

LINEAR ALGEBRA IN DIGITAL AUDIO

COPYRIGHT © 2003 BY RANDY A. FREEMAN

ABSTRACT. Basic linear algebra concepts such as linear combination, span, linear independence, basis, dimension, rank, and orthogonality are explained in the context of digital audio. Simple MATLAB functions are written to compute the spectrum and the spectrogram of a digital audio signal.

1. AUDIO SIGNALS, SAMPLING, AND RECONSTRUCTION

One can think of an audio signal as a function $p(t)$ representing the variation of air pressure p versus time t . In digital audio systems such as compact disc (CD) systems, audio signals are *sampled* to produce sequences of numbers which can then be stored, processed, transmitted, and received using digital technology. For example, we can sample our audio signal $p(t)$ every T_s seconds to produce a sequence of numbers $\{s_0, s_1, s_2, \dots\}$ as follows:

$$\begin{aligned} s_0 &= p(0) \\ s_1 &= p(T_s) \\ s_2 &= p(2T_s) \\ s_3 &= p(3T_s) \\ &\vdots \\ s_k &= p(kT_s) \\ &\vdots \end{aligned}$$

The number T_s is known as the *sample period*, and its reciprocal $f_s = 1/T_s$ is known as the *sample rate*. The units of f_s are Hertz, abbreviated as Hz, where 1 Hz equals one sample per second (we also use the standard metric prefixes such as *kilo* with 1 kHz = 1000 Hz). We can visualize the sampling process using Figure 1. The top graph of this figure represents the first 10 milliseconds of Handel's *Hallelujah Chorus*, plotted as air pressure versus time. If we sample this audio signal at the rate $f_s=8192$ Hz, we obtain the sequence of numbers $\{0.000, -0.006, -0.075, -0.031, 0.006, \dots\}$ which we represent as a sequence of dots in the bottom graph of this figure. This sequence of numbers is the *digital audio signal* which we can store on a CD or inside a computer.¹ When we play a CD on a stereo system, the sequence of numbers is converted back into a pressure signal which travels from the speakers through the air to our ears. This conversion process is called *reconstruction* and is accomplished by “connecting the dots” in the bottom part of Figure 1. A fundamental issue in digital audio is whether or not the reconstructed pressure signal sounds the same to human ears as the original signal (*e.g.* the one shown in the top part of Figure 1). It turns out that if the sample rate f_s is large enough, reconstruction can indeed adequately reproduce the original signal over the range of frequencies audible to humans. In CD audio, the value of f_s is 44.1 kHz which means there are 44100 numbers in the sequence for every second of music.² This sample rate is high enough that most

This document was last modified on 9/26/2003.

¹A real digital audio signal is also *quantized* in value, that is, each number in the sequence is rounded to the nearest member of a finite set of possible signal values. For example, in CD audio, each number in the sequence is represented by 16 bits, which means it is rounded to the nearest member of a set of $2^{16} = 65536$ possible signal values. We will ignore quantization effects throughout this study and assume that each number in the sequence can take on any possible value (including irrational values such as π).

²There are actually two of these sequences in a stereo recording, one for the left channel and one for the right, for a total of 88200 numbers per second of music. In this study we will consider only a single channel.

listeners cannot discern the effects of sampling in a CD audio system; nevertheless, the use of higher sample rates is becoming common as the field of digital audio evolves beyond the CD standard.³

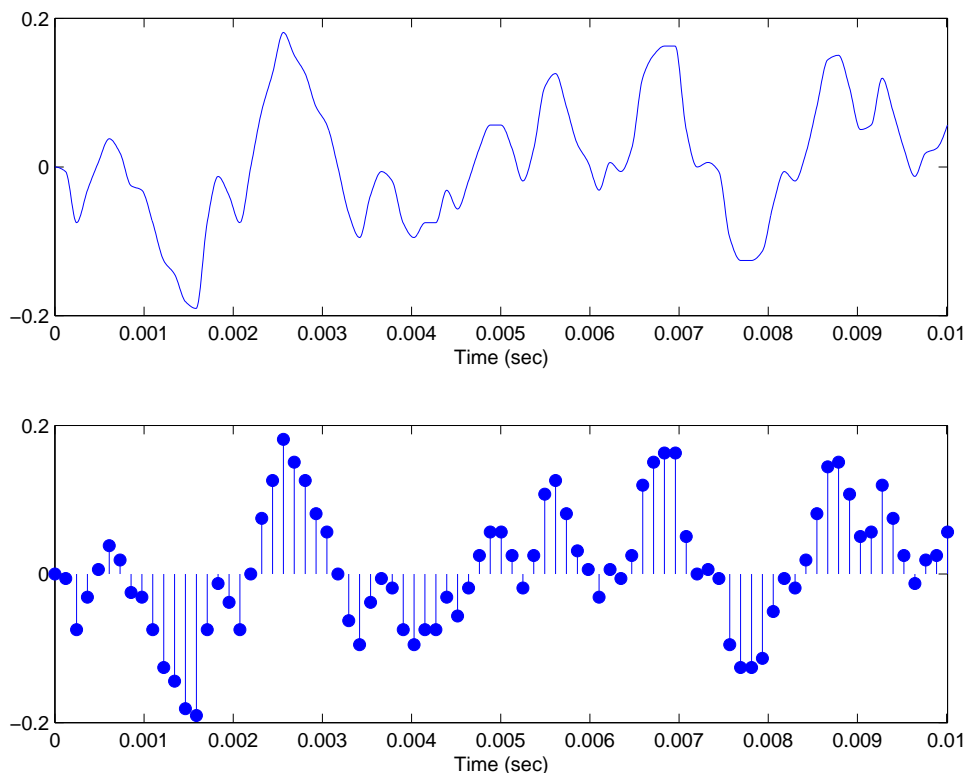


FIGURE 1. The sampling process.

2. DIGITAL AUDIO SIGNALS AS VECTORS

Because a digital audio signal of finite duration is just a finite sequence of numbers, we can represent such a signal as an element of the vector space \mathbb{R}^N , where N is the number of samples in the signal. In other words, given a sample rate f_s and an integer $N \geq 1$, we may interpret every vector $s \in \mathbb{R}^N$ as a N/f_s -second digital audio signal. Thus a 10-second excerpt of a single channel of a CD audio track would contain 441000 samples, and we may interpret every such 10-second excerpt as an element of \mathbb{R}^{441000} . Let us now interpret basic vector operations in this digital audio context.

If we multiply a digital audio signal $s \in \mathbb{R}^N$ by a scalar $c \geq 0$, the resulting vector cs is the same audio signal at a different volume: if $c > 1$ then the volume is louder relative to s , and if $c < 1$ then the volume is softer relative to s . The zero vector represents absolute silence. How do we interpret cs when c is negative? If you have set up a stereo system before, then you know that there are two wires running from the power amplifier to each speaker. If you reverse the connection of these two wires at the speaker, you essentially replace the signal s going to the speaker with its polar opposite $-s$. In theory, if you were to send the same signal to two different speakers located at the same spot but with opposite polarity connections, the sounds from each speaker would cancel each other out and you would hear nothing.⁴

We may interpret the linear combination $c_1s_1 + c_2s_2$ of two digital audio signals s_1 and s_2 as the result of playing signal s_1 (with volume adjusted by c_1) and signal s_2 (with volume adjusted by c_2) simultaneously. In fact, an important device found in any recording studio is a *mixer*, a device which takes a collection of input signals, adjusts their volumes independently, and produces their combined output. We can thus consider a mixer as a device which produces linear combinations: for example, a 3-channel mixer will take

³For example, many DVD video discs use $f_s=96$ kHz with 24 bits of resolution for “better-than-CD-quality” audio.

⁴This is the main idea behind some active noise cancellation methods.



FIGURE 2. A digital mixer.

three signals s_1, s_2, s_3 , adjust their volumes by amounts c_1, c_2, c_3 , and produce their linear combination $c_1s_1 + c_2s_2 + c_3s_3$. Suppose signals $s_1, s_2, s_3 \in \mathbb{R}^{441000}$ represent three different 10-second excerpts from a CD containing performances of works by Mozart. If we send these signals through a 3-channel mixer before sending them to our speaker, and if the mixer volume settings are at values c_1, c_2, c_3 (respectively), then what we will hear is the linear combination $c_1s_1 + c_2s_2 + c_3s_3$. The *span* of these three vectors is just the set of all possible 10-second signals we could hear by making adjustments of the volume levels on our 3-channel mixer. These three vectors are *linearly independent* if the only way to hear absolute silence at the output of the mixer is if all three channels are muted, namely, if $c_1 = c_2 = c_3 = 0$. Suppose we now randomly choose a fourth 10-second excerpt $s_4 \in \mathbb{R}^{441000}$ from this Mozart CD. Chances are that s_4 will *not* belong to $\text{span}\{s_1, s_2, s_3\}$, namely, that there will be no way to choose the mixer volume settings c_1, c_2, c_3 so that the output of the mixer, which is the linear combination $c_1s_1 + c_2s_2 + c_3s_3$, is exactly the same as the signal s_4 .

Next imagine that we have a 441000-channel mixer. There are thousands of Mozart CD's on the market, so it is feasible to obtain 441000 different 10-second excerpts $s_1, s_2, \dots, s_{441000}$ from this collection of CD's. We imagine that these excerpts are inputs to our mixer, and that the mixer has volume settings $c_1, c_2, \dots, c_{441000}$ (one for each channel). If these excerpts are chosen at random from our collection of Mozart CD's, then chances are that the collection $s_1, s_2, \dots, s_{441000}$ will be a linearly independent collection. In this case, the Basis Theorem [Lay, Section 2.9, Theorem 15] tells us that this collection of excerpts forms a *basis* for \mathbb{R}^{441000} , namely, that *every* vector $s \in \mathbb{R}^{441000}$ can be written uniquely as a linear combination of the vectors $s_1, s_2, \dots, s_{441000}$. In other words, given any 10-second excerpt s of music from any CD (jazz, pop, blues, or whatever), there is a unique way to adjust the volume settings on our mixer so that the mixer output is exactly the signal s . These unique mixer volume settings $c_1, c_2, \dots, c_{441000}$ are called the *coordinates* of the excerpt s with respect to our Mozart basis. One could therefore argue that the first 10 seconds of any track on the latest popular CD is not “new” music—it is just the result of simultaneously playing these 441000 excerpts of Mozart through our mixer with specific choices for the different volume settings. Keep in mind, however, that to find these volume settings (Mozart coordinates) for a given excerpt s , you would have to solve the system of equations $Ac = s$ for the coordinate vector $c \in \mathbb{R}^{441000}$, where A is the square matrix whose columns are the Mozart excerpts $A = [s_1 \ s_2 \ \dots \ s_{441000}]$. This is a system of 441000 equations with the same number of unknowns and may thus be difficult to solve in MATLAB; just storing the matrix A as a MATLAB variable would require over 300 GB of memory (assuming 2 bytes of memory per matrix entry).

Of course, nobody would actually want to compute Mozart coordinates for a given 10-second excerpt of CD music, but the principle behind this discussion is important for many digital audio applications, especially if we use *pure tones* as our basis vectors rather than the Mozart excerpts. The use of pure tones as basis vectors is the subject of the next sections.

3. PURE TONES AND ALIASING

A pure tone is a sinusoid pressure signal of the form $p(t) = A \cos(2\pi ft - \phi)$, where A is the *amplitude* of the tone (which is related to the volume or loudness of the tone), f is the *frequency* of the tone (which is related to the pitch of the tone), and ϕ is the *phase* of the tone. Note that the period of this sinusoid signal

is just the reciprocal of the frequency f (where f has units of Hz = cycles per second). If we sample a pure tone using a sample rate f_s , we obtain the sequence $\{s_0, s_1, s_2, \dots\}$ where $s_k = A \cos(2\pi f k / f_s - \phi)$. Using the sum formula for cosines, we note that

$$\begin{aligned} A \cos(2\pi f k / f_s - \phi) &= A \cos(\phi) \cos(2\pi f k / f_s) + A \sin(\phi) \sin(2\pi f k / f_s) \\ &= x \cos(2\pi f k / f_s) + y \sin(2\pi f k / f_s) \end{aligned} \quad (1)$$

where the scalar parameters $x = A \cos(\phi)$ and $y = A \sin(\phi)$ are independent of the time variable k . Therefore every sampled pure tone $s_k = A \cos(2\pi f k / f_s - \phi)$ can be written as the linear combination of the cosine tone $\cos(2\pi f k / f_s)$ and the sine tone $\sin(2\pi f k / f_s)$. If we regard the amplitude A and the phase ϕ as the polar coordinates of a point on the plane \mathbb{R}^2 , then the weights x and y of this linear combination are simply the Cartesian (rectangular) coordinates of this point.

We can use MATLAB to listen to pure tones through the computer speakers. For example, suppose we wish to hear the cosine tone $\cos(2\pi f k / f_s)$ having frequency $f=440$ Hz and sample rate $f_s=8$ kHz (one of many standard rates in digital audio). Two seconds of this tone requires $N=16000$ samples, which we can generate with the following MATLAB code:

```
fs = 8000;           % sample rate in Hz
N = 2*fs;           % N for a 2-second tone
k = 0:(N-1);        % time vector
f = 440;            % 440 Hz is the frequency of A above middle C
s = cos(2*pi*f*k/fs);
```

The vector \mathbf{s} should now be a row vector with 16000 entries representing the samples of the sinusoid over the 2-second interval. We can send this signal to the speaker on our computer by typing `sound(s,fs)` in MATLAB. Try this and see what happens; you should hear a 2-second tone at 440 Hz (A above middle C on a piano). If you get the message, 'Audio capabilities are not available on this machine.', then you will have to figure out how to play sound on your computer. For example, I get this message on the machine in my office which runs the Linux operating system, so to send \mathbf{s} to the speaker I must enter the following command in MATLAB:

```
wavwrite(s,fs,'tmp.wav'); !play tmp.wav; rm tmp.wav
```

You may wish to type `help audio` in MATLAB for more information. Note that the amplitude of the sinusoid corresponds to volume: entering the command `sound(s/5,fs)` (which decreases the amplitude of \mathbf{s} by a factor of 5) should result in the same tone at a lower volume. Also, keep in mind that the values of the entries in a signal \mathbf{s} sent to the audio device should always be between -1 and 1 ; numbers outside this range will be "clipped" and cause distortion (try `sound(s*20,fs)` for a distorted tone).

Once you have the sound working on your computer, change the frequency \mathbf{f} to $\mathbf{f}=1000$, recompute \mathbf{s} , send \mathbf{s} to the speakers, and note that this change affects the perceived pitch of the tone. Try the following values for \mathbf{f} : 1000, 2000, 3000, 3500, 4000, 4500, 5000, 6000. You should notice that the perceived pitch increases as \mathbf{f} increases up until the frequency 4 kHz, at which point the pitch starts to *decrease* as \mathbf{f} gets larger. In particular, the 4.5 kHz tone should sound exactly the same as the 3.5 kHz tone, the 5 kHz tone should sound like the 3 kHz tone, and so on. This phenomenon is called *aliasing*: because the cosine function is periodic with period 2π , the sampled cosine tones $\cos(2\pi f k / f_s)$ and $\cos(2\pi(f_s \pm f)k / f_s)$ give exactly the same sequence in \mathbb{R}^N for any frequency f . If we would increase \mathbf{f} beyond $\mathbf{f}=8000$, the perceived pitch would start to increase again until $\mathbf{f}=12000$, at which point the pitch will again start to decrease, and so on. Because of aliasing, a digital audio signal can only contain useful frequencies of up to half the sample rate f_s (any higher frequencies are aliased to, and perceived as, lower ones). This means a CD audio signal only contains useful frequencies of up to 22.05 kHz, but fortunately frequencies higher than this are beyond normal human perception.

4. SPECTRAL ANALYSIS

By just looking at a plot of the pressure signal $p(t)$, it may be difficult to anticipate what this signal would sound like if it reached our ears (try looking again at Figure 1). However, if $p(t)$ is the linear combination of pure tones, then knowledge of the amplitudes and frequencies of these tones, these being related to volume and pitch as discussed above, would give us a better grasp of the nature of the audio signal. The decomposition of a signal into a linear combination of pure tones is known as *spectral analysis* and has

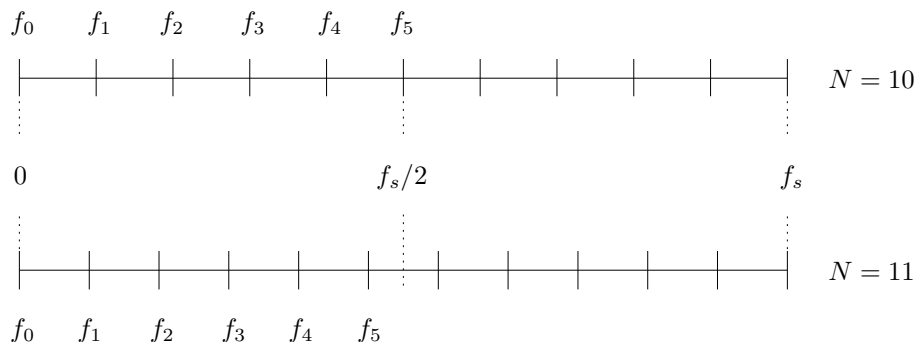


FIGURE 3. Partitioning the frequency range.

important applications in many areas of science and engineering. In what follows we will write MATLAB programs to perform spectral analysis on digital audio signals.

Let us consider digital audio signals of length N samples, where $N \geq 1$ is a fixed integer. As discussed above, we can regard each such signal s as an element of the vector space \mathbb{R}^N . Which frequencies f should we use in our spectral analysis? Because of aliasing, we know that it is useless to consider frequencies above half the sample rate f_s , but this still leaves us an infinite number of frequencies between 0 and $f_s/2$ from which to choose. To simplify the analysis, we divide this frequency range into N equal segments and define frequencies $f_0, f_1, \dots, f_m, \dots$ where $f_m = mf_s/N$ for integers $m \geq 0$. If we let M denote the largest value of m such that $f_m \leq f_s/2$, then $M = N/2$ when N is even and $M = (N - 1)/2$ when N is odd. Figure 3 illustrates this frequency partitioning for $N = 10$ and $N = 11$ (note that $M = 5$ in both cases).

Now that we know which frequencies we want to consider, we should define sampled pure tones at these frequencies. Recall from equation (1) on the preceding page that a pure tone at frequency f_m can be written as a linear combination of a cosine tone and a sine tone at this frequency. If we sample these cosine and sine tones, we obtain vectors $C_m \in \mathbb{R}^N$ and $S_m \in \mathbb{R}^N$ (respectively), namely,

$$C_m = \begin{bmatrix} 1 \\ \cos(m\theta_N) \\ \cos(2m\theta_N) \\ \cos(3m\theta_N) \\ \vdots \\ \cos(km\theta_N) \\ \vdots \\ \cos((N-1)m\theta_N) \end{bmatrix} \quad S_m = \begin{bmatrix} 0 \\ \sin(m\theta_N) \\ \sin(2m\theta_N) \\ \sin(3m\theta_N) \\ \vdots \\ \sin(km\theta_N) \\ \vdots \\ \sin((N-1)m\theta_N) \end{bmatrix}$$

where we have defined the radian angle $\theta_N = 2\pi/N$ for convenience. Now let us form a signal $s \in \mathbb{R}^N$ by taking the linear combination of these vectors C_m and S_m using scalar weights x_m and y_m (respectively), adding up the results for $0 \leq m \leq M$ so that

$$s = \sum_{m=0}^M [x_m C_m + y_m S_m] \quad (2)$$

We can express this linear combination in matrix form by defining two $N \times (M + 1)$ matrices C and D as

$$\begin{aligned} C &= [C_0 \ C_1 \ \dots \ C_M] \\ S &= [S_0 \ S_1 \ \dots \ S_M] \end{aligned}$$

and two vectors $x, y \in \mathbb{R}^{(M+1)}$ as

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix} \quad (3)$$

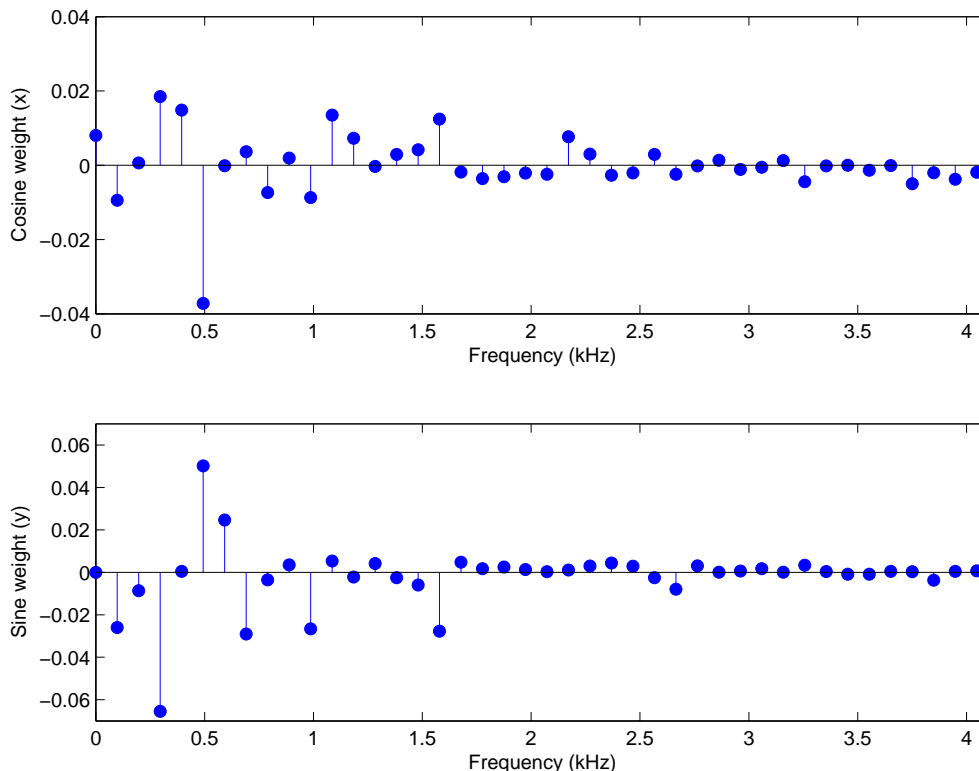


FIGURE 4. Handel in the frequency domain.

With these definitions we can write equation (2) as

$$s = Cx + Sy = \begin{bmatrix} C & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (4)$$

If we place C and S side-by-side to form a new $N \times (2M + 2)$ matrix $F = [C \ S]$, and if we stack x and y on top of one another to form a new vector

$$z = \begin{bmatrix} x \\ y \end{bmatrix}$$

then the linear combination (4) can be written in compact form as $s = Fz$. What have we accomplished here? Recall that we interpret the vector s as a time sequence of sampled audio signal values. In other words, we regard the signal s as a “function” of sampled *time* (recall the bottom portion of Figure 1). How should we interpret the vector z , namely, the vectors x and y ? We regard these vectors also as signals, but as “functions” of sampled *frequency*. Indeed, the m^{th} elements of these vectors, x_m and y_m , are the respective cosine and sine weights associated with the pure tone of frequency f_m . For example, if s is the sampled Handel excerpt shown in Figure 1, then it turns out that we can write s as the linear combination $s = Cx + Sy$ where vectors x and y are plotted versus frequency f_m in Figure 4. Again, think of the entries in these vectors x and y as mixer volume settings, where the inputs to the mixer are the sampled cosine and sine tones at the frequencies f_m , and the output of the mixer is exactly the Handel excerpt.

In most applications we are interested in the *amplitude* A_m of the pure tone at frequency f_m as in equation (1) on page 4, not its separate cosine and sine weights x_m and y_m . The plot of this amplitude $A_m = \sqrt{x_m^2 + y_m^2}$ versus frequency f_m , as illustrated in Figure 5 for the Handel excerpt, is known as the *spectrum* of the signal s and is commonly shown as a bar chart. Just as a prism breaks down a beam of white light into its color spectrum, each color representing a different frequency of light, the spectrum in Figure 5 illustrates how the Handel excerpt is broken down into its separate frequency components.

To calculate the spectrum of a given signal $s \in \mathbb{R}^N$, we must form the matrix $F = [C \ S]$ and solve the linear system of equations $Fz = s$ for the vector z . Are we guaranteed to always find a solution for z ,

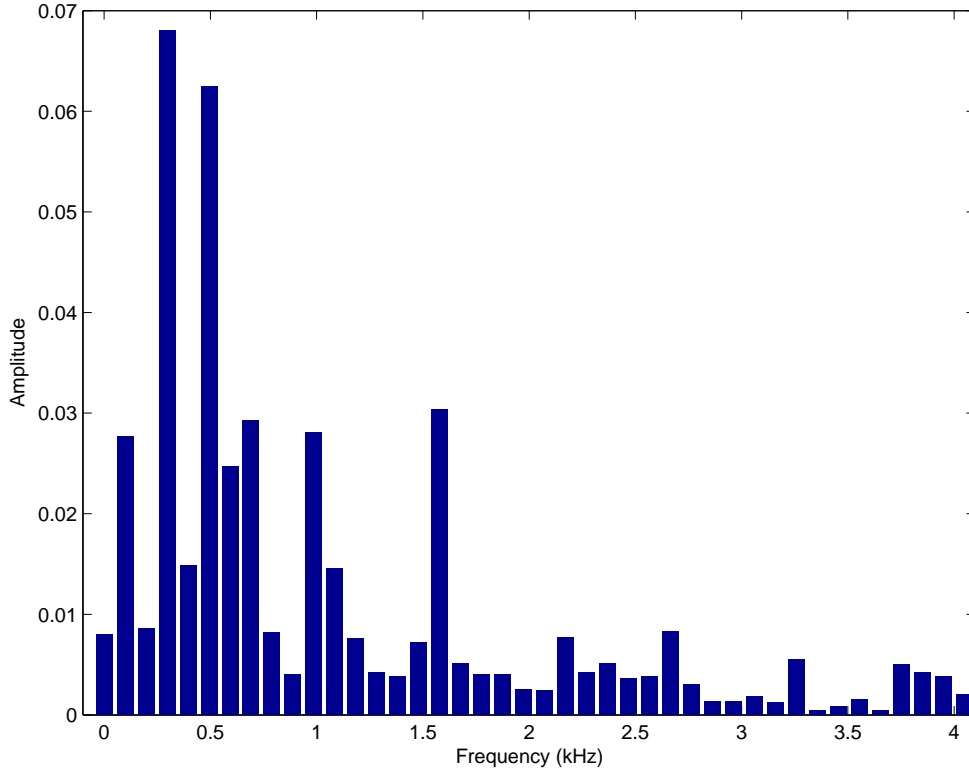


FIGURE 5. The spectrum of the Handel excerpt

regardless of our choice for the signal s ? To answer this question, we need to know the *rank* of F , namely, the dimension of its range space (recall that the range space of a matrix is the span of its columns). If this rank is equal to N , then there will always be a solution z because in this case the range space of F is the entire space \mathbb{R}^N (the only subspace of \mathbb{R}^N having dimension N is the space \mathbb{R}^N itself). If this rank is less than N , then there will be many choices for the signal s for which the system of equations $Fz = s$ admits no solution (just choose any s not belonging to the range space of F). For a given signal length N , we could try to compute the rank of F in MATLAB by generating the matrix F and then using the `rank` function. However, when N is large, the sheer size of the matrix F may make such a straightforward calculation impractical; in fact, one could easily exceed the memory capacity of the computer. Instead, we will take a closer look at this matrix F to see if we can take advantage of any special properties it may possess. As an example, let us look at F for $N = 6$:

$$F = [C \ S] = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1/2 & -1/2 & -1 & 0 & \sqrt{3}/2 & \sqrt{3}/2 & 0 \\ 1 & -1/2 & -1/2 & 1 & 0 & \sqrt{3}/2 & -\sqrt{3}/2 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1/2 & -1/2 & 1 & 0 & -\sqrt{3}/2 & \sqrt{3}/2 & 0 \\ 1 & 1/2 & -1/2 & -1 & 0 & -\sqrt{3}/2 & -\sqrt{3}/2 & 0 \end{bmatrix}$$

Note that any two different columns of this matrix F are orthogonal to each other⁵ (in other words, the columns of F form an *orthogonal set*). This is no coincidence; in fact, one can show that F will have this property for any signal length N (one simply applies certain trigonometric identities to the cosine and sine functions which make up the matrix F). As we will see below, this orthogonality property plays a crucial role in the computation of the spectrum of a signal. For now, we can use this property to determine the rank of the matrix F . Indeed, because nonzero orthogonal vectors are linearly independent [Lay, Section 6.2, Theorem 4], the rank of F is precisely the number of *nonzero* columns it has. Thus in the above example

⁵Recall that two vectors v_1 and v_2 are orthogonal when their dot product $v_1 \cdot v_2 = v_1^T v_2$ is zero.

with $N = 6$, we see that the rank of F is 6 which is the same as N . In general, the first column of S will always be the zero vector (because $\sin(0) = 0$), and if N is even then the last column of S will also be the zero vector (because $\sin(0) = \sin(\pi) = 0$). These are the only possible zero columns of F , which means F will always have exactly N nonzero columns (if N is odd, then F has $N + 1$ columns, one of which is zero, and if N is even, then F has $N + 2$ columns, two of which are zero). Therefore the rank of F is equal to N for every choice of the signal length N , which means the system of equations $Fz = s$ will always admit a solution for the vector z . Will this solution z be unique? Unfortunately no, because F has at least one zero column: if $Fz = s$ for some vector z , then we can always change the elements of z corresponding to zero columns of F without altering the product Fz (in other words, changing the volume level of absolute silence produces only absolute silence). However, if we throw out the zero columns of F , then the remaining N columns will form an *orthogonal basis* for \mathbb{R}^N [Lay, Section 6.2, Theorem 4], which means the N elements of the solution z corresponding to the N nonzero columns of F are indeed uniquely determined. If we adopt the convention that the volume settings for the zero columns of F must be themselves zero, then every signal $s \in \mathbb{R}^N$ will uniquely determine a collection z of mixer settings such that $Fz = s$. According to this convention, the first element of the vector y will always be zero, and if N is even then the last element of y will also be zero.⁶

Because the nonzero columns of F form an orthogonal basis for \mathbb{R}^N , it is easy to solve the system of equations $Fz = s$ for the mixer settings z given the signal s . There is no need for row reduction here; we simply apply the formula from [Lay, Section 6.2, Theorem 5] to obtain the following solutions for the elements x_m and y_m of the vectors x and y from (3):

$$x_m = \frac{C_m \cdot s}{C_m \cdot C_m} \tag{5}$$

$$y_m = \begin{cases} 0 & \text{when } m = 0 \text{ or } m = N/2 \text{ (by convention)} \\ \frac{S_m \cdot s}{S_m \cdot S_m} & \text{otherwise} \end{cases}$$

The dot products $C_m \cdot C_m$ and $S_m \cdot S_m$ in these solutions are independent of the signal s and can be found in advance. In fact, one can use trigonometric identities to show that $C_m \cdot C_m = N$ when $m = 0$ or $m = N/2$, and otherwise $C_m \cdot C_m = S_m \cdot S_m = N/2$. Upon plugging these values into (5), we obtain

$$x_m = \begin{cases} \frac{C_m \cdot s}{N} & \text{when } m = 0 \text{ or } m = N/2 \\ \frac{2C_m \cdot s}{N} & \text{otherwise} \end{cases} \tag{6}$$

$$y_m = \begin{cases} 0 & \text{when } m = 0 \text{ or } m = N/2 \text{ (by convention)} \\ \frac{2S_m \cdot s}{N} & \text{otherwise} \end{cases}$$

Note that the first element x_0 of the vector x is just the mean (average) of the elements of s (the vector C_0 is the vector of all ones, so the dot product $C_0 \cdot s$ is just the sum of the elements of s , and dividing by the number of elements N gives the mean).

Let us now write a MATLAB function called `spec` which computes the spectrum of a signal s .⁷ The outputs of this function will be the vectors x and y , namely, the cosine and sine parts of the spectrum. The only input to this function will be the signal s (we also need to know the signal length N , but this can be

⁶We could avoid the need for adopting such a convention by throwing out all zero columns in our definition of the matrix S in the first place. In this case $F = [C \ S]$ would be square with nonzero orthogonal columns, but the matrices C and S (and hence the vectors x and y) would no longer be the same size. This would complicate the computation and plotting of the spectrum of a signal in MATLAB, so we have chosen to include the zero columns in our definition of F and thereby maintain symmetry between the cosine and sine parts.

⁷There is already a function called `spectrum` in the MATLAB Signal Processing Toolbox, so we have chosen a different name.

found from s). We will use the formulas in (6) to calculate the elements of the vectors x and y . The steps in this calculation are as follows:

1. Compute the signal length $N = \text{length}(s)$ and the largest integer M less than or equal to $N/2$, namely, $M = \text{fix}(N/2)$.
2. Initialize the output variables x and y to zero vectors with $M+1$ elements.
3. Calculate the special cases x_m and y_m for $m = 0$. Because MATLAB begins indexing vectors at $\text{index}=1$ instead of $\text{index}=0$, the element x_0 will be stored in $x(1)$ and not $x(0)$. In general, the elements x_m and y_m will be stored in $x(m+1)$ and $y(m+1)$, respectively. Thus to find x_0 we write $x(1) = \text{mean}(s)$. We already have $y(1) = 0$ from the initialization step.
4. For each integer m in the range $0 < m < N/2$, we calculate x_m and y_m according to the formulas in (6). This is most naturally accomplished using a `for` loop.
5. If N is even, then we need to calculate the special cases x_m and y_m for $m = M = N/2$. This needs to be done only for x_M because y_M is already zero from the initialization step.

This is a simple but inefficient algorithm for computing the spectrum; the calculation can be done much faster using more advanced algorithms which are beyond the scope of this study.⁸ One way to implement this algorithm in MATLAB (without error checking) is as follows:

```
function [x,y] = spec(s)
% SPEC Compute the spectrum of a signal.
% [x,y] = spec(s) returns the cosine part x and the sine part y
% of the spectrum of the signal s.

% Find the signal length.
N = length(s);
M = fix(N/2);

% Initialize the vectors x and y.
x = zeros(M+1,1);
y = x;

% The first element x0 of the vector x is just the mean of s.
% The first element y0 of the vector y is zero by convention.
x(1) = mean(s);

% Generate the vector t so that Cm = cos(m*t) and Sm = sin(m*t).
t = 2*pi*(0:(N-1))/N;

% Find xm and ym for 0 < m < N/2.
for m = 1:(N-1)/2
    x(m+1) = 2*dot(cos(m*t), s)/N;
    y(m+1) = 2*dot(sin(m*t), s)/N;
end

% If N is even, find xm for m=M (ym=0 for m=M by convention).
if 2*M == N
    x(M+1) = dot(cos(M*t), s)/N;
end
```

To test our function, let us plot the spectrum of the 440 Hz pure tone sampled at 8 kHz which we generated on page 4:

```
fs = 8000;
N = 0.05*fs; % 50 millisecond duration
s = cos(2*pi*440*(0:(N-1))/fs); % 440 Hz tone
```

⁸This calculation time required for this simple algorithm grows like N^2 as the signal length N gets larger. With more advanced algorithms, such as the *fast Fourier transform*, the calculation time grows instead like $N \cdot \log N$ as N gets larger.

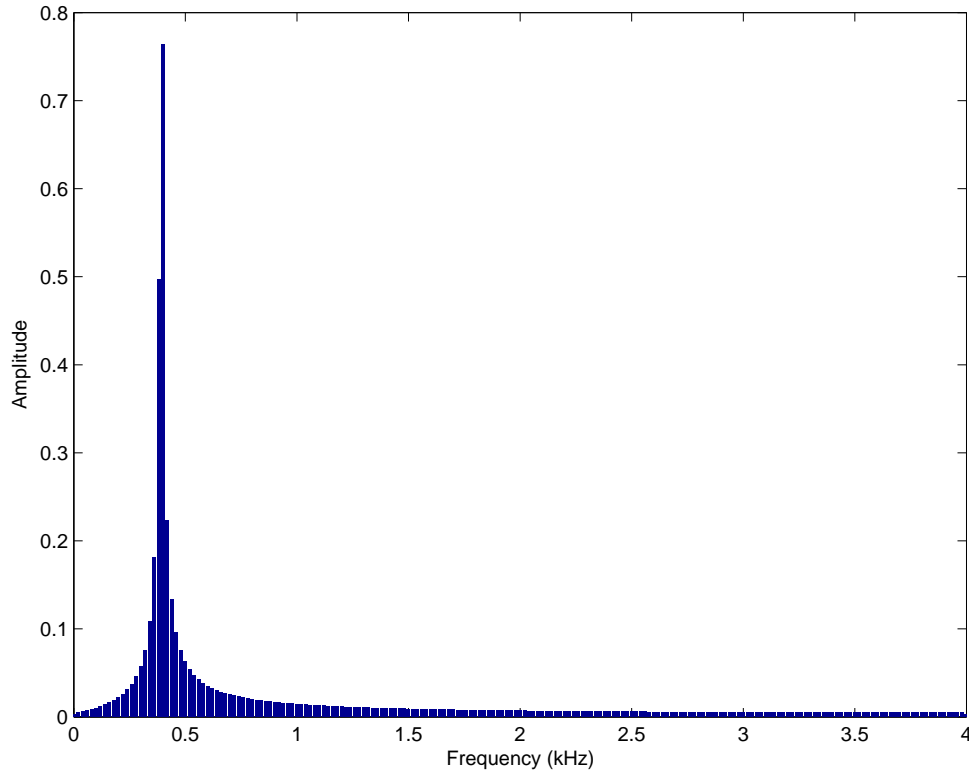


FIGURE 6. Spectrum of a 392 Hz pure tone (G above middle C), sampled at 8 kHz.

```
[x,y] = spec(s);
fm = (0:fix(N/2))*fs/N;           % vector of frequencies fm
A = sqrt(x.^2+y.^2);             % vector of amplitudes
bar(fm,A);
```

As expected, the spectrum shows a single nonzero bar of unit amplitude at a frequency of 440 Hz. If we try the same thing with a frequency of 392 Hz (G above middle C), we obtain the spectrum of Figure 6. Unlike the 440 Hz spectrum, the 392 Hz spectrum is not a single bar but is instead spread out over a range of frequencies. Why is the 392 Hz spectrum so much different? At a sample rate of 8 kHz, the frequency of 440 Hz is exactly one of the frequencies f_m used in the sampled pure tones C_m and S_m which form the columns of the matrix F ; thus in this case the signal s is simply one of the columns of F . However, the frequency 392 Hz is *not* one of these frequencies f_m , which means there is no column of F corresponding to exactly 392 Hz. As a result, the 392 Hz spectrum has large amplitudes at the adjacent frequencies $f_m=380$ Hz and $f_m=400$ Hz, but there are also significant amplitudes over a range of neighboring frequencies. This phenomenon is called *leakage*.

As a final example, let us return to the 440 Hz frequency but now use a square wave rather than a sinusoid: replace `cos(...)` with `sign(cos(...))` in the generation of the vector \mathbf{s} above. The square wave spectrum is shown in Figure 7. Not only is there a large amplitude at 440 Hz (the *fundamental frequency*), but there are also significant amplitudes at odd multiples of 440 Hz (the *odd harmonics*). Note that aliases of odd multiples also appear; for example, there is a bar at 3160 Hz which is not itself an odd multiple of 440 Hz but which is an alias of its eleventh multiple 4840 Hz.

5. THE SPECTROGRAM: MIXING TIME AND FREQUENCY DOMAINS

Suppose we have a signal s of length $N = 400$ sampled at 8 kHz such that the first half of the signal is a pure tone at 440 Hz and the second half is a pure tone at 880 Hz, one octave higher. As expected, the spectrum of this signal (Figure 8) has two primary peaks, one at 440 Hz and one at 880 Hz. Unfortunately, by looking at this spectrum we cannot tell which tone came first, the lower one or the higher one. In other

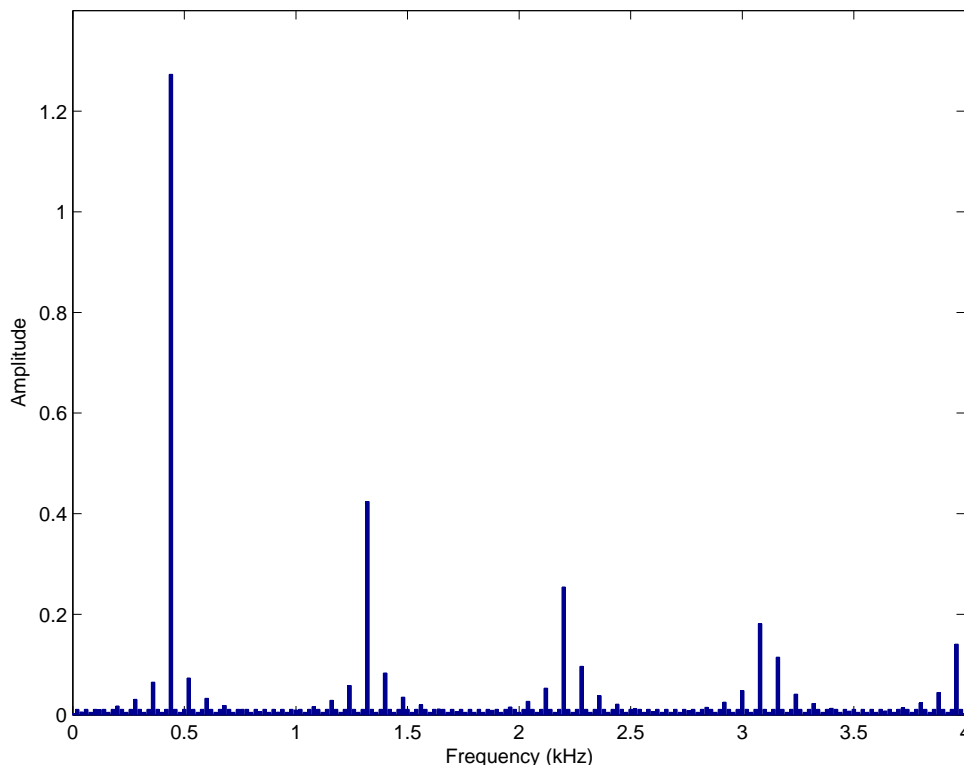


FIGURE 7. Spectrum of a 440 Hz square wave, sampled at 8 kHz.

words, all time information is lost when we move to the frequency domain. As an alternative to computing the spectrum for the entire signal, we could compute one spectrum for the first half of the signal (which would have a single peak at 440 Hz) and another for the second half (which would have a single peak at 880 Hz). In doing so we have recovered some time information: by looking at these two spectrum plots, we now know which tone came first and which came second. The penalty for splitting up the original 400-sample signal into two 200-sample sections is that we have lost resolution in the frequency domain (by cutting the effective value of N in half we obtain only half as many lines in the partition of the frequency range illustrated in Figure 3). However, such a compromise makes sense for signals such as music signals whose frequency content changes with time (most performers play different notes at different times). The goal of this section is to further explore such compromises between the time and frequency domains.

Next suppose we split our digital audio signal s into more than two parts. For example, suppose $s \in \mathbb{R}^{1000}$ but we choose to compute the spectrum with $N = 100$, namely, 100 samples at a time. We could split s into ten separate consecutive sections of 100 samples each, but we typically include some overlap; for example, with 50% overlap the first section would be samples 1 through 100, the second section would be samples 51 through 150, the third section would be samples 101 through 200, and so forth for a total of 19 sections of 100 samples each. We can visualize this process in Figure 9 using samples from the Handel excerpt. This figure shows the first three sections of the signal using $N = 100$ with 50% overlap, each section being the segment between the two vertical red lines. We can think of these red lines as the boundaries of a “window” having a constant size of N samples which slides along our signal as time progresses. With a window size of $N = 100$, the spectrum of each of these sections will be a vector of length 51, the m^{th} entry in this vector corresponding to the amplitude at frequency f_m . Let us now place these 19 different spectrum vectors side-by-side to form the columns of a 51×19 matrix P . This matrix P is called a *spectrogram* of the signal s . Each row of P corresponds to a different frequency f_m , and each column of P corresponds to a different placement of the window in time.

There are many ways to plot the data contained in a spectrogram P . One way is to plot it as a color image, the entries in the matrix P representing the pixels with their different values represented by different

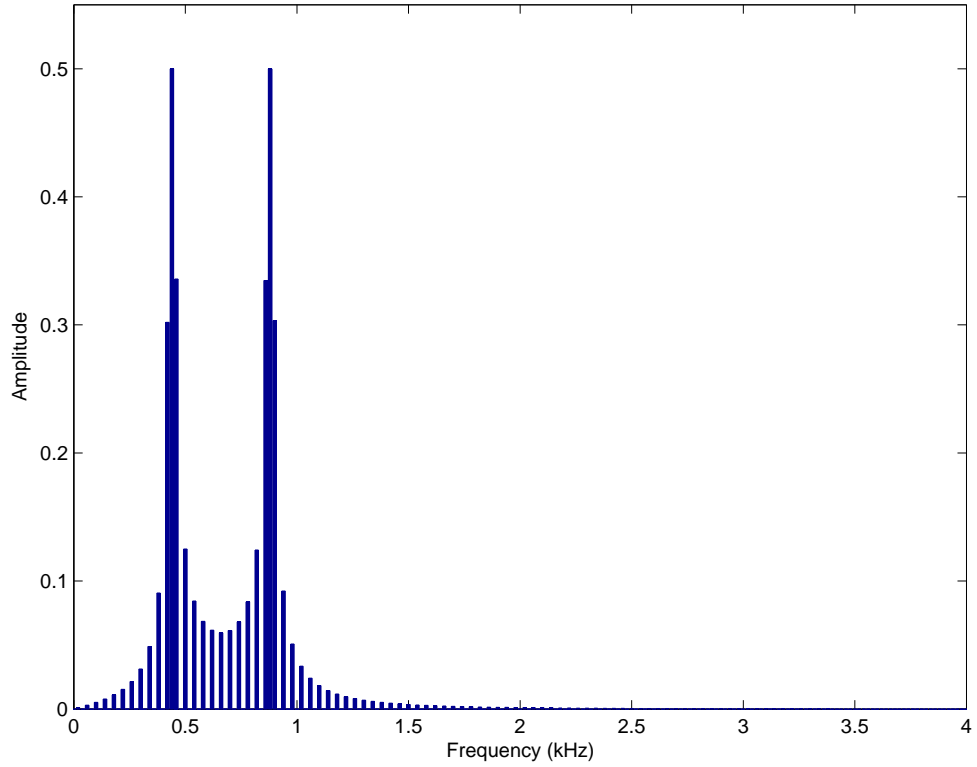


FIGURE 8. Spectrum of a change in octave from 440 Hz to 880 Hz.

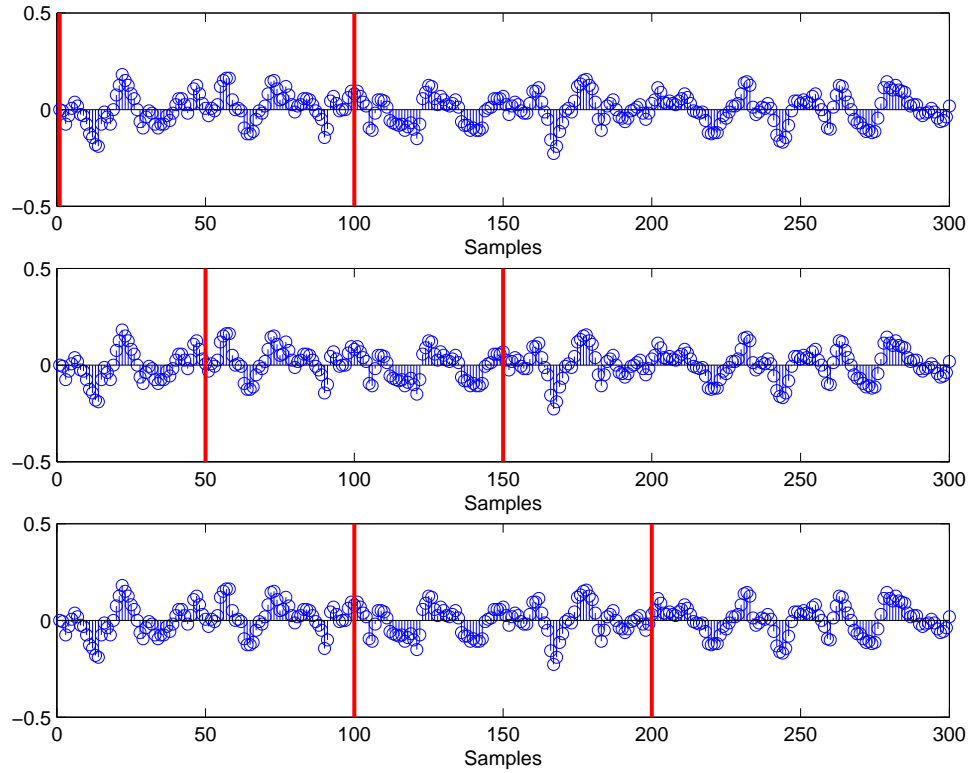


FIGURE 9. Sliding a window across the Handel excerpt.

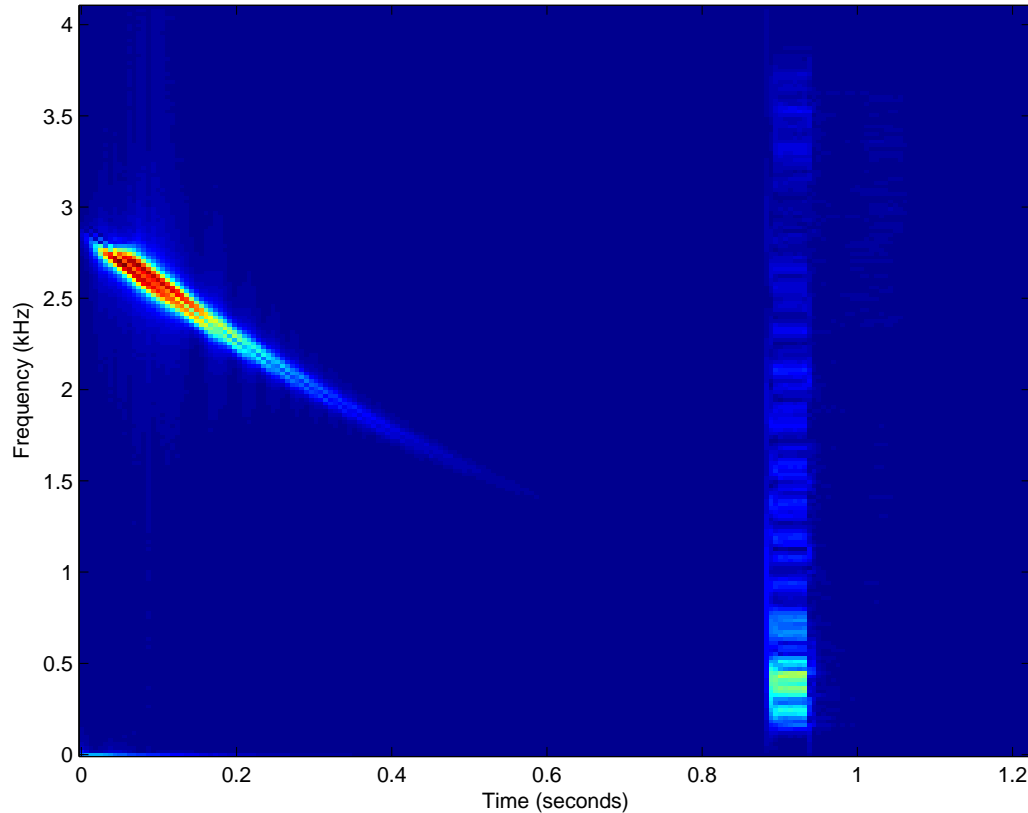


FIGURE 10. A spectrogram of the `splat` signal with window length $N = 400$ and 87.5% overlap.

colors. As an example, consider the `splat` signal we obtain by typing `load splat` in MATLAB. First listen to this signal using `sound(y,Fs)` in MATLAB (or its equivalent for your computer). The signal starts with a tone which decays in both pitch and volume, and after a moment of silence ends with a “splat” sound. These distinguishing characteristics of this signal are clearly evident in its color-image spectrogram shown in Figure 10. The sample rate for this signal is 8192 Hz, and the spectrogram was computed with window length $N = 400$ and 87.5% overlap.

We end this study by writing a MATLAB function to compute the spectrogram matrix P of a signal s . We will call the function `spectrogram`, and it will have two input arguments, the signal s and the window length N (we will assume a fixed overlap of 50%), and a single output argument P . Our function will call the `spec` function of Section 4 several times within a `for` loop, once for each shifted window. As we have already described the algorithm for computing P in detail above, we will proceed to examining sample code for implementing this algorithm (without error checking):

```
function P = spectrogram(s,N)
% SPECTROGRAM Compute the spectrogram of a signal.
% P = spectrogram(s,N) returns a matrix P containing
% spectrogram data for the signal s for window length N.

% Total number of shifted windows (assume 50% overlap)
num_win = round(2*length(s)/N) - 1;

% The sample numbers for the left edges of the windows
left_samples = round(linspace(1, (length(s)-N+1), num_win));

% Initialize P to the zero matrix.
P = zeros(fix(N/2)+1, num_win);
```

```
% Compute one spectrum for each shifted window, and store the
% amplitude as a column of P.
for ii = 1:num_win
    k = left_samples(ii);
    [x,y] = spec(s(k:(k+N-1)));
    P(:,ii) = sqrt(x.^2+y.^2);
end
```

Once we have computed the spectrogram matrix P using this function, we can use the MATLAB function `imagesc` to display P as a color image as in Figure 10.

REFERENCES

- [Lay] David C. Lay, *Linear Algebra and Its Applications*, Third edition, Addison-Wesley, 2003.
[Pohlmann] Ken C. Pohlmann, *The Compact Disc Handbook*, Second edition, A-R Editions, 1992.