_{1,0}$ and $\psi_{1,1}$ are shown in Figure 4. The functions $\psi_{1,k}$ assume the values of 1 and -1 on smaller intervals, and so can be used to perform differencing on smaller scale than $\psi_{0,0}$. Show that $\{\phi_{0,0}, \psi_{0,0}, \psi_{1,0}, \psi_{1,1}\}$ is a basis for V_2.

As Project Activity 3 suggests, we can make a basis for V_n from $\phi_{0,0}$ and functions of the form $\psi_{n,k}$ defined by $\psi_{n,k}(x) = \psi\left(2^n x - k\right)$ for k from 0 to $2^n - 1$. More specifically, if we let $\mathcal{S}_n = \{\psi_{n,k} : 0 \leq k \leq 2^n - 1\}$, then the set

$$\mathcal{W}_n = \{\phi_{0,0}\} \cup \bigcup_{j=0}^{n-1} \mathcal{S}_j$$

is a basis for $V_n^{\perp}$ (we state this without proof). The functions $\psi_{n,k}$ are the *wavelets*.

[3]The first mention of wavelets appeared in an appendix to the thesis of A. Haar in 1909.

Project Activity 4. We can now write any function in V_n using the basis $\mathcal{W}_n$. As an example, the string 50, 16, 14, 28 represents a piecewise constant function which can be written as $50\phi_{2,0}+16\phi_{2,1}+14\phi_{2,2}+28\phi_{2,3}$, an element in V_2.

(a) Specifically identify the functions in $\mathcal{W}_0$, $\mathcal{W}_1$, and $\mathcal{W}_2$, and $\mathcal{W}_3$.

(b) As mentioned earlier, we can identify a signal, and each wavelet function, with a vector in $\mathbb{R}^m$ for an appropriate value of m. We can then use this identification to decompose any signal as a linear combination of wavelets. We illustrate this idea with the signal $[50\ 16\ 14\ 28]^\mathsf{T}$ in $\mathbb{R}^4$. Recall that we can represent this signal as the function $f = 50\phi_{2,0} + 16\phi_{2,1} + 14\phi_{2,2} + 28\phi_{2,3}$.

   i. Find the the vectors $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$, and $\mathbf{w}_4$ in $\mathbb{R}^m$ that are identified with $\phi_{0,0}$, $\psi_{0,0}$, $\psi_{1,0}$, and $\psi_{1,1}$, respectively.

   ii. Any linear combination $c_1\phi_{0,0} + c_2\psi_{0,0} + c_3\psi_{1,0} + c_4\psi_{1,1}$ is then identified with the linear combination $c_1\mathbf{w}_1 + c_2\mathbf{w}_2 + c_3\mathbf{w}_3 + c_4\mathbf{w}_4$. Use this idea to find the weights to write the function f as a linear combination of $\phi_{0,0}$, $\psi_{0,0}$, $\psi_{1,0}$, and $\psi_{1,1}$.

Although is it not necessarily easy to observe, the weights in the decomposition $f = 27\phi_{0,0} + 6\psi_{0,0} + 17\psi_{1,0} - 7\psi_{1,1}$ are just averages and differences of the original weights in $f = 50\phi_{2,0} + 16\phi_{2,1} + 14\phi_{2,2} + 28\phi_{2,3}$. To see how, notice that if we take the overall average of the original weights we obtain the value of 27. If we average the original weights in pairs (50 and 16, and 14 and 28) we obtain the values 33 and 21, and if we take average differences of the original weights in pairs (50 and 16, and 14 and 28) we obtain the values 17 and -7. We can treat the signal $[33\ 21]^\mathsf{T}$ formed average of the pairs of the original weights as a smaller copy of the original signal. The average difference of the entries of this new signal is 6. So the weights in our final decomposition are obtained by differences between successive averages and certain coefficients. The coefficients in our final decomposition $27\phi_{0,0} + 6\psi_{0,0} + 17\psi_{1,0} - 7\psi_{1,1}$ are called *wavelet coefficients*. This is the idea that makes wavelets so useful for image compression. In many images, pixels that are near to each other often have similar coloring or shading. These pixels are coded with numbers that are close in value. In the differencing process, these numbers are replaced with numbers that are close to 0. If there is little difference in the shading of the adjacent pixels, the image will be changed only a little if the shadings are made the same. This results in replacing these small wavelet coefficients with zeros. If the processed vectors contain long strings of zeros, the vectors can be significantly compressed.

Once we have recognized the pattern in expressing our original function as an overall average and wavelet coefficients we can perform these operations more quickly with matrices.

Project Activity 5. The process of averaging and differencing discussed in and following Project Activity 4 can be viewed as a matrix-vector problem. As we saw in Project Activity 4, we can translate the problem of finding wavelet coefficients to the matrix world.

(a) Consider again the problem of finding the wavelet coefficients contained in the vector $[27\ 6\ 17\ -7]^\mathsf{T}$ for the signal $[50\ 16\ 14\ 28]^\mathsf{T}$. Find the matrix A_4 that has the property that $A_4[50\ 16\ 14\ 28]^\mathsf{T} = [27\ 6\ 17\ -7]^\mathsf{T}$. (You have already done part of this problem in Project Activity 4.) Explain how A_4 performs the averaging and differencing discussed earlier.

(b) Repeat the process in part (a) to find the matrix A_8 that converts a signal to its wavelet coefficients.

(c) The matrix A_i is called a *forward wavelet transformation matrix* and A_i^{-1} is the *inverse wavelet transform matrix*. Use A_8 to show that the wavelet coefficients for the data string $[80\ 48\ 4\ 36\ 28\ 64\ 6\ 50]^\mathsf{T}$ are contained in the vector $[39.5\ 2.5\ 22\ 9\ 16\ -16\ -18\ -22]^\mathsf{T}$.

Now we have all of the necessary background to discuss image compression. Suppose we want to store an image. We partition the image vertically and horizontally and record the color or shade at each grid entry. The grid entries will be our pixels. This gives a matrix, M, of colors, indexed by pixels or horizontal and vertical position. To simplify our examples we will work in gray-scale, where our grid entries are integers between 0 (black) and 255 (white). We can treat each column of our grid as a piecewise constant function. As an example, the image matrix M that produced the picture at left in Figure 1 is given in (1).

We can then apply a 16 by 16 forward wavelet transformation matrix A_{16} to M to convert the columns to averages and wavelet coefficients that will appear in the matrix $A_{16}M$. These wavelet coefficients allow us to compress the image – that is, create a smaller set of data that contains the essence of the original image.

Recall that the forward wavelet transformation matrix computes weighted differences of consecutive entries in the columns of the image matrix M. If two entries in M are close in values, the weighted difference in $A_{16}M$ will be close to 0. For our example, the matrix $A_{16}M$ is approximately

$$\begin{bmatrix}
208.0 & 202.0 & 178.0 & 165.0 & 155.0 & 172.0 & 118.0 & 172.0 & 155.0 & 153.0 & 176.0 & 202.0 & 208.0 & 210.0 & 209.0 & 208.0 \\
33.4 & 24.1 & -0.625 & 0.938 & -2.50 & -5.94 & 42.8 & -5.94 & -2.50 & 12.8 & 0.938 & 24.7 & 30.6 & 33.4 & 32.5 & 31.6 \\
-1.88 & -13.8 & 19.4 & 2.50 & 0.0 & -2.50 & 8.12 & -2.50 & 0.0 & 2.50 & 19.4 & -13.8 & 1.88 & -3.75 & -1.88 & 0.0 \\
17.5 & 61.9 & 61.9 & 6.88 & 0.0 & 61.9 & 0.0 & 61.9 & 0.0 & 30.6 & 65.0 & 66.9 & 66.9 & 19.4 & 66.9 & 66.9 \\
0.0 & 27.5 & 43.8 & 16.2 & 0.0 & -11.2 & 16.2 & -11.2 & 0.0 & 16.2 & 43.8 & 27.5 & 0.0 & 0.0 & 0.0 & 0.0 \\
3.75 & 0.0 & 27.5 & -11.2 & 0.0 & -16.2 & 22.5 & -16.2 & 0.0 & -11.2 & 27.5 & 0.0 & -3.75 & -7.50 & -3.75 & 0.0 \\
47.5 & 0.0 & 0.0 & 13.8 & 82.5 & 0.0 & 0.0 & 0.0 & 82.5 & 13.8 & 0.0 & 3.75 & 3.75 & 51.2 & 3.75 & 3.75 \\
82.5 & 41.2 & 41.2 & 82.5 & 82.5 & 41.2 & 0.0 & 41.2 & 82.5 & 35.0 & 35.0 & 35.0 & 35.0 & 82.5 & 35.0 & 35.0 \\
0.0 & 0.0 & 0.0 & 55.0 & -22.5 & -22.5 & 55.0 & -22.5 & -22.5 & 55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 55.0 & -22.5 & -22.5 & 22.5 & 0.0 & -22.5 & 0.0 & 22.5 & -22.5 & -22.5 & 55.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-7.50 & 0.0 & -55.0 & 22.5 & 22.5 & -22.5 & 0.0 & -22.5 & 22.5 & 22.5 & -55.0 & 0.0 & 7.50 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 22.5 & -55.0 & 0.0 & -55.0 & 22.5 & 0.0 & 0.0 & 0.0 & -15.0 & 0.0 & -7.50 & 0.0 \\
0.0 & 0.0 & 0.0 & -55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -55.0 & 0.0 & 7.50 & 7.50 & 7.50 & 7.50 & 7.50 \\
95.0 & 0.0 & 0.0 & -82.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -82.5 & 0.0 & 0.0 & 0.0 & 95.0 & 0.0 & 0.0 \\
0.0 & -82.5 & 82.5 & 0.0 & 0.0 & -82.5 & 0.0 & -82.5 & 0.0 & 95.0 & -95.0 & 95.0 & -95.0 & 0.0 & 95.0 & -95.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}.$$

Note that there are many wavelet coefficients that are quite small compared to others – the ones where the weighted averages are close to 0. In a sense, the weighted differences tell us how much "detail" about the whole that each piece of information contains. If a piece of information contains only a small amount of information about the whole, then we shouldn't sacrifice much of the picture if we ignore the small "detail" coefficients. One way to ignore the small "detail" coefficients is to use *thresholding*.

With thresholding (this is *hard thresholding* or *keep or kill*), we decide on how much of the detail we want to remove (this is called the *tolerance*). So we set a tolerance and then replace each entry in our matrix $A_{16}M$ whose absolute value is below the tolerance with 0 to obtain a new matrix M_1. In our example, if you use a threshold value of 10 we obtain the new matrix M_1:

$$\begin{bmatrix}
208.0 & 202.0 & 178.0 & 165.0 & 155.0 & 172.0 & 118.0 & 172.0 & 155.0 & 153.0 & 176.0 & 202.0 & 208.0 & 210.0 & 209.0 & 208.0 \\
33.4 & 24.1 & 0.0 & 0.0 & 0.0 & 0.0 & 42.8 & 0.0 & 0.0 & 12.8 & 0.0 & 24.7 & 30.6 & 33.4 & 32.5 & 31.6 \\
0.0 & -13.8 & 19.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 19.4 & -13.8 & 0.0 & 0.0 & 0.0 & 0.0 \\
17.5 & 61.9 & 61.9 & 0.0 & 0.0 & 61.9 & 0.0 & 61.9 & 0.0 & 30.6 & 65.0 & 66.9 & 66.9 & 19.4 & 66.9 & 66.9 \\
0.0 & 27.5 & 43.8 & 16.2 & 0.0 & -11.2 & 16.2 & -11.2 & 0.0 & 16.2 & 43.8 & 27.5 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 27.5 & -11.2 & 0.0 & -16.2 & 22.5 & -16.2 & 0.0 & -11.2 & 27.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
47.5 & 0.0 & 0.0 & 13.8 & 82.5 & 0.0 & 0.0 & 0.0 & 82.5 & 13.8 & 0.0 & 0.0 & 0.0 & 51.2 & 0.0 & 0.0 \\
82.5 & 41.2 & 41.2 & 82.5 & 82.5 & 41.2 & 0.0 & 41.2 & 82.5 & 35.0 & 35.0 & 35.0 & 35.0 & 82.5 & 35.0 & 35.0 \\
0.0 & 0.0 & 0.0 & 55.0 & -22.5 & -22.5 & 55.0 & -22.5 & -22.5 & 55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 55.0 & -22.5 & -22.5 & 22.5 & 0.0 & -22.5 & 0.0 & 22.5 & -22.5 & -22.5 & 55.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -55.0 & 22.5 & 22.5 & -22.5 & 0.0 & -22.5 & 22.5 & 22.5 & -55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 22.5 & -55.0 & 0.0 & -55.0 & 22.5 & 0.0 & 0.0 & 0.0 & -15.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & -55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -55.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
95.0 & 0.0 & 0.0 & -82.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -82.5 & 0.0 & 0.0 & 0.0 & 95.0 & 0.0 & 0.0 \\
0.0 & -82.5 & 82.5 & 0.0 & 0.0 & -82.5 & 0.0 & -82.5 & 0.0 & 95.0 & -95.0 & 95.0 & -95.0 & 0.0 & 95.0 & -95.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}.$$

We now have introduced many zeros in our matrix. This is where we compress the image. To store the original image, we need to store every pixel. Once we introduce strings of zeros we can identify a new code (say 256) that indicates we have a string of zeros. We can then follow that code with the number of zeros in the string. So if we had a string of 15 zeros in a signal, we could store that information in 2 bytes rather than 15 and obtain significant savings in storage. This process removes some detail from our picture, but only the small detail. To convert back to an image, we just undo the forward processing by multiplying our thresholded matrix M_1 by A_{16}^{-1}. The ultimate goal is to obtain significant compression but still have $A_{16}^{-1}M_1$ retain all of the essence of the original image.

In our example using M_1, the reconstructed image matrix is $A_{16}^{-1}M_1$ (rounded to the nearest whole number) is

$$\begin{bmatrix}
242 & 240 & 241 & 237 & 132 & 138 & 232 & 138 & 132 & 238 & 239 & 240 & 238 & 244 & 242 & 240 \\
242 & 240 & 241 & 127 & 178 & 183 & 122 & 183 & 178 & 128 & 239 & 240 & 238 & 244 & 242 & 240 \\
242 & 240 & 131 & 127 & 178 & 183 & 122 & 183 & 178 & 128 & 129 & 240 & 238 & 244 & 242 & 240 \\
242 & 130 & 176 & 172 & 132 & 183 & 167 & 183 & 132 & 172 & 174 & 130 & 238 & 244 & 242 & 240 \\
242 & 240 & 131 & 177 & 178 & 133 & 183 & 133 & 178 & 178 & 129 & 240 & 238 & 244 & 242 & 240 \\
242 & 240 & 241 & 132 & 132 & 178 & 183 & 178 & 132 & 132 & 239 & 240 & 238 & 244 & 242 & 240 \\
242 & 240 & 131 & 177 & 178 & 133 & 138 & 133 & 178 & 178 & 129 & 240 & 223 & 244 & 242 & 240 \\
242 & 240 & 131 & 177 & 132 & 243 & 138 & 243 & 132 & 178 & 129 & 240 & 253 & 244 & 242 & 240 \\
240 & 240 & 239 & 124 & 238 & 234 & 75 & 234 & 238 & 130 & 241 & 244 & 244 & 248 & 244 & 244 \\
240 & 240 & 239 & 234 & 238 & 234 & 75 & 234 & 238 & 240 & 241 & 244 & 244 & 248 & 244 & 244 \\
240 & 240 & 239 & 69 & 73 & 234 & 75 & 234 & 73 & 75 & 241 & 244 & 244 & 240 & 244 & 244 \\
50 & 240 & 239 & 234 & 73 & 234 & 75 & 234 & 73 & 240 & 241 & 244 & 244 & 50 & 244 & 244 \\
240 & 75 & 239 & 248 & 238 & 69 & 75 & 69 & 238 & 240 & 51 & 240 & 50 & 240 & 240 & 50 \\
240 & 240 & 74 & 248 & 238 & 234 & 75 & 234 & 238 & 50 & 241 & 50 & 240 & 240 & 50 & 240 \\
75 & 75 & 74 & 83 & 73 & 69 & 75 & 69 & 73 & 75 & 76 & 75 & 75 & 75 & 75 & 75 \\
75 & 75 & 74 & 83 & 73 & 69 & 75 & 69 & 73 & 75 & 76 & 75 & 75 & 75 & 75 & 75
\end{bmatrix}.$$

We convert this into a gray-scale image and obtain the image at right in Figure 1. Compare this image to the original at right in Figure 1. It is difficult to tell the difference.

There is a Sage file you can use at `http://faculty.gvsu.edu/schlicks/Wavelets_Sage.html` that allows you to create your own 16 by 16 image and process, process your image with the Haar wavelets in $\mathbb{R}^{16}$, apply thresholding, and reconstruct the compressed image. matrix. You can create your own image, experiment with several different threshold levels, and choose the one that you feel gives the best combination of strings of 0s while reproducing a reasonable copy of the original image.

Wavelets Sage cells

This page contains a series of Sage cells that can be used to create a 16 by 16 grayscale image, process the image with wavelets, apply hard thresholding, and construct compressed images. Provided this page is not reloaded, results from one cell may be used in another. A cell can be evaluated using the **Evaluate (Sage)** button or by pressing Shift-Enter in a cell. In the first cell, enter the grayscale levels for your image in a 16 by 16 matrix A. Executing this cell will enter the matrix A into memory and print it to the screen. Enter the matrix as a list of vectors as indicated.

```
A=matrix(QQ,[[240,240,240,240,130,130,240,130,130,240,240,240,240,240,240,240],
[240,240,240,130,175,175,130,175,175,130,240,240,240,240,240,240],
[240,240,130,130,175,175,130,175,175,130,130,240,240,240,240,240],
[240,130,175,175,130,175,175,175,130,175,175,130,240,240,240,240],
[240,240,130,175,175,130,175,130,175,175,130,240,240,240,240,240],
[255,240,240,130,130,175,175,175,130,130,240,240,225,240,240,240],
[240,240,130,175,175,130,130,130,175,175,130,240,225,255,240,240],
[240,240,130,175,130,240,130,240,130,175,130,240,255,255,255,240],
[240,240,240,130,240,240,75,240,240,130,240,255,255,255,255,255],
[240,240,240,240,240,240,75,240,240,240,240,240,240,240,240,240],
[240,240,240,75,75,240,75,240,75,75,240,240,240,240,240,240],
[50,240,240,240,75,240,75,240,75,240,240,240,240,50,240,240],
```

Evaluate (Sage)

```
[240 240 240 240 130 130 240 130 130 240 240 240 240 240 240 240]
[240 240 240 130 175 175 130 175 175 130 240 240 240 240 240 240]
[240 240 130 130 175 175 130 175 175 130 130 240 240 240 240 240]
[240 130 175 175 130 175 175 175 130 175 175 130 240 240 240 240]
[240 240 130 175 175 130 175 130 175 175 130 240 240 240 240 240]
[255 240 240 130 130 175 175 175 130 130 240 240 225 240 240 240]
[240 240 130 175 175 130 130 130 175 175 130 240 225 255 240 240]
[240 240 130 175 130 240 130 240 130 175 130 240 255 255 255 240]
[240 240 240 130 240 240  75 240 240 130 240 255 255 255 255 255]
[240 240 240 240 240 240  75 240 240 240 240 240 240 240 240 240]
[240 240 240  75  75 240  75 240  75  75 240 240 240 240 240 240]
[ 50 240 240 240  75 240  75 240  75 240 240 240 240  50 240 240]
[240  75 240 240 240  75  75  75 240 240  50 240  50 240 240  50]
[240 240  75 240 240 240  75 240 240  50 240  50 240 240  50 240]
[ 75  75  75  75  75  75  75  75  75  75  75  75  75  75  75  75]
[ 75  75  75  75  75  75  75  75  75  75  75  75  75  75  75  75]
```

Help | Powered by SageMath

The next cell creates the 16 by 16 grayscale image defined by your matrix A. The name given to the image is g, and you can save this image as an eps (or png, svg, pdf) file with the command g.save('filename.eps'), where filename is whatever name you want to assign to the file. (Use a png, pdf, svg extension to save to other formats.) There is a blank Sage input line at the end of this file that you can use for this purpose.

```
g=Graphics()
for i in range(15):
    for j in range(15):
        p=polygon2d([[i,16-j], [i+1,16-j], [i+1,16-j-1], [i,16-j-1]],
            rgbcolor=(A[j][i]/255,A[j][i]/255,A[j][i]/255), axes=false)
        g=g+p
```

Evaluate (Sage)

Help | Powered by SageMath

The next cell creates the 16 by 16 wavelet matrix and its inverse for computational purposes, then applies wavelets to produce the matrix of wavelet coefficients. This matrix is the output that you see.

```
WM = matrix(QQ,[[1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, -1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
[1, 1, -1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[1, 1, -1, 0, 0, 1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0],
[1, 1, -1, 0, 0, -1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[1, 1, -1, 0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0],
[1, -1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[1, -1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, -1, 0, 0, 0],
[1, -1, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[1, -1, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0],
[1, -1, 0, -1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
```

Evaluate (Sage)

```
[  208.44   202.19   177.50   165.31   155.00   172.19   117.81   172.19   155.00   153.44   175.94   201.56   207.50   210.31   209.38   208.44]
[  33.438   24.062 -0.62500  0.93750  -2.5000  -5.9375   42.812  -5.9375  -2.5000   12.812  0.93750   24.688   30.625   33.438   32.500   31.562]
[ -1.8750  -13.750   19.375   2.5000  0.00000  -2.5000   8.1250  -2.5000  0.00000   2.5000   19.375  -13.750   1.8750  -3.7500  -1.8750  0.00000]
[  17.500   61.875   61.875   6.8750  0.00000   61.875  0.00000   61.875  0.00000   30.625   65.000   66.875   66.875   19.375   66.875   66.875]
[ 0.00000   27.500   43.750   16.250  0.00000  -11.250   16.250  -11.250  0.00000   16.250   43.750   27.500  0.00000  0.00000  0.00000  0.00000]
[  3.7500  0.00000   27.500  -11.250  0.00000  -16.250   22.500  -16.250  0.00000  -11.250   27.500  0.00000  -3.7500  -7.5000  -3.7500  0.00000]
[  47.500  0.00000  0.00000   13.750   82.500  0.00000  0.00000  0.00000   82.500   13.750  0.00000   3.7500   3.7500   51.250   3.7500   3.7500]
[  82.500   41.250   41.250   82.500   82.500   41.250  0.00000   41.250   82.500   35.000   35.000   35.000   35.000   82.500   35.000   35.000]
[ 0.00000  0.00000  0.00000   55.000  -22.500  -22.500   55.000  -22.500  -22.500   55.000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000]
[ 0.00000   55.000  -22.500  -22.500   22.500  0.00000  -22.500  0.00000   22.500  -22.500  -22.500   55.000  0.00000  0.00000  0.00000  0.00000]
[ -7.5000  0.00000  -55.000   22.500   22.500  -22.500  0.00000  -22.500   22.500   22.500  -55.000  0.00000   7.5000  0.00000  0.00000  0.00000]
[ 0.00000  0.00000  0.00000  0.00000   22.500  -55.000  0.00000  -55.000   22.500  0.00000  0.00000  0.00000  -15.000  0.00000  -7.5000  0.00000]
[ 0.00000  0.00000  0.00000  -55.000  0.00000  0.00000  0.00000  0.00000  0.00000  -55.000  0.00000   7.5000   7.5000   7.5000   7.5000   7.5000]
[  95.000  0.00000  0.00000  -82.500  0.00000  0.00000  0.00000  0.00000  0.00000  -82.500  0.00000  0.00000  0.00000   95.000  0.00000  0.00000]
[ 0.00000  -82.500   82.500  0.00000  0.00000  -82.500  0.00000  -82.500  0.00000   95.000  -95.000   95.000  -95.000  0.00000   95.000  -95.000]
[ 0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000]
```

Help | Powered by SageMath

The next cell inputs the threshold level (as the variable `thresh'). The default is set at 10. This is the one cell where you should feel free to change the value and re-execute several times to test different threshold levels. The cells after this one will produce the thresholded matrix for you to view and the compressed image.

```
thresh=10
print(thresh)
```

Evaluate (Sage)

```
10
```

Help | Powered by SageMath

The following cell calculates and prints the thresholded matrix using the threshold value of thresh for you to view.

```
TestPA=copy(PA)
for i in range(15):
    for j in range(15):
        if abs(PA[i,j]) <= thresh:
            TestPA[i,j]=0
```

Evaluate (Sage)

```
[ 208.44  202.19  177.50  165.31  155.00  172.19  117.81  172.19  155.00  153.44  175.94  201.56  207.50  210.31  209.38  208.44]
[ 33.438  24.062 0.00000 0.00000 0.00000 0.00000  42.812 0.00000 0.00000  12.812 0.00000  24.688  30.625  33.438  32.500  31.562]
[0.00000 -13.750  19.375 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000  19.375 -13.750 0.00000 0.00000 0.00000 0.00000]
[ 17.500  61.875  61.875 0.00000 0.00000  61.875 0.00000  61.875 0.00000  30.625  65.000  66.875  66.875  19.375  66.875  66.875]
[0.00000  27.500  43.750  16.250 0.00000 -11.250  16.250 -11.250 0.00000  16.250  43.750  27.500 0.00000 0.00000 0.00000 0.00000]
[0.00000 0.00000  27.500 -11.250 0.00000 -16.250  22.500 -16.250 0.00000 -11.250  27.500 0.00000 0.00000 0.00000 0.00000 0.00000]
[ 47.500 0.00000 0.00000  13.750  82.500 0.00000 0.00000 0.00000  82.500  13.750 0.00000 0.00000 0.00000  51.250 0.00000  3.7500]
[ 82.500  41.250  41.250  82.500  82.500  41.250 0.00000  41.250  82.500  35.000  35.000  35.000  35.000  82.500  35.000  35.000]
[0.00000 0.00000 0.00000  55.000 -22.500 -22.500  55.000 -22.500 -22.500  55.000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000]
[0.00000  55.000 -22.500 -22.500  22.500 0.00000 -22.500 0.00000  22.500 -22.500 -22.500  55.000 0.00000 0.00000 0.00000 0.00000]
[0.00000 0.00000 -55.000  22.500  22.500 -22.500 0.00000 -22.500  22.500  22.500 -55.000 0.00000 0.00000 0.00000 0.00000 0.00000]
[0.00000 0.00000 0.00000 0.00000  22.500 -55.000 0.00000 -55.000  22.500 0.00000 0.00000 0.00000 -15.000 0.00000 0.00000 0.00000]
[0.00000 0.00000 0.00000 -55.000 0.00000 0.00000 0.00000 0.00000 0.00000 -55.000 0.00000 0.00000 0.00000 0.00000 0.00000  7.5000]
[ 95.000 0.00000 0.00000 -82.500 0.00000 0.00000 0.00000 0.00000 0.00000 -82.500 0.00000 0.00000 0.00000  95.000 0.00000 0.00000]
[0.00000 -82.500  82.500 0.00000 0.00000 -82.500 0.00000 -82.500 0.00000  95.000 -95.000  95.000 -95.000 0.00000  95.000 -95.000]
[0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000]
```

Help | Powered by SageMath

Finally, executing this last cell shows the compressed image (saved as h, you can save a copy of h as a graphics file as directed earlier with g.)

```
newA=WM*TestPA
h=Graphics()
for i in range(15):
    for j in range(15):
        p=polygon2d([[i,16-j], [i+1,16-j], [i+1,16-j-1], [i,16-j-1]],
            rgbcolor=(newA[j][i]/255,newA[j][i]/255,newA[j][i]/255), axes=false)
        h=h+p
```

Evaluate (Sage)

Help | Powered by SageMath

Use this cell to save image files or for whatever other commands you might want to implement. If you save a file as, say, g.save('figure.eps'), you should see an output of figure.eps (perhaps followed by an Updated 0 time(s)] message). Click on flower.eps to see the image you saved.

```
1
```

Evaluate (Sage)

Examples of Applications

- The Principal Axis Theorem: The Tennis Racket Theorem

Application: The Tennis Racket Effect

Try an experiment with a tennis racket (or a squash racket, or a ping pong paddle). Assume that the origin is at the center of the racket. Let $\mathbf{u}_1$ be the vector from the origin to the handle, $\mathbf{u}_2$ a vector in the plane of the head perpendicular to $\mathbf{u}_1$ as in Figure 1. Let $\mathbf{u}_3$ be a vector through the origin perpendicular to the plane of the head of the racket. Hold the racket by the handle and spin it to make one rotation around the $\mathbf{u}_1$ axis. This is pretty easy. Now toss the racket into the air to make one complete rotation around the axis of the vector $\mathbf{u}_2$ and catch the handle. Repeat this several times. You should notice that in most instances, the racket will also have made a half rotation around the $\mathbf{u}_1$ axis so that the other face of the racket now points up. It is not difficult to throw the racket so that it rotates around the $\mathbf{u}_3$ axis without the added half rotation we see around the $\mathbf{u}_2$ axis. A good video that illustrates this behavior can be seen at https://www.youtube.com/watch?v=4dqCQqI-Gis.

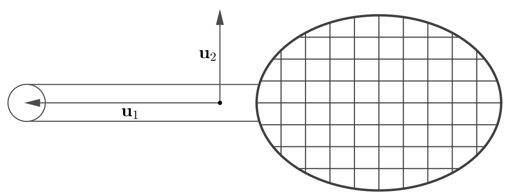

Figure 1: Two principal axes of a tennis racket.

This effect is a result in classical mechanics that describes the rotational movement of a rigid body in space, called the *tennis racket effect* (or the Dzhanibekov effect, after the Russian cosmonaut Vladimir Dzhanibekov who discovered the theorem's consequences while in zero gravity in space – you can see an illustration of this in the video at https://www.youtube.com/watch?v=L2o9eBl_Gzw). The result is simple to see in practice, but is difficult to intuitively understand why the behavior is different around the intermediate axis. There is a story of a student who asked the famous physicist Richard Feynman if there is any intuitive way to understand the result; Feynman supposedly went into deep thought for about 10 or 15 seconds and answered, "no." As we will see later in this section, we can understand this effect using the principal axes of a rigid body.

Project: The Tennis Racket Theorem

If a particle of mass m and velocity v is moving in a straight line, its kinetic energy KE is given by $KE = \frac{1}{2}mv^2$. If, instead, the particle rotates around an axis with angular velocity ω (in radians per unit of time), its linear velocity is $v = r\omega$, where r is the radius of the particle's circular path. Substituting into the kinetic energy formula shows that the kinetic energy of the rotating particle is then $KE = \frac{1}{2}\left(mr^2\right)\omega^2$. The quantity rm^2 is called the *moment of inertia* of the particle and is denoted by I. So $KE = \frac{1}{2}I\omega^2$ for a rotating particle. Notice that the larger the value of r, the larger the inertia. You can imagine this with a figure skater. When a skater spins along their major axis with their arms outstretched, the speed at which they rotate is lower than when they bring their arms into their bodies. The moment of inertia for rotational motion plays a role similar to the mass in linear motion. Essentially, the inertia tells us how resistant the particle is to rotation.

To understand the tennis racket effect, we are interested in rigid bodies as they move through space. Any rigid body in three space has three principal axes about which it likes to spin. These axes are at right angles to each other and pass through the center of mass. Think of enclosing the object in an ellipsoid – the longest axis is the *primary* axis, the middle axis is the *intermediate* axis, and the third axis is the third axis. As a rigid body moves through space, it rotates around these axes and there is inertia along each axis. Just like with a tennis racket, if you were to imagine an axle along any of the principal axes and spin the object along that axel, it will either rotate happily with no odd behavior like flipping, or it won't. The former behavior is that of a stable axis and the latter an unstable axis. The Tennis Racket Theorem is a statement about the rotation of the body. Essentially, the Tennis Racket Theorem states that the rotation of a rigid object around its primary and third principal axes is stable, while rotation around its intermediate axis is not. To understand why this is so, we need to return to moments of inertia.

Assume that we have a rigid body moving through space. Assume that I_1, I_2, and I_3 are the moments of inertia around the primary, intermediate, and third principal axes with $I_1 > I_2 > I_3$. Also assume that ω_1, ω_2, and ω_3 are the components of the angular velocity along each axis. Euler's equation tell us that

$$I_1\dot{\omega}_1 = (I_2 - I_3)\omega_2\omega_3 \tag{1}$$
$$I_2\dot{\omega}_2 = (I_3 - I_1)\omega_3\omega_1 \tag{2}$$
$$I_3\dot{\omega}_3 = (I_1 - I_2)\omega_1\omega_2. \tag{3}$$

(The dots indicate a derivative with respect to time, which is common notation in physics.) We will use Euler's equations to understand the Tennis Racket Theorem.

Project Activity 1. To start, we consider rotation around the first principal axis. Our goal is to show that rotation around this axis is stable. That is, small perturbations angular velocity will have only small effects on the rotation of the object. So we assume that ω_2 and ω_3 are small. In general, the product of two small quantities will be much smaller, so (1) implies that $\dot{\omega}_1$ must be very small. So we can disregard $\dot{\omega}_1$ in our calculations.

(a) Differentiate (2) with respect to time to explain why

$$I_2\ddot{\omega}_2 \approx (I_3 - I_1)\dot{\omega}_3\omega_1.$$

(b) Substitute for $\dot{\omega}_3$ from (3) to show that ω_2 is an approximate solution to

$$\ddot{\omega}_2 = -k\omega_2 \tag{4}$$

for some positive constant k.

(c) The equation (4) is a differential equation because it is an equation that involves derivatives of a function. Show by differentiating twice that, if

$$\omega_2 = A\cos\left(\sqrt{k}t + B\right) \tag{5}$$

(where A and B are any scalars), then ω_2 is a solution to (4). (In fact, $\omega_2 = A\cos\left(\sqrt{k}t + B\right)$ is the general solution to (4), which is verified in just about any course in differential equations.)

Equation 5 shows that ω_2 is is bounded, so that any slight perturbations in angular velocity have a limited effect on ω_2. A similar argument can be made for ω_3. This implies that the rotation around the principal axes is stable – slight changes in angular velocity have limited effects on the rotations around the other axes.

We can make a similar argument for rotation around the third principal axes.

Project Activity 2. In this activity, repeat the process from Project Activity to show that rotation around the third principal axis is stable. So assume that ω_1 and ω_3 are small, which implies by (3) implies that $\dot{\omega}_3$ must be very small and can be disregarded in calculations.

Now the issue is why is rotation around the second principal axis different.

Project Activity 3. Now assume that ω_1 and ω_3 are small. Thus, $\dot{\omega}_2$ is very small by (2), and we consider $\dot{\omega}_2$ to be negligible.

(a) Differentiate (1) to show that

$$I_1\ddot{\omega}_1 \approx (I_2 - I_3)\omega_2\dot{\omega}_3.$$

(b) Substitute for $\dot{\omega}_3$ from (3) to show that ω_1 is an approximate solution to

$$\ddot{\omega}_1 = k\omega_1 \tag{6}$$

for some positive scalar k.

(c) The fact that the constant multiplier in (6) is positive instead of negative as in (4) completely changes the type of solution. Show that

$$\omega_1 = Ae^{\sqrt{k}t+B} \tag{7}$$

(where A and B are any scalars) is a solution to (6) (and, in fact, is the general solution). Explain why this shows that rotation around the second principal axis is not stable.

Linear Algebra and Applications

Section Title	Application
1. Introduction to Systems of Linear Equations	Electrical Circuits and the Wheatstone Bridge
2. The Matrix Representation of a Linear System	A Polynomial Fitting Application: Simpson's Rule
3. Row Echelon Forms	Modeling a Chemical Reaction
4. Vector Representation	Analyzing Knight Moves (G)
5. The Matrix-Vector Form of a Linear System	An Input-Output Model in Economics
6. Linear Dependence and Independence	Generating Bézier Curves (G)
7. Matrix Transformations	The Geometry of Matrix Transformations (G)
8. Matrix Operations	Strassen's Algorithm
9. Introduction to Eigenvalues and Eigenvectors	The Google PageRank Algorithm (G)
10. The Inverse of a Matrix	The Richardson Arms Race Model

Linear Algebra and Applications

Section Title	Application
11. The Invertible Matrix Theorem	None
12. The Structure of $\mathbb{R}^n$	Connecting GDP and Consumption in Economics
13. The Null Space and Column Space of a Matrix	Solving the Lights Out Game (G)
14. Eigenspaces of a Matrix	Modeling Population Migration
15. Bases and Dimension	Lattice Based Cryptography
16. The Determinant	Area and Volume using Determinants
17. The Characteristic Equation	The Ehrenfest Model of Second Law of Thermodynamics
18. Diagonalization	Binet's Formula for the Fibonacci Numbers
19. Approximating Eigenvalues and Eigenvectors	Leslie Matrices and Population Modeling (G)
20. Complex Eigenvalues	The Gershgorin Disk Theorem

Linear Algebra and Applications

Section Title	Application
21. Properties of Determinants	None
22. Vector Spaces	Hamming Codes and the Hat Puzzle
23. Bases for Vector Spaces	Image Compression with Wavelets (S)
24. The Dimension of a Vector Space	Principal Component Analysis
25. Coordinate Vectors and Coordinate Transformations	Finding Formulas for Sums of Powers
26. Change of Basis	Describing Orbits of Planets
27. The Dot Product in $\mathbb{R}^n$	Back-Face Culling
28. Orthogonal and Orthonormal Bases in $\mathbb{R}^n$	Rotations in 3-Space (G)
29. Inner Products	Fourier Series and Musical Tones

Linear Algebra and Applications

Section Title	Application
30. The Gram-Schmidt Process	Gaussian Quadrature and Legendre Polynomials
31. Orthogonal Diagonalization	The Multivariable Second Derivative Test
32. Quadratic Forms and the Principal Axis Theorem	The Tennis Racket Effect
33. The Singular Value Decomposition	Latent Semantic Indexing
34. Applications of the Singular Value Decomposition	The Global Positioning System
35. Linear Transformations	Fractals and Iterated Function Systems (S)
36. The Matrix of a Linear Transformation	Shamir's Secret Sharing and Lagrange Polynomials
37. Eigenvalues of Linear Transformations	Second Order Linear Differential Equations

End

Linear Algebra and Applications: An Inquiry-Based Approach,
Feryal Alayont and Steven Schlicker

Contact information

alayontf@gvsu.edu
schlicks@gvsu.edu

End

Linear Algebra and Applications: An Inquiry-Based Approach,
Feryal Alayont and Steven Schlicker

Contact information

alayontf@gvsu.edu
schlicks@gvsu.edu

Thank you for listening!