AN INTRODUCTION TO RANDOM PROCESSES FOR THE SPECTRAL ANALYSIS OF SPEECH DATA

Patrick F. Reidy
Ohio State University

Abstract

Spectral analysis of acoustic data is a common analytical technique with which phoneticians have ample practical experience. The primary goal of this paper is to introduce to the phonetician, whose primary interest is the analysis of linguistic data, a portion of the theory of random processes and the estimation of their spectra, knowledge of which bears directly on the choices made in the process of analyzing time series data, such as an acoustic waveform. The paper begins by motivating the use of random processes as a model for acoustic speech data, and then introduce the spectral representation (or, spectrum) of a random process, taking care to relate this notion of spectrum to one that is more familiar to phoneticians and speech scientists. A final section presents two methods for estimating the values of the spectrum of a random process. Specifically, it compares the commonly-used (windowed) periodogram to the multitaper spectrum, and it is shown that the latter has many beneficial theoretical properties over the former.

1 Introduction

This paper discusses some of the statistical methods involved in the spectral analysis of speech data. Specifically, its aim is to introduce phoneticians to random processes, the class of mathematical object used to model speech data; their spectral representation; and some of the methods for estimating the values of a random process's spectrum.

In order to appreciate the place that random processes hold in a spectral analysis of speech data, we first consider a concrete example of such an analysis. Suppose that a researcher wishes to investigate the spectral properties of the English voiceless sibilant /s/. The first step in this investigation is to collect data by recording several tokens of /s/ from multiple English speakers. We refer to this type of data as *speech data*, measurements of the air pressure fluctuations caused by a particular speech sound wave, as sensed by a microphone. In practice, these measurements are typically stored by a digital recording device as a sequence of numbers, where each number represents the instantaneous air pressure at a given time.

So, the actual physical sound wave generated during speech production and the experimenter's record of that sound wave differ in basic ways. Whereas, the physical sound wave causes continuous air pressure fluctuations over a continuous interval of time, the record of the sound wave has been both sampled and quantized, which results in discrete air pressure fluctuations that occur over a discrete time interval. Because the researcher has access to only the record of the sound wave and because our focus is the analysis of speech data, we choose to represent a sound wave and its waveform as a numeric sequence.

Figure 1 shows the waveform of a token of /s/ that might be recorded by the researcher. The values of this waveform appear to vary randomly from one sample to the next. This random variation is expected in the waveform of /s/ because its noise source is generated by turbulent airflow, which by definition involves random air pressure variation. However, there are more fundamental sources of randomness in all speech data that affect not only turbulent sounds such as sibilants, but also quasi-periodic sounds like vowels.

One source of this randomness is the recording equipment itself. The microphone, by its very nature as a physical sensor, is subject to small random changes in its behavior over time. Since the microphone mediates the physical sound wave and its record, these small random changes in the microphone's behavior engender random errors in the recorded data. Likewise, the recording device may introduce low-frequency background noise whose intensity varies randomly over time.

Moreover, it is known that speech is subject to intra-speaker variation. The waveforms of two tokens of the same word spoken by the same person, even in proximate succession, are assured to show unpredictable differences in their values, which may be due to differences in speaking rate, vocal effort, articulatory gestures, etc. that the speaker, much less the researcher, is unable to control from one production to the next; hence, this variation can be considered random.

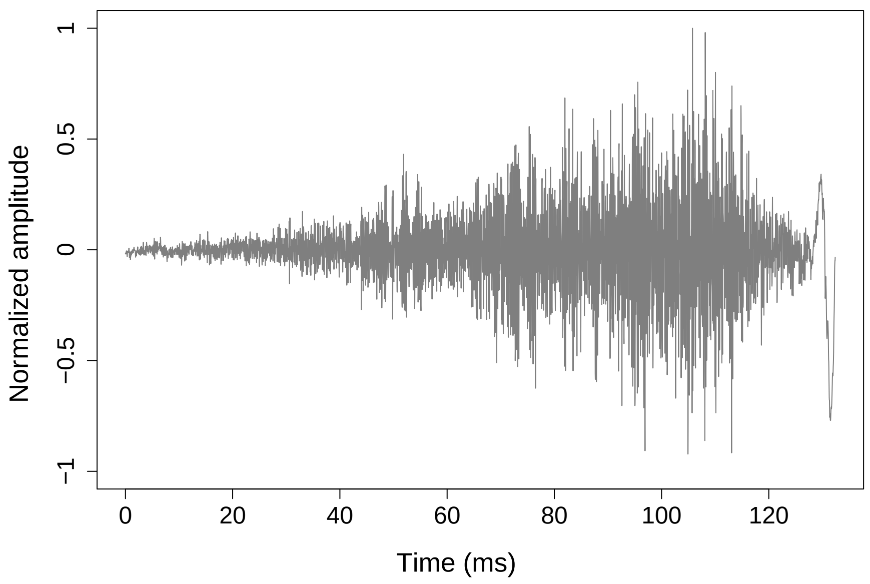

Figure 1: A realization of word-initial English /s/ excised from a token of 'sodas' as produced by an adult male speaker.

Due to the randomness intrinsic to speech data, each value of a waveform should be construed as a particular value taken by a random variable. For example, suppose that the researcher records a token of /s/ as the sequence of n numbers $x_1, x_2, \ldots, x_n$, where each value x_t is the t^{th} sampled value of the sound wave. Then, a natural model for the waveform of /s/ is a sequence of random variables $X_1, X_2, X_3, \ldots$, which just so happened to assume the values $x_1, x_2, \ldots, x_n$ when that particular token of /s/ was recorded. The decision to model the waveform of /s/ as a sequence of random variables correctly captures the fact that the values of the waveform of a token of /s/ are random in the sense discussed in the preceding two paragraphs, and motivates the introduction and definition of random processes.

Definition 1.1 (Random process)**.** A *random process* is a sequence of random variables, denoted by $\{X_t\}$, that are all defined on the same probability space, take values in the same measurable space, and are indexed by a variable t that ranges over (a subset of) the integers.

The measurable space in which each random variable takes its values is called the *state space* of the random process.

The linguistic objects suitable to be modeled by random processes are not limited to just the waveforms of phonetic segments. Indeed, the definition of a random process is a sequence of random variables, all of which are defined on a common probability space and share a common state space. The specifics of the probability space and state space are left open. So, all of the following could equally well be modeled by a random process: the waveform of a word, an f_0 track, a sequence of articulator positions, a text corpus. Each of these examples reflects a change in the state space.

When a finite number of the variables in a random process $\{X_t\}$ assume values, the result is a sequence of numbers, referred to as a *realization* of the process and denoted by $\{x_t\}$; hence, the acoustic tokens of /s/ that form the data in the example are modeled as realizations of the random process that models /s/. So, when a random process's realizations are acoustic data, it should be clear that the random process models some waveform, not some sequence of states that describe constrictions in the vocal tract during the generation of that waveform, or some sequence of articulator postures that formed those vocal tract constrictions, or some sequence of motor unit activations that postured the articulators, or any other sequence of "articulatory states" at an even earlier point in the speech chain.

When a random process $\{X_t\}$ is used to model an acoustic waveform that has been sampled, the index t represents the (discrete) ordinal points in time at which the sound wave is sampled; hence, in the example of /s/, each random variable X_t models the t^{th} value of /s/'s waveform when sampled. If the sampling period T is known in seconds, then the "time" of each random variable in the random process can be given a physical meaning by associating each random variable X_t to the time tT seconds.

Once the researcher has collected a number of /s/ tokens, the spectral analysis can begin. For concreteness, suppose that the goal of the spectral analysis is to determine the peak frequency of the spectrum of /s/. From a procedural point of view, this analysis is straightforward: First, the spectrum of each /s/ token is computed; then, the peak frequency of each spectrum is determined; and finally, these values are used to estimate the peak frequency of /s/'s spectrum.

From a conceptual point of view, however, some elaboration is needed before this type of analysis can be considered meaningful. First, the notions of the "spectrum of /s/" and "the spectrum of a token of /s/" need to be clarified. Each of these ideas is resolved through the mathematical model of each linguistic object: A token of /s/ is modeled as a sequence of numbers, whose spectrum is known from the discrete Fourier transform (DFT). Therefore, the spectrum of a token of /s/ can be understood as the spectrum of the numeric sequence that models that /s/ token.

Likewise, the spectrum of /s/ refers to the spectrum of the random process that models /s/. But since the reach of traditional Fourier theory does not extend to random processes, this immediately exposes a hole in the logic of the procedure above. That is, the DFT is a map whose domain is a particular class of numeric sequence. This domain excludes all random processes; therefore, the DFT cannot be used to transform a random process into its spectrum. The change in mathematical object, whose spectrum is to be found, demands an extension of traditional Fourier theory. Without a theory of the spectral representation of random processes, a spectral analysis of /s/, or any other phonetic segment for that matter, is devoid of meaning.

In §2, the necessary extensions to traditional Fourier theory are reviewed. Specifically, it turns out that not all random processes have a spectral representation, so conditions on a random process that guarantee the existence of its spectrum are presented. Furthermore, it is shown that each value of the spectrum of a random process $\{X_t\}$ depends on the infinite number of random variables in the process. However, $\{X_t\}$ is only ever observed as a realization of a finite number of variables; hence, each value of $\{X_t\}$'s spectrum can never be computed exactly. Instead, these values must be estimated from a finite realization.

In §3, two methods are presented for estimating the spectrum of a random process $\{X_t\}$ from a particular realization $x_1, x_2, \ldots, x_n$. Each method of estimation is evaluated analytically in order to explore how "close" to the true spectrum of $\{X_t\}$ an estimate computed from either method is expected to be. The form of these spectral estimators reveals the connection between the spectrum of $x_1, x_2, \ldots, x_n$ provided by the DFT and the spectrum of $\{X_t\}$. Specifically, both methods for estimating the spectrum of $\{X_t\}$ from $x_1, x_2, \ldots, x_n$ are based on the DFT of $x_1, x_2, \ldots, x_n$. This discussion, by extension, elucidates how the spectrum of a token of /s/ may be considered a representation of the spectrum of /s/.

Since the ultimate goal of the spectral analysis described above is the estimation of the peak frequency of /s/, rather than just its spectrum, the relationship between this or any other spectral property of /s/ and an estimate of it from a token of /s/ should be clarified as well. Mathematically, a spectral property of /s/ corresponds to a transformation of the spectrum of a random process $\{X_t\}$. For example, if the spectrum of $\{X_t\}$ is denoted by f_X, which ranges over a variable ω that denotes frequency, then the peak frequency of /s/ is given by the transformation

$$\mathsf{Peak}(X) = \arg\max_{\omega} f_X(\omega).$$

Similarly, a spectral property of a token of /s/ corresponds to a transformation of a spectral estimate computed from a realization $x_1, x_2, \ldots, x_n$. If this spectral estimate is denoted by S_x, which ranges over the discrete variable ω_j, then the peak frequency of the /s/ token is given by

$$\mathsf{Peak}(x) = \arg\max_{\omega_j} S_x(\omega_j).$$

While the discussion in §3 tells how S_x relates to f_X, it says nothing about how $\mathsf{Peak}(x)$ relates to $\mathsf{Peak}(X)$. The paper concludes with a discussion of the difficulties attendant with determining analytically how a spectral property of a random process $\{X_t\}$ relates to an estimate of that property computed from a realization $x_1, x_2, \ldots, x_n$. This difficulty of analysis implies that an analytic comparison of different methods for estimating a spectral property is for all practical purposes intractable. Instead, the researcher must justify which method of spectral estimation yields the "best" estimate of a given spectral property, by way of simulation rather than assuming that the relative merits of one spectral estimator over another transfer to estimates of spectral properties derived from that estimator.

2 Spectral representation of a random process

In this section, the theory of spectral representation for random processes is reviewed. The discussion is based on Shumway and Stoffer (2006), and the reader is referred there for a thorough general introduction to random processes and their spectral representation.

In the sequel, upper case letters $X, Y, \ldots$ are used to denote random variables; $\mathbb{E}(X)$ denotes the expected value of the random variable X; $\mathrm{Var}(X)$ denotes the variance of X; and $\mathrm{Cov}(X, Y)$ denotes the covariance between the random variables X and Y.[1] It is assumed that the reader is familiar with the meaning of all these terms.

In general, the methods from classical statistics cannot be applied to a random process because these methods assume that the random variables $\{X_t\}$ are independent and all follow the same distribution; however, a random process will not always obtain both of these properties. When used to model the waveform of a phonetic segment, the dependence structure of a random process and the change in its distributional properties over time are due to the nature of and physical constraints on speech production. First, speech production necessarily involves the movement of articulators, and as the posture of the articulators changes over time, the generated sound wave changes as well; hence, the distributional properties of a random process that models the wave form are expected to change with time as well. Second, the articulators move smoothly during the production of speech, and the posture that they can assume next depends on their current postural state. This dependence is projected forward in the speech chain, to the acoustic sound wave.

A complete description of the dependence structure of a random process $\{X_t\}$ would be had from knowing the joint cumulative distribution function of all finite subsets of random variables in $\{X_t\}$; however, such a complete description is usually unattainable. Instead, a much more limited description of $\{X_t\}$'s dependence structure is taken from its autocovariance function, which reports the covariance between each pair of variables in $\{X\}$. Below it is shown that the autocovariance function is intimately related to the spectral representation of a random process.

Definition 2.1 (Autocovariance function)**.** If $\{X_t\}$ is a random process, then the *autocovariance function* γ_X is defined by

$$\gamma_X(s, t) = \mathrm{Cov}(X_s, X_t). \tag{1}$$

In general, a process does not have a spectral representation; however, there is a very general subclass of random processes—the (weakly) stationary processes—which do admit such a frequency-domain representation.

[1] In general, it is possible that any of $\mathbb{E}(X)$, $\mathrm{Var}(X)$, or $\mathrm{Cov}(X, Y)$ may not converge to a value, making that value undefined; however, in the sequel it is assumed that all random variables have a finite expected value and variance and that all pairs of random variables have a finite covariance.

Definition 2.2 (Stationary process)**.** A random process $\{X_t\}$ is said to be *(weakly) stationary*[2] if it satisfies the following conditions:

1. $\mathbb{E}(X_t) = \mu$, for all t in the index set;
2. $\mathrm{Var}(X_t) < \infty$, for all t in the index set;
3. $\mathrm{Cov}(X_s, X_t) = \mathrm{Cov}(X_{s+h}, X_{t+h})$, for all s, t, $s+h$, and $t+h$ in the index set.

A process that does not satisfy the three conditions above is said to be *non-stationary*.

The third condition says that the covariance between any two random variables in a stationary process depends only on the amount of time that separates them, which allows its autocovariance function to be expressed in terms of a single variable denoting the separation between two random variables in the process: If $\{X_t\}$ is a stationary process with autocovariance function γ_X, then for all indices s and t with $h = s - t$, it follows that

$$\begin{aligned}\gamma_X(s,t) &= \gamma_X(s+h, t+h)\\ &= \gamma_X(s+h, s)\\ &= \gamma_X(h, 0),\end{aligned}$$

which does not depend on either time argument s or t. Hence, the autocovariance function of a stationary process can be expressed as a function of just the separation (or *lag*) h between two random variables,

$$\gamma_X(h) =_{def} \mathrm{Cov}(X_0, X_h). \tag{2}$$

If the $\{X_t\}$ models a waveform that is sampled with sampling period T seconds, then the lag h that separates two random variables in the process can be given the physical meaning of a separation of hT seconds.

The spectral representation of a stationary process can now be introduced. It can be proved that any stationary process can be expressed as a random linear combination of simple periodic functions oscillating at different frequencies (Shumway and Stoffer, 2006, Theorem C.2). Additionally, the autocovariance function of a stationary process also has a spectral representation, which is provided by the following theorem, stated without proof (Shumway and Stoffer, 2006, Property P4.1 & Theorem C.3).

Theorem 2.3 (Spectral Representation)**.** *If $\{X_t\}$ is a stationary process whose autocovariance function γ_X satisfies*

$$\sum_{h=-\infty}^{\infty} |\gamma_X(h)| < \infty,$$

[2]By contrast, a process is said to be *strictly stationary* if the distributional properties of all finite subcollections of random variables in the process do not depend on time.

then there is a unique function f_X for which

$$\gamma_X(h) = \int_{-1/2}^{1/2} f_X(\omega) e^{2\pi i \omega h} \, \mathrm{d}\omega, \qquad h = 0, \pm 1, \pm 2, \ldots \tag{3}$$

The function f_X in (3) is called the spectral density *or* spectrum *of $\{X_t\}$ and is defined by*

$$f_X(\omega) = \sum_{h=-\infty}^{\infty} \gamma_X(h) e^{-2\pi i \omega h}, \qquad \omega \in \mathbb{R}. \tag{4}$$

Readers who are familiar with traditional Fourier theory may notice that the spectral density f_X above is the Fourier transform of the (aperiodic, discrete) autocovariance function γ_X (Beerends et al., 2003, §18.5). This implies that f_X and γ_X uniquely determine each other, and that the spectral density f_X and the autocovariance function γ_X contain the same information since each value of γ_X can be recovered from f_X by integrating the right-hand side of (3). Therefore, we take the spectral density f_X of a stationary process $\{X_t\}$ as its foremost spectral representation and in the remainder of this section present some of the practical consequences of Theorem 2.3 for the spectral analysis of speech data.

2.1 Existence of a spectral representation of speech data

The first of these consequences concerns the speech data that a researcher is able to use to investigate spectral properties of a phonetic segment's waveform. Recall the example from the introduction, in which the waveform of /s/ is modeled by a random process $\{X_t\}$, and the data used in the study are modeled as realizations of $\{X_t\}$. In this setting, Theorem 2.3 implies that it is only meaningful to talk about the spectrum of the waveform of /s/ if that waveform is stationary. If /s/'s waveform is not stationary, then a stationary portion of /s/ must be isolated and used for the purposes of the spectral analysis.

Since the waveform of /s/ is only ever observed through a realization of it, this condition on the existence of a spectrum of (a portion of) /s/'s waveform, this raises the question of how to determine whether a particular token of /s/ is a realization of a stationary process. A rough but common method for checking this involves plotting the recorded token $x_1, x_2, \ldots, x_n$ as a function of time, and visually inspecting the mean and variance properties of its waveform. In particular, if the data is a realization of a stationary process, then it follows from definition 2.2(1) that the mean of $x_1, x_2, \ldots, x_n$ should be constant across time, and it follows from the following proposition that the variance should be constant as well.

Proposition 2.4 (Variance of a stationary process)**.** *If $\{X_t\}$ is a stationary process, then for every X_s and X_t in the process,* $\mathrm{Var}(X_s) = \mathrm{Var}(X_t)$.

Proof. Let X_s and X_t be random variables from a stationary process, and let $h = t - s$. Then, $\mathrm{Var}(X_s) = \mathrm{Cov}(X_s, X_s) = \mathrm{Cov}(X_{s+h}, X_{s+h}) = \mathrm{Cov}(X_t, X_t) = \mathrm{Var}(X_t)$.

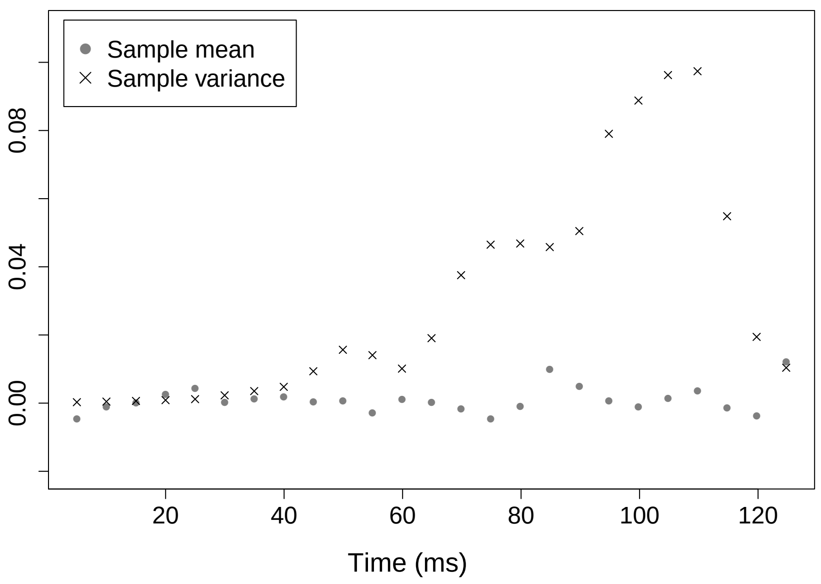

Figure 2: The sample mean (gray) and sample variance (black) of successive 10 ms data windows, with 5 ms overlap among adjacent windows, taken from the /s/ token shown in Figure 1. The value of each statistic is plotted against the time of the midpoint of the data window from which it was calculated.

The first and last equalities follow from the definition of covariance, and the second equality follows from the assumption that X_s and X_t come from a stationary process. □

Figure 2 shows the temporal progression of the sample mean and sample variance of successive 10 ms windows taken from the /s/ token shown in Figure 1. These statistics estimate the behavior of the evolution of the mean and variance of the random process $\{X_t\}$ that models /s/. From these plots, it is seen that the mean remains approximately constant, but the variance increases with time before decreasing sharply. So, this token of /s/ does not seem to be a realization of a stationary process, which, when considered in light of Theorem 2.3, suggests that it would be imprudent, much less meaningful, to use all the data from this token to estimate spectral properties of /s/.

In order to surmount this problem, the data can be used to hypothesize the location of a stationary subprocess of $\{X_t\}$, whose spectrum can be used as a proxy for that of the entire process. The data in Figure 2 suggest that the initial 40 ms interval of /s/ is stationary, as are the intervals between 40 and 60 ms and 70 and 90 ms. However, when automating a spectral analysis over a large data set, it is practically impossible to inspect each token individually in order to locate a stationary portion. Instead, it is common practice to take from each token a short interval placed in the same relative location, e.g. a 20 ms interval centered at the temporal midpoint of the waveform. It is taken on faith that the interval is of short enough duration that the statistical properties of the random process do not change too drastically to violate the condition of stationarity. The fact that phoneticians typically restrict spectral analyses to “steady-state” portions of speech data suggests that random processes already occupy a very real, albeit unappreciated, role in phonetic analyses.

2.2 The domain of the spectral density function

In equation (4), the spectral density function f_X is defined over the entire real line; however, phoneticians are accustomed to visualizing the spectrum of speech data only on the interval of frequency values that ranges from 0 to the Nyquist frequency, $1/2T$ Hz, where T is the sampling period of the recorded data. Propositions 2.5 and 2.7 reconcile this discrepancy between theory and practice.

Proposition 2.5 (f_X is a periodic function)**.** *If $\{X_t\}$ is a random process that models an acoustic wave that is sampled with a sampling period T seconds, then its spectral density f_X is a periodic function with period $1/T$ Hz.*

Proof. From the discussion immediately following equation (2), each lag value h corresponds to hT units of time if $\{X_t\}$ models an acoustic wave that is sampled with sampling period T seconds. Therefore, equation (4) can be written as

$$f_X(\omega) = \sum_{h=-\infty}^{\infty} \gamma_X(hT)e^{-2\pi i\omega hT}, \tag{5}$$

where ω is expressed in Hz. Evaluating f_X at $\omega + 1/T$ then yields

$$\begin{aligned} f_X(\omega + 1/T) &= \sum_{h=-\infty}^{\infty} \gamma_X(hT)e^{-2\pi i(\omega+1/T)hT} && (6) \\ &= \sum_{h=-\infty}^{\infty} \gamma_X(hT)e^{-2\pi i\omega hT}e^{-2\pi ih} && (7) \\ &= \sum_{h=-\infty}^{\infty} \gamma_X(hT)e^{-2\pi i\omega hT} && (8) \\ &= f_X(\omega). && (9) \end{aligned}$$

Equation (8) follows by virtue of the identity $e^{i\pi n} = 1$ for any integer n; in this case, $n = -2h$. □

The preceding proposition shows that the values of f_X are determined by the values that it takes on any interval whose size is $1/T$. Proposition 2.7 shows that the size of this interval can effectively be cut in half.

Lemma 2.6 (γ_X is an even function)**.** *If γ_X is the autocovariance function of a stationary process $\{X_t\}$ as defined by equation (2), then γ_X is an even function in the sense that $\gamma_X(-h) = \gamma_X(h)$ for all h.*

Proof. If γ_X is as described in the statement of the lemma, then

$$\begin{aligned}
\gamma_X(h) &= \mathrm{Cov}(X_0, X_h) && (10)\\
&= \mathrm{Cov}(X_{-h}, X_0) && (11)\\
&= \mathrm{Cov}(X_0, X_{-h}) && (12)\\
&= \gamma_X(-h) && (13)
\end{aligned}$$

Equation (11) follows from definition 2.2(3); equation (12), from the elementary fact that $\mathrm{Cov}(X, Y) = \mathrm{Cov}(Y, X)$ for all random variables X and Y. □

Proposition 2.7 (f_X is an even function)**.** *If f_X is the spectral density function of a stationary process as defined by equation (4), then f_X is an even function.*

Proof. If f_X is the spectral density function of a stationary process, then

$$f_X(-\omega) = \sum_{h=-\infty}^{\infty} \gamma_X(h)e^{2\pi i\omega h}. \qquad (14)$$

Making the change of variable $h = -j$ yields

$$\begin{aligned}
f_X(-\omega) &= \sum_{j=\infty}^{-\infty} \gamma_X(-j)e^{-2\pi i\omega j} && (15)\\
&= \sum_{j=\infty}^{-\infty} \gamma_X(j)e^{-2\pi i\omega j}. && (16)
\end{aligned}$$

Equation (16) follows from Lemma 2.6. Comparison of equation (16) to equation (4) reveals that the former is just an alphabetic variant of the latter, where the summation is carried out in reverse. Therefore, it follows that $f_X(-\omega) = f_X(\omega)$, which proves the proposition. □

Taken together Propositions 2.5 and 2.7 show that if $\{X_t\}$ is a stationary process that models an acoustic wave sampled with sampling period T seconds, then its spectrum f_X is completely determined by the values that f_X takes on the frequency interval $[0, 1/2T]$ Hz. Since f_X is an even function, the values that it takes on the interval $[0, 1/2T]$ can be used to reconstruct its values on the interval $[-1/2T, 1/2T]$. The size of this reconstructed interval is $1/T$; hence, the values taken by f_X on this interval completely determine f_X on its entire domain since f_X is periodice with period $1/T$. Consequently, the spectrum of any given waveform need only be considered on the interval $[0, 1/2T]$, and in the following section all graphs of these spectra are shown only on this interval.

3 Spectral Estimation

In equation (4), each ordinate of the spectral density, $f_X(\omega)$, is expressed in terms of the autocovariance function γ_X; however, it is possible to express each ordinate in terms of the random variables in $\{X_t\}$ by replacing γ_X in (4) with the righthand side of equation (2),

$$f_X(\omega) = \sum_{h=-\infty}^{\infty} \mathrm{Cov}(X_0, X_h) e^{-2\pi i \omega h}, \qquad \omega \in \mathbb{R}. \tag{17}$$

From this equation, it is immediately clear that the computation of each ordinate of f_X requires knowledge of the distributional properties of all the random variables in $\{X_t\}$; however, the random process is only ever observed as a finite realization $x_1, x_2, \ldots, x_n$. So, the value of each ordinate $f_X(\omega)$ cannot be computed exactly, but must instead be estimated.

A method for estimating the ordinates of f_X, which often takes the form of a function of a finite number of random variables $X_1, X_2, \ldots, X_n$ from a stationary process $\{X_t\}$, is referred to as a *spectral estimator*. The random variables $X_1, X_2, \ldots, X_n$ are called a *sample* of the process $\{X_t\}$. This section presents two spectral estimators that have been used in the spectral analysis of speech data: the windowed periodogram and the multitaper spectrum. Each of these spectral estimators finds its roots in the discrete Fourier transform (DFT), a spectral transform that is typically defined in terms of a finite numeric sequence (see Beerends et al. (2003, p. 360)). For the discussion of spectral estimators that follows, it is more convenient to define the DFT in terms of random variables rather fixed numbers.

Definition 3.1 (Discrete Fourier transform)**.** If $X_1, X_2, \ldots, X_n$ is a finite sequence of random variables from a stationary process $\{X_t\}$, then the *discrete Fourier transform* d_X of the sample is defined by

$$d_X(\omega_j) = \sum_{t=1}^{n} X_t e^{-2\pi i \omega_j t}, \tag{18}$$

where $\omega_j = j/n$ for $j = 0, \ldots, n-1$. The frequencies ω_j are referred to as the *Fourier frequencies*.

Other commonly encountered spectral transformations derived from the DFT are the *amplitude spectrum* $|d_X|$, defined by

$$|d_X|(\omega_j) = |d_X(\omega_j)|, \tag{19}$$

and the *power spectrum* $|d_X|^2$, defined by

$$|d_X|^2(\omega_j) = |d_X(\omega_j)|^2. \tag{20}$$

In equation (18), each ordinate of the DFT, $d_X(\omega_j)$ is defined as a sum of the random variables $X_1, X_2, \ldots, X_n$; hence, $d_X(\omega_j)$ is a univariate random variable since a sum of univariate random variables is itself a univariate random variable. Furthermore, each $d_X(\omega_j)$ estimates the value of $f_X(\omega_j)$, and as such is an example of a *point estimator*, i.e. an estimator of a single value. It follows that it is meaningful to investigate each ordinate's distributional properties, such as its expected value and its variance. Knowlege of these properties for $d_X(\omega_j)$ enables a discussion of its *bias* and *mean square error (MSE)* as a point estimator. The latter is commonly used as a measure of the quality of a point estimator, so by extension the spectral estimators presented below can be compared via the MSE of their ordinates.

Definition 3.2 (Bias)**.** If $\hat{\theta}$ is a point estimator of a number θ, then the *bias* of $\hat{\theta}$, denoted $\beta\left(\hat{\theta}\right)$, is defined to be $\beta\left(\hat{\theta}\right) = \mathbb{E}\left(\hat{\theta}\right) - \theta$.

If $\beta\left(\hat{\theta}\right) = 0$, then $\hat{\theta}$ is said to be an *unbiased* estimator.

Definition 3.3 (Mean square error)**.** If $\hat{\theta}$ is a point estimator of a number θ, then the *mean square error* of $\hat{\theta}$, denoted $\mathsf{MSE}\left(\hat{\theta}\right)$, is defined to be $\mathsf{MSE}\left(\hat{\theta}\right) = \beta\left(\hat{\theta}\right) + \mathrm{Var}\left(\hat{\theta}\right)$.

The rest of this section is devoted to introducing and comparing the windowed periodogram and the multitaper spectrum. For each spectral estimator, its bias and variance are discussed only qualitatively; however, some comparison of the two estimators is still possible.

3.1 The windowed periodogram

The periodogram arises from scaling the power spectrum in (20) by the inverse of the number n of random variables available to the estimator.

Definition 3.4 (Periodogram)**.** If $X_1, X_2, \ldots, X_n$ are a sample from a stationary process $\{X_t\}$, then the *periodogram* I_X of the sample is defined by

$$I_X(\omega_j) = n^{-1}|d_X(\omega_j)|^2, \tag{21}$$

where j and ω_j are as they are in definition (3.1).

The periodogram is, in some sense, the most "direct" estimator of the spectral density f_X given a particular sample $X_1, X_2, \ldots, X_n$. To see why this is so, recall the definition of f_X from equation (4). One immediately apparent method for estimating $f_X(\omega)$ is to estimate the autocovariance function γ_X and then compute the DFT of the result. A common estimator of γ_X is the *sample autocovariance function* (Shumway and Stoffer, 2006, p. 30).

Definition 3.5 (Sample autocovariance function)**.** If $X_1, X_2, \ldots, X_n$ is a sample of a random process $\{X_t\}$, then the *sample autocovariance function* is defined by

$$\hat{\gamma}_X(h) = n^{-1}\sum_{t=1}^{n-h}(X_{t+h} - \bar{X})(X_t - \bar{X}), \tag{22}$$

where $\bar{X} = n^{-1}\sum_t X_t$ is called the *sample mean*.

It is possible to show that for Fourier frequencies other than $\omega_0 = 0$ the DFT of the sample autocovariance function is equal to the periodogram.

Proposition 3.6 (DFT of the sample autocovariance function)**.** *If $X_1, X_2, \ldots, X_n$ is a sample of a stationary process $\{X_t\}$, with sample autocovariance function $\hat{\gamma}_X$ and periodogram I_X, then for Fourier frequencies other than $\omega_0 = 0$,*

$$I_X(\omega_j) = \sum_{|h|<n} \hat{\gamma}_X(h) e^{-2\pi i \omega_j h}.$$

Proof. First note that for $\omega_j \neq 0$, the DFT can be written as[3]

$$d_X(\omega_j) = \sum_{t=1}^{n} (X_t - \bar{X}) e^{-2\pi i \omega_j t}, \tag{23}$$

Therefore, for Fourier frequencies other than ω_0 it follows that

$$\begin{aligned} I_X(\omega_j) = n^{-1}|d_X(\omega_j)|^2 &= n^{-1} \sum_{t=1}^{n}\sum_{s=1}^{n} (X_t - \bar{X})(X_s - \bar{X}) e^{-2\pi i \omega_j (t-s)} & (24)\\ &= n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} (X_{t+|h|} - \bar{X})(X_t - \bar{X}) e^{-2\pi i \omega_j h} & (25)\\ &= \sum_{|h|<n} \hat{\gamma}_X(h) e^{-2\pi i \omega_j h}. & (26) \end{aligned}$$

Comparing equation (26) to definition 3.1, it is clear that the periodogram is equal to the DFT of the sample autocovariance function. □

This proposition establishes a nice parallel among the spectral representations of a stationary process and a sample of that process: The spectrum of each is the Fourier transform of its appropriate autocovariance function. In order to establish a more direct relationship between the spectral density and an estimator of it, the *windowed periodogram* is introduced.

Definition 3.7 (Windowed periodogram)**.** If $X_1, X_2, \ldots, X_n$ is a sample of a stationary process, and $w_1, w_2, \ldots, w_n$ is a sequence of numbers, then the *w-windowed periodogram* I_{wX} of the sample is defined by

$$I_{wX}(\omega_j) = n^{-1} \sum_{t=1}^{n} w_t X_t e^{-2\pi i \omega_j t}. \tag{27}$$

The sequence of numbers $w_1, w_2, \ldots, w_n$ is referred to as a *data window* or *data taper*.

[3]For any complex number $z \neq 1$, $\sum_{t=1}^{n} z^t = z(1 - z^n)/(1 - z)$. Let $\omega_j \neq 0$ and let $z = \exp(-2\pi i \omega_j)$. Then, $z \neq 1$ and $z^n = \exp(-2\pi i \omega_j n) = \exp(-2\pi i j n/n) = \exp(-2\pi i j) = 1$ since j is an integer; hence, $\sum_{t=1}^{n} z^n$ = z(1-1)/(1-z) = 0.
Then, to prove equation (23), expand the sum therein and cancel the $\bar{X} \sum_{t=1}^{n} e^{-2\pi i \omega_j t}$ term.

In Proposition 3.8 below, it is assumed that $X_1, X_2, \ldots, X_n$ is a sample from a zero-mean process $\{X_t\}$, meaning that $\mathbb{E}(X_t) = 0$, for each random variable X_t in the process. It is likely that a zero-mean process is a valid model for the acoustic waveform of speech because the sound waves generated during speech production travel as a chain of increases and decreases in air pressure, which are likely to cancel each other over time. Indeed, the data shown in Figure 2 suggest that the acoustic waveform of /s/ is well-modeled by a zero-mean process.

Proposition 3.8 (Expected value of periodogram ordinates). *If $X_1, X_2, \ldots, X_n$ is a sample of a zero-mean stationary process $\{X_t\}$, and $w_1, w_2, \ldots, w_n$ is a data window, then the expected value of the w-windowed periodogram I_{wX} is*

$$\mathbb{E}[I_{wX}(\omega_j)] = \int_{-1/2}^{1/2} W_n(\omega_j - \omega) f_X(\omega)\ \mathrm{d}\omega, \tag{28}$$

where

$$W_n(\omega) = n^{-1} \left| \sum_{t=1}^{n} w_t e^{-2\pi i \omega t} \right|^2, \quad w \in \mathbb{R}. \tag{29}$$

W_n is called the kernel *of the data window $w_1, w_2, \ldots, w_n$.*

Proof. If the righthand side of (27) is expanded and one of the variables of summation changed to $h = t - s$, the result is

$$I_{wX}(\omega_j) = n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} w_t w_{t+|h|} X_t X_{t+|h|} e^{-2\pi i \omega_j h}.$$

Taking the expectation of both sides yields

$$\begin{aligned}
\mathbb{E}[I_{wX}(\omega_j)] &= n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} w_t w_{t+|h|} e^{-2\pi i \omega_j h} \mathbb{E}(X_t X_{t+|h|}) &(30)\\
&= n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} w_t w_{t+|h|} e^{-2\pi i \omega_j h} \mathbb{E}[(X_t - \mathbb{E}(X_t))(X_{t+|h|} - \mathbb{E}(X_{t+|h|}))] &(31)\\
&= n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} w_t w_{t+|h|} e^{-2\pi i \omega_j h} \gamma_X(h), &(32)
\end{aligned}$$

where the first equation follows from the linearity of the expected value operator; the second, from the fact that $\{X_t\}$ is a zero-mean process; and the third from equation (2).

Finally, substituting the righthand side of (3) for $\gamma_X(h)$ gives

$$\mathbb{E}[I_{wX}(\omega_j)] = \int_{-1/2}^{1/2} n^{-1} \sum_{|h|<n} \sum_{t=1}^{n-|h|} w_t w_{t+|h|} e^{-2\pi i(\omega_j-\omega)h} f_X(\omega)\ \mathrm{d}\omega \tag{33}$$

$$= \int_{-1/2}^{1/2} n^{-1} \left| \sum_{t=1}^{n} w_t e^{-2\pi i(\omega_j-\omega)t} \right|^2 f_X(\omega)\ \mathrm{d}\omega \tag{34}$$

$$= \int_{-1/2}^{1/2} W_n(\omega_j-\omega) f_X(\omega)\ \mathrm{d}\omega, \tag{35}$$

where the last equation follows from (29). □

Proposition 3.8 shows that the w-windowed periodogram I_{wX} and the spectral density f_X are mediated by the kernel W_n of the particular data window used on the sample. More specifically, the integral on the righthand side of equation (35) says that the expected value of the estimator $I_{wX}(\omega_j)$ is found by taking the kernel W_n, "laying it on top" of f_X so that $W_n(0)$ coincides with $f_X(\omega_j)$, multiplying the values of each function that overlap, and then summing these products. This operation is called the *convolution* of W_n and f_X.

However, it is important to recognize that equation (35) does not describe how the windowed periodogram estimate of $f_X(\omega_j)$ is computed from a realization; that information is found in equation (27). Instead, equation (35) provides information about how the estimator $I_{wX}(\omega_j)$ would behave if a number of estimates of $f_X(\omega_j)$ were computed from different realizations and then averaged, which is a doorway to the bias of $I_{wX}(\omega_j)$.

3.1.1 Bias properties of the windowed periodogram

It should be clear from equation (35) that in order for the expected value of $I_{wX}(\omega_j)$ to be determined, it is necessary to know both the kernel W_n and the spectral density f_X; however, in applications involving speech data, it is rarely the case that anything is known about f_X since this would require knowledge of the distributional properties of the random process $\{X_t\}$ that models the speech data. It is possible that such distributional knowledge could become available from a complete theory of the aeroacoustics of speech production, but at the moment this theory is lacking. Consequently, the bias of each ordinate in the windowed periodogram, $\beta\left(I_{wX}(\omega_j)\right) = \mathbb{E}\left[I_{wX}(\omega_j)\right] - f_X(\omega_j)$, is unknown because both terms involved in its computation depend on f_X.

Since the direct computation of $\beta\left(I_{wX}(\omega_j)\right)$ is often impossible in practice, the bias properties of the windowed periodogram are explored through the kernel W_n, whose form depends on the particular window applied to the data before the spectral estimate is computed. This section discusses the kernel's of two data windows that should be familiar to phoneticians and other speech researchers: the rectangular window and the Hamming window.

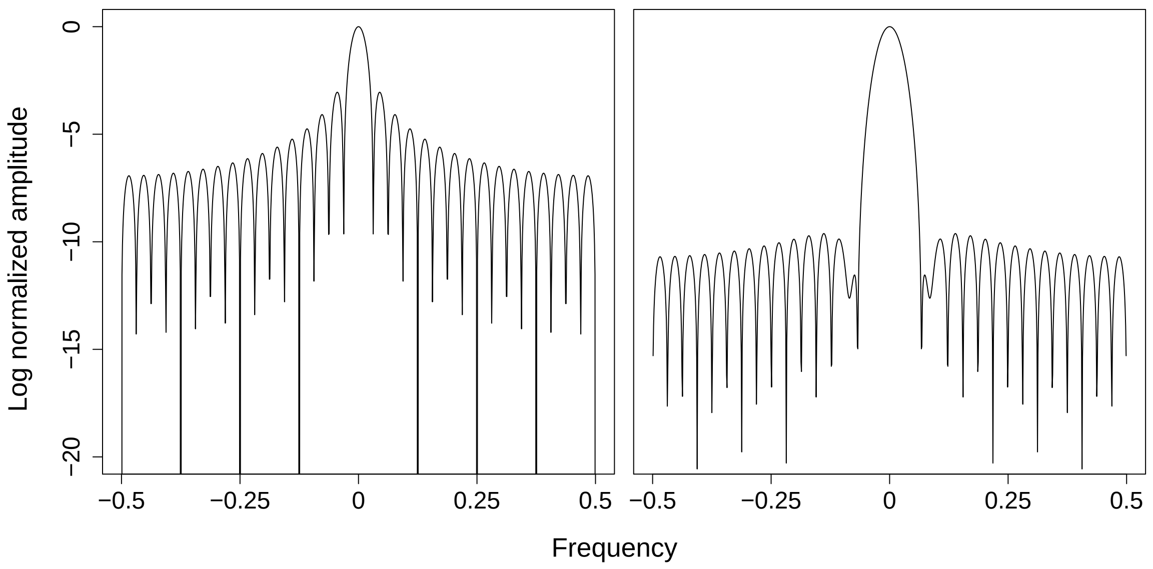

Figure 3: The kernel of the 32-point rectangular window (left panel) and the 32-point Hamming window (right panel). The values of each kernel were normalized by dividing by the maximum value of the kernel.

Definition 3.9 (Rectangular window)**.** The *(n-point) rectangular window* $r_1, r_2, \dots, r_n$ is the sequence of n elements, each of which is equal to 1

Since $r_t X_t = X_t$ for $t = 1, 2, \dots, n$, the rectangular window can be thought of as the "default" data window applied to the sample $X_1, X_2, \dots, X_n$ when no other data window is used. From this it follows that the periodogram from Definition 3.4 is equal to the windowed periodogram from Definition 3.7 whose data window is the rectangular window; hence, the relationship between the windowed periodogram ordinate $I_{wX}(\omega_j)$ and the spectral density f_X established in Proposition 3.8 applies to the "unwindowed" periodogram as well, which implies that the bias properties of the periodram ordinates depend on the kernel of the rectangular window.

The kernel of the 32-point rectangular window is shown in the left panel of Figure 3. The shape of this kernel is characterised by a dominant peak, called the *main lobe*, centered at 0 with several other peaks, referred to collectively as the *side lobes*, on either side of it, whose respective heights decrease with their distance from the main lobe. While the number of side lobes in the kernel of a rectangular window depends on the length n of the window, the downward sloping pattern from the peak of the main lobe through the peaks of the side lobes is the same independent of n.

Consider how, according to equation (35), the shape of a kernel W_n affects the expected value, and by extension the bias, of $I_{wX}(\omega_j)$. The bias of $I_{wX}(\omega_j)$ is minimized when the righthand side of this equation equals $f_X(\omega_j)$; however, when W_n and f_X are convolved, the value of f_X at each frequency $\omega \neq \omega_j$ is scaled by $W_n(\omega_j - \omega)$, and if $W_n(\omega_j - \omega) \neq 0$, then $W_n(\omega_j - \omega) f_X(\omega) \neq 0$ as well. Consequently, the degree to which $\mathbb{E}[I_{wX}(\omega_j)]$ is influenced by the value of the spectral density at a frequency $\omega \neq \omega_j$ is directly related to the magnitude of $W_n(\omega_j - \omega)$. Therefore, the height of the sidelobes of W_n gives some indication of the extent to which $\mathbb{E}[I_{wX}(\omega_j)]$ is corrupted by the values of f_X at frequencies different, and potentially far away, from $f_X(\omega_j)$, which in turn increases the magnitude of its bias. In sum, the height of the sidelobes of a kernel is a rough proxy measure of the bias of the spectral estimator related to that kernel—the greater the height of the kernel's sidelobes, the more biased the estimator.

The righthand panel of Figure 3 shows the kernel of the 32-point Hamming window.

Definition 3.10 (Hamming window)**.** The *n-point Hamming window* $h_1, h_2, \ldots, h_n$ is the sequence of numbers defined by

$$h_t = 0.5\left(1 - \cos\left(\frac{2\pi(t-1)}{n-1}\right)\right), \qquad t = 1, 2, \ldots, n. \tag{36}$$

The size of the sidelobes in the kernel of the Hamming window, relative to those in the rectangular window's kernel, suggests that the Hamming-windowed periodogram I_{hX} has better bias properties than the rectangular-window periodogram I_{rX}. Further support for this conclusion is provided by Figure 4, which shows a Hamming-window periodogram spectral estimate overlaid on a rectangular-window periodogram estimate, both of which were computed from the center 20 ms of the token of /s/ shown in Figure 1. Both estimates suggest that the most prominent peak of the spectral density occurs just below 5 kHz. Taking this together with Proposition 3.8 and both panels of Figure 3, it is expected that at high frequencies the values of the rectangular-windowed periodogram estimate would be higher than those of the hamming-windowed periodogram estimate since the sidelobes of the rectangular window's kernel are larger than those of the Hamming window's kernel.

While the differences in bias properties between the rectangular- and the Hamming-windowed periodogram are borne out by the example estimates in Figure 4, for the purposes of analyzing speech data, it is more important to focus on how or whether these differences affect the analysis rather than to focus on the purely theoretical concern as to whether they exist at all. For example, both windowed periodogram estimates share roughly the same shape; hence, if the analysis dictates that after the spectrum is estimated, its values are used to compute statistics that summarize its shape, e.g. the first four spectral moments, then it is not a foregone conclusion that the two spectral estimators will deliver different results, just by virtue of their having different bias properties.

3.1.2 Variance of the windowed periodogram

While the use of a data window such as the Hamming window can reduce the bias of each ordinate of the periodogram, the ordinates of the Hamming-windowed periodogram

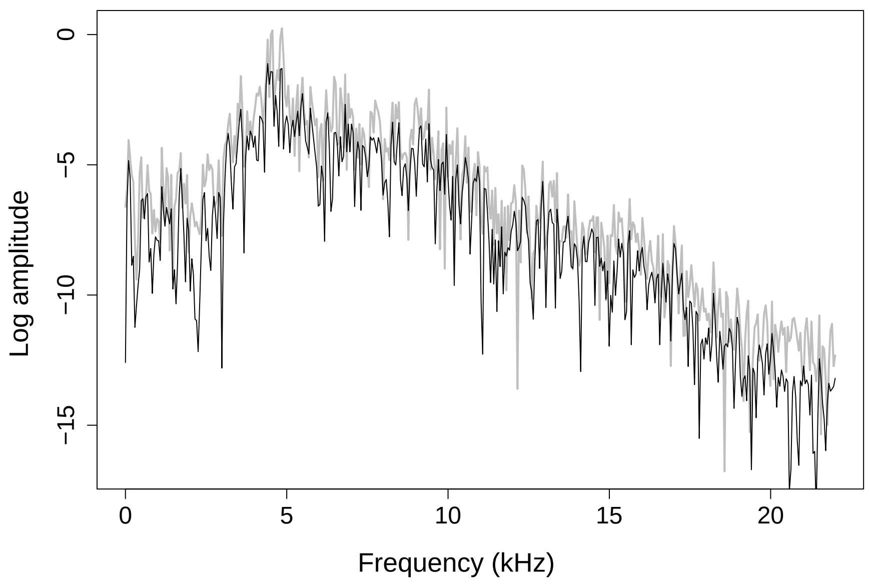

Figure 4: A comparison of a Hamming-window periodogram estimate (thin black line) and a rectangular-window periodogram estimate (thick gray line). Both estimates were computed from the center 20 ms of the token of /s/ shown in Figure 1.

I_{hX} are still prone to having a large MSE because of their large variance. The following theorem, based on Shumway and Stoffer (2006, p. 193, Property P4.2) and stated without proof, establishes the asymptotic distribution of each $I_{hX}(\omega_j)$, from which it is possible to investigate the variance of the estimator's ordinates.

Theorem 3.11 (Distribution of the windowed periodogram ordinates)**.** *If ω_j, $j = 0, 1, \ldots, n-1$, are distinct Fourier frequencies such that $f_X(\omega_j) \neq 0$, for all j, and if for each ω_j, $\{j_n\}$ is a sequence of integers such that $j_n/n \longrightarrow \omega_j$ as $n \longrightarrow \infty$, then as $n \longrightarrow \infty$,*

$$I_{hX}(j_n/n) \xrightarrow{d} \frac{f_X(\omega_j)}{2}\chi_2^2, \tag{37}$$

where $\xrightarrow{d}$ denotes convergence in distribution.

Hence, the variance of each ordinate of a Hamming-windowed periodogram is approximately

$$\operatorname{Var}\left[I_{hX}(\omega_j)\right] \approx \operatorname{Var}\left[\frac{f_X(\omega_j)}{2}\chi_2^2\right] \tag{38}$$

$$= \left(\frac{f_X(\omega_j)}{2}\right)^2 \operatorname{Var}[\chi_2^2] \tag{39}$$

$$= f_X(\omega_j)^2. \tag{40}$$

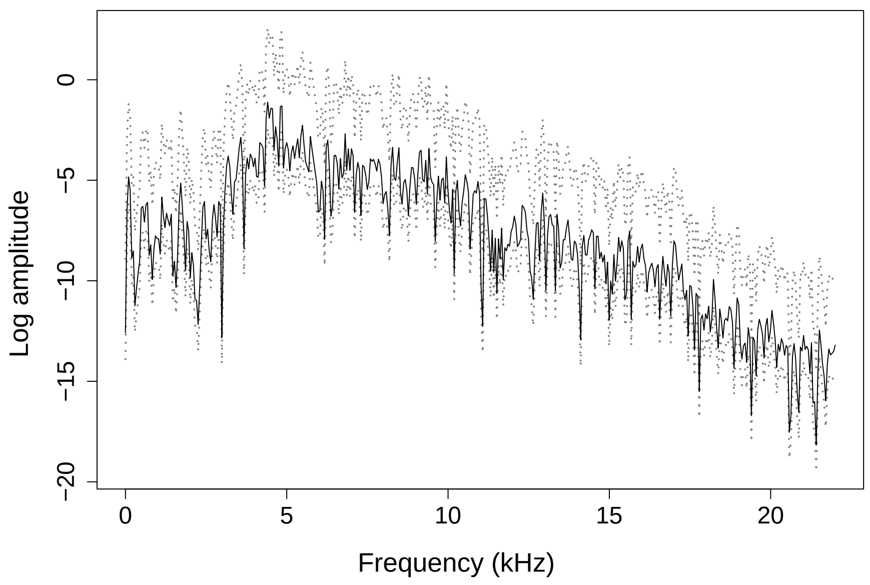

Figure 5: The Hamming-window periodogram estimate (black line) redrawn from Figure 4 plotted with the upper and lower bounds (gray dotted line) of a 95% confidence interval for each ordinate.

The asymptotic distribution of $I_{hX}(\omega_j)$ can also be used to approximate a confidence interval for $f_X(\omega_j)$ with confidence level $(1-\alpha)$. For a given α such that $0 < \alpha < 1$, under the asymptotic distribution of $I_{hX}(\omega_j)$ in (37), there is $(1-\alpha)$ probability that $I_{hX}(\omega_j)$ falls within the interval

$$\frac{f_X(\omega_j)}{2}\chi_2^2(\alpha/2) \leq I_{hX}(\omega_j) \leq \frac{f_X(\omega_j)}{2}\chi_2^2(1-\alpha/2),$$

where $\chi_2^2(\alpha)$, which is referred to as the *lower α probability tail*, is the number that satisfies $\mathbb{P}\left(\chi_2^2 < \chi_2^2(\alpha)\right) = \alpha$. Rearranging the terms in the above inequality yields a $100(1-\alpha)\%$ confidence interval for $f_X(\omega_j)$:

$$\frac{2I_{hX}(\omega_j)}{\chi_2^2(1-\alpha/2)} \leq f_X(\omega_j) \leq \frac{2I_{hX}(\omega_j)}{\chi_2^2(\alpha/2)}. \tag{41}$$

These confidence intervals can be visualized by plotting their upper and lower bounds against frequency. Figure 5 shows the Hamming-windowed periodogram estimate from Figure 4 along with the upper and lower bounds of a 95% confidence interval for each ordinate.

Furthermore, the form of the inequality in (41) suggests how the size of each confidence interval can be reduced. Specifically, the size of each confidence interval is directly related to the distance between the lower $(1-\alpha/2)$ and the lower $\alpha/2$ probability tails for the chi-squared distribution with two degrees of freedom. If it were possible to find a spectral estimator whose ordinates had an asymptotic distribution that depended on a distribution

δ whose variance was less than that of χ_2^2, then the distance between the lower $(1 - \alpha/2)$ and the lower $\alpha/2$ probability tails of δ would be less than that for χ_2^2, and the size of the confidence interval for each ordinate of the spectral estimator would decrease as well. The desire for such a reduced-variance estimator motivates the introduction of the multitaper spectrum.

3.2 The multitaper spectrum

The multitaper spectrum was introduced by Thomson (1982), and the method of its calculation is simple enough: K copies of a sample $X_1, X_2, \ldots, X_n$ of a stationary process are weighted by K different data windows $\{w_{k,t}\}$. Then, for each windowed realization $\{w_{k,t}x_t\}$, its *eigenspectrum* S_k is found by computing its power spectrum. Finally, the K eigenspectra are averaged to produce the multitaper spectrum $M_X^{(K)}$.

It is also easy to get a sense for why this method would yield a spectral estimator, the variance of whose ordinates is less than that of the Hamming-windowed periodogram. If for a fixed Fourier frequency ω_j, the K ordinates $\{S_k(\omega_j)\}$ all have equal variance and are pairwise uncorrelated, then the ordinate of the multitaper spectrum at that frequency, $M_X^{(K)}(\omega_j)$, will have variance that is $1/K$ the size of the variance of $S_k(\omega_j)$. The aim is therefore to find data windows that will yield uncorrelated eigenspectra whose ordinates each have reasonable variance.

Data windows that satisfy these conditions are found in the family of discrete prolate spheroidal (DPS) sequences (Slepian and Pollak, 1961; Landau and Pollak, 1961, 1962; Slepian, 1964). These sequences were originally discovered as a solution to the spectral concentration problem, which asks whether it is possible to find a sequence of finite duration whose spectrum contains the maximal proportion of its energy in a fixed frequency band. To state the problem more concretely, the Fourier transform of a finite sequence is introduced (Beerends et al., 2003, § 18.5).

Definition 3.12 (Fourier transform of finite sequence)**.** If $x_1, x_2, \ldots, x_n$ is a finite sequence of real numbers, then its Fourier transform $\mathcal{X}$ is defined by

$$\mathcal{X}(\omega) = \sum_{k=1}^{n} x_k e^{-2\pi i \omega k}, \qquad -1/2 \le \omega \le 1/2. \tag{42}$$

If $x_1, x_2, \ldots, x_n$ is a sequence of length n, whose Fourier transform is $\mathcal{X}$, and W is a frequency such that $0 < W < 1/2$, then the *spectral concentration* of $\mathcal{X}$ in the band $[-W, W]$, denoted by $\lambda(n, W)$ is defined as

$$\lambda(n, W) = \frac{\int_{-W}^{W} |\mathcal{X}(\omega)|^2 \, \mathrm{d}\omega}{\int_{-\infty}^{\infty} |\mathcal{X}(\omega)|^2 . \, \mathrm{d}\omega} \tag{43}$$

The spectral concentration problem asks whether, given parameters n and W as above, it is possible to find the sequence that maximizes $\lambda(n, W)$. It turns out that the answer to this question is positive (Percival and Walden, 1993, Ch. 3 & 8). Moreover, it is possible to rank the sequences of length n according to their concentration $\lambda(n, W)$. This leads to the definition of a DPS sequence.

Definition 3.13 (DPS sequence)**.** Given fixed parameters n and W to the spectral concentration problem, the *DPS sequence of order k*, denoted by $\{v_t^{(k)}\}$ is the $(k+1)^{\text{th}}$ maximal concentration $\lambda(n, W)$.

So, the sequence that has the greatest energy concentration in the frequency band $[-W, W]$ is the DPS sequence of order 0; the sequence that has the second greatest energy concentration in $[-W, W]$ is the DPS sequence of order 1; and so on.

In order to generate DPS sequences that can be used as data windows for a spectral estimator is it necessary to set the frequency bandwidth parameter W. Conventionally, W is chosen so that the product nW is an integer that satisfies $nW \leq 4$. Futhermore, the choice of W places an upper bound on the number of eigenspectra K that are averaged to compute the multitaper spectrum. In particular, K should satisfy $K \leq 2nW$ (Percival and Walden, 1993, pp. 334-5).

As an illustration, DPS sequences were generated using the `multitaper` package for `R`, with the parameters $n = 883$ (the number of data points in a 20 ms waveform sampled at 44.1 kHz) and $W = 4/883$. For these parameters, the DPS sequences of orders $k = 0$ through $k = 5$ are shown in the top row of Figure 6. The corresponding eigenspectra for the center 20 ms of the /s/ from Figure 1 are shown in the bottom row of that same figure.

It can be shown that the ordinates of each eigenspectrum S_k all have the same asymptotic distribution as the ordinates of the Hamming-window periodogram (Percival and Walden, 1993, p. 343). That is, for all k such that $0 \leq k \leq K$ and j such that $0 \leq j \leq n-1$, the spectral ordinate estimator $S_k(\omega_j)$ converges in distribution to a scaled χ_2^2 random variable.

The DPS sequences are mutually *orthogonal*, in the sense that, for all orders j and k such that $j \neq k$,

$$\sum_{t=1}^{n} v_t^{(j)} \cdot v_t^{(k)} = 0,$$

which ensures that the eigenspectra used in the computation of the multitaper spectrum are pairwise uncorrelated (Percival and Walden, 1993).

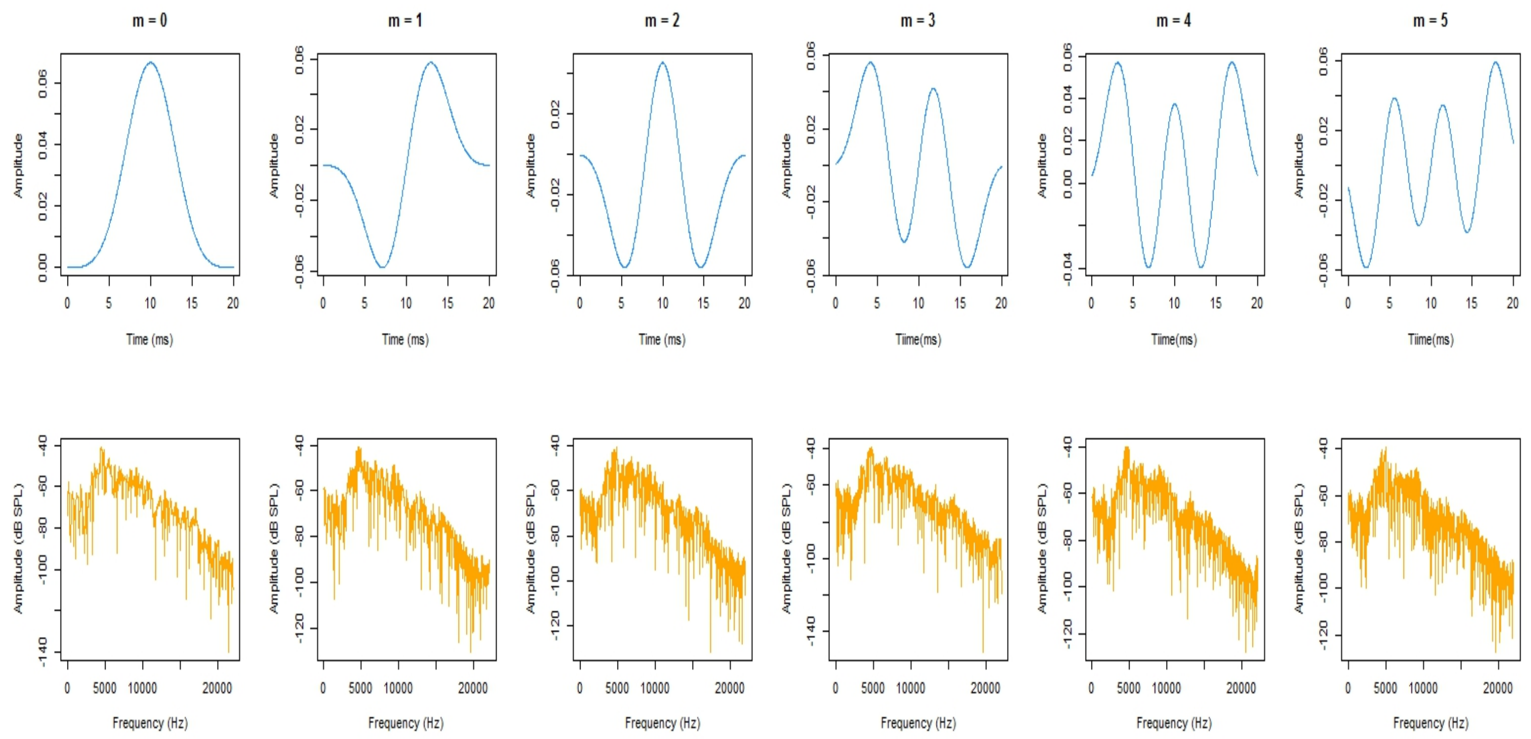

Figure 6: *Top row:* The DPS sequences of order $m = 0$ to $m = 5$, computed using the parameters $n = 883$ and $W = 4/883$. *Bottom row:* The eigenspectrum of the center 20 ms window of the /s/ from Figure 1, each computed using the DPS sequence above as a data window.

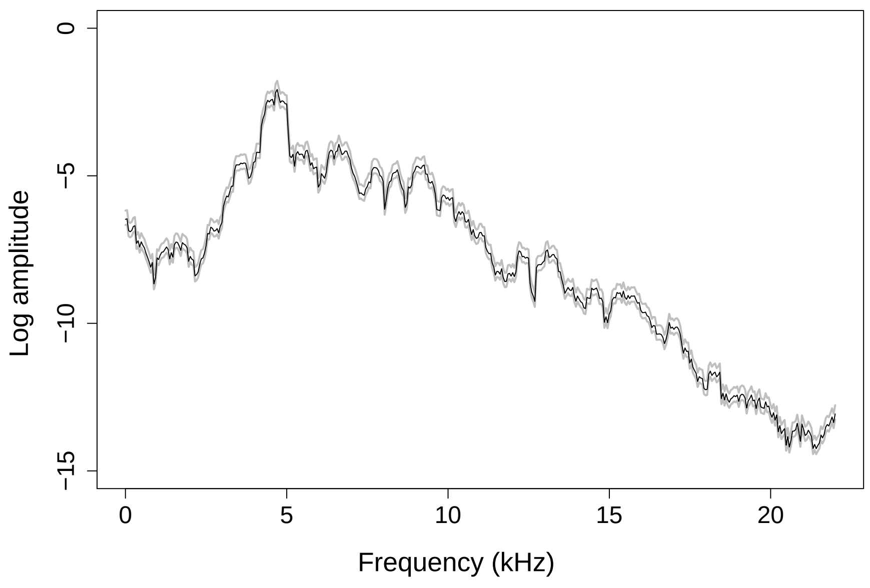

Figure 7: The multitaper spectrum (black line) of the center 20 ms of the /s/ from Figure 1, plotted with the upper and lower bounds (gray lines) of a 95% confidence interval for each ordinate.

The K mutually uncorrelated eigenspectra are averaged pointwise to compute the multitaper spectrum $M_X^{(K)}$; hence, the asymptotic distribution of each ordinate of $M_X^{(K)}$ is a scaled chi-squared with $2K$ degrees of freedom:

$$M_X^{(K)}(\omega_j) = \frac{1}{K}\sum_{k=0}^{K-1} S_k(\omega_j) \xrightarrow{d} \frac{f_X(\omega_j)}{2K}\chi^2_{2K}. \tag{44}$$

Using this distribution to approximate the variance of each ordinate $M_X^{(K)}(\omega_j)$ yields

$$\mathrm{Var}\left[M_X^{(K)}(\omega_j)\right] \;=\; \left(\frac{f_X(\omega_j)}{2K}\right)^2 \mathrm{Var}\left[\chi^2_{2K}\right] \;=\; \frac{f_x(\omega_j)^2}{K}. \tag{45}$$

Comparing (45) to (40), it is obvious that the ordinates of a multitaper spectrum have $1/K$ the variance of the ordinates of a Hamming-window periodogram.

The benefit of this reduced variance is revealed by the size of the confidence intervals for the ordinates of the multitaper spectrum. The confidence interval for each ordinate $M_X^{(K)}(\omega_j)$ is calculated analogously to (41),

$$\frac{2K\,M_X^{(K)}(\omega_j)}{\chi^2_{2K}(1-\alpha/2)} \leq f_X(\omega_j) \leq \frac{2K\,M_X^{(K)}(\omega_j)}{\chi^2_{2K}(\alpha/2)}. \tag{46}$$

Figure 7 shows the multitaper spectrum of the 20 ms of /s/ plotted with upper and lower bounds of a 95% confidence interval for each ordinate. The size of these confidence intervals is noticeably smaller than those in Figure 5: the mean size of the confidence intervals for the ordinates of the Hamming-window periodogram estimate is 4.982, while the mean value of those of the multitaper spectrum estimate is 0.486. The correct interpretation of this difference in size is that when estimating an *ordinate* of the spectral density at a given frequency, $f_X(\omega_j)$, it is possible to circumscribe a smaller set of values within which $f_X(\omega_j)$ is likely to fall if the multitaper spectrum is used rather than the Hamming-window periodogram.

This brief introduction to spectral estimation has focused on two spectral estimators that have been used in speech applications: the Hamming-windowed periodogram and the multitaper spectrum. The comparison of these two estimators was carried out primarily in terms of the asymptotic variance of each estimator's ordinates, and it was shown that the multitaper spectrum has a variance that is a fraction of that of the Hamming-windowed periodogram.

4 Conclusion

In this paper, the spectral representation theory for random processes was reviewed, and two methods for estimating the spectrum of a random process were introduced and evaluated. The evaluation of the spectral estimators was carried out in theoretical terms, e.g. by comparing the variance of the asymptotic distribution of each estimator's ordinates. It was shown that the multitaper spectrum is a much "better" estimator than the Hamming-windowed periodogram in the sense that the variance of the former's ordinates is a fraction of that of the latter.

The reader is advised to bear in mind that this notion of "better" is purely theoretical, and in practice a spectral analysis of speech data usually does not end with the estimation of a spectrum, but with the estimation of *properties* of a spectrum, e.g. the peak frequency or one or more of the formants or spectral moments. Therefore, the comparison of the multitaper spectrum and the Hamming-windowed periodogram presented in §3 does not settle the question of which estimator is better-suited to a particular spectral analysis. In fact, it doesn't even address the question since doing so can only be done meaningfully once the details of the analysis are known.

References

Beerends, R. J.; H. G. ter Morsche; J. C. van den Berg; and E. M. van de Vrie. 2003. *Fourier and Laplace transforms*. Cambridge University Press, Cambridge, UK.

Landau, H. J., and H. O. Pollak. 1961. Prolate spheroidal wave functions, Fourier analysis, and uncertainty—ii. *Bell System Technical Journal* 40.65–84.

Landau, H. J., and H. O. Pollak. 1962. Prolate spheroidal wave functions, Fourier analysis, and uncertainty—iii: The dimension of the space of essentially time- and band-limited signals. *Bell System Technical Journal* 41.1295–1336.

Percival, Donald B., and Andrew T. Walden. 1993. *Spectral analysis for physical applications: Multitaper and conventional univariate techniques*. Cambridge University Press, Cambridge, UK.

Shumway, Robert H., and David S. Stoffer. 2006. *Time series analysis and its applications*. Springer Texts in Statistics. Springer, 2nd edition.

Slepian, David. 1964. Prolate spheroidal wave functions, Fourier analysis, and uncertainty—iv: Extensions to many dimensions; generalized prolate spheroidal functions. *Bell System Technical Journal* 43.3009–3058.

Slepian, David, and H. O. Pollak. 1961. Prolate spheroidal wave functions, Fourier analysis, and uncertainty—i. *Bell System Technical Journal* 40.43–64.

Thomson, David J. 1982. Spectrum estimation and harmonic analysis. *Proceedings of the IEEE* 70.1055–1096.

Appendices

The appendices below contain `R` code that can be used for the spectral analysis of speech data—in particular, the computation of Hamming-windowed periodogram and multitaper spectrum estimates.

A **`Waveform.r`**

```
# Author:        Patrick Reidy
# Affiliations: The Ohio State University
#                Department of Linguistics
#                www.ling.ohio-state.edu
#                Learning To Talk
#                www.learningtotalk.org
# Email:         reidy@ling.ohio-state.edu
# Mail:          24A Oxley Hall
#                1712 Neil Ave.
#                Columbus, OH 43210-1298
# License:       GPL-3


# The Waveform package depends on the Simon Urbanek's
# 'audio' package.
library('audio', quietly=TRUE)
```

```
#######################
#  Utility functions  #
#######################

`%@%` <- function(...) {
# %@% is a generic function for getting the value of an
# object’s attribute(s).
# Methods available in the Waveform package:
#   %@%.default
  UseMethod(’%@%’)
}

`%@%.default` <- function(object, attribute) {
# %@%.default is the default method for getting the value of
# an object’s attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#                   slot of object.
# Returns:
#   The value of the R object in the attribute slot of
#   object.
  attr(object, attribute)
}

`%@%<-` <- function(...) {
# %@%<- is a generic function for setting the value of an
#   object’s attribute(s).
# Methods available in the Waveform package:
#   %@%<-.default
  UseMethod(’%@%<-’)
}

`%@%<-.default` <- function(object, attribute, value) {
# %@%<-.default is the default method for setting the value
# of an object’s attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#                   slot of object.
#       value: Any R object.
# Returns:
#   Nothing.  Instead, the value of object’s attribute slot
#   is changed to value.
  `attr<-`(object, attribute, value)
```

```
}

.ConvertUnitNameToMultiplier <- function(unitName) {
# .ConvertUnitNameToMultiplier is a utility function for
# converting the time unit c('second', 'millisecond',
# 'microsecond', 'nanosecond') to proportions of one second.
# .ConverUnitNameToMultiplier implements the following map:
#        'second'   |-->   1
#   'millisecond'   |-->   1000
#   'microsecond'   |-->   1000000
#    'nanosecond'   |-->   1000000000
  if (unitName == 'second') {
    multiplier <- 1
  } else if (unitName == 'millisecond') {
    multiplier <- 1000
  } else if (unitName == 'microsecond') {
    multiplier <- 1000000
  } else if (unitName == 'nanosecond') {
    multiplier <- 1000000000
  }
  return(multiplier)
 }

.FindSampleAtTime <- function(waveform, timeOfSample,
                    timeUnit=(waveform %@% 'timeUnit')) {
# .FindSampleAtTime is a utility function for finding the
# index of the sample that occurs at a given time. Each
# sample point of the Waveform object is conceived of as
# being a half-open interval that is closed on the left and
# open on the right. The time value of each sample point
# is the value of the left boundary.
# Arguments:
#       waveform: A Waveform object.
#   timeOfSample: A numeric specifying the time of the
#                 sample whose index is to be found.
#       timeUnit: A character string specifying the unit of
#                 the timeValue argument.  Legal values:
#                 c('second', 'millisecond', 'microsecond',
#                 'nanosecond').
#                 Default is the unit of time for the time
#                 values of the Waveform object.
# Returns:
#   An integer specifying the index of the sample of the
#   Waveform object that occurs at the time specified by
#   the timeOfSample argument.
```

```
  # Create a vector of the sample times for the Waveform
  # object.
  sample.times <- .ComputeSampleTimes(waveform)

  # Convert the timeOfSample to the same unit as the sample
  # times.
  wave.time.unit <-
   .ConvertUnitNameToMultiplier(waveform %@% 'timeUnit')
  sample.time.unit <-
   .ConvertUnitNameToMultiplier(timeUnit)
  time.of.sample <-
   timeOfSample * (wave.time.unit / sample.time.unit)

  # Find the sample times that are prior or equal to the
  # time.of.sample.
  prior.sample.times <-
   which(sample.times <= time.of.sample)

  # The sample index is the last sample whose time is prior
  # or equal to the time.of.sample; hence, the index is
  # equal to the length of prior.sample.times
  sample.index <- length(prior.sample.times)

  # Return the index of the sample.
  return(sample.index)
}

.ComputeSampleTimes <- function(waveform) {
# .ComputeSampleTimes is a utility function for computing
# the time values of the sampled values of the waveform--
# i.e., the values that are not zeroes padded at the end of
# waveform in the case when waveform has been zero-padded.
# Arguments:
#   waveform: A Waveform object.
# Returns:
#   An integer specifying the number of sampled values in
#   waveform.

  # Find the start time of the waveform.
  start.time <- waveform %@% 'startTime'

  # Find the end time of the waveform.
  end.time <- waveform %@% 'endTime'
  # Find the number of samples in the waveform.
  sample.n <- waveform %@% 'N'

  # Create a vector of sample times from the start time, end
  # time and number of samples.
  sample.times <-
   seq(from=start.time, to=end.time, length.out=sample.n)

  # Return the vector of sample times.
  return(sample.times)
}

###########################
#  Object initialization  #
###########################

Waveform <- function(...) {
# Waveform is a generic function for creating a Waveform
# object.
# methods available in Waveform package:
#   Waveform.audioSample
#   Waveform.character
  UseMethod('Waveform')
}

Waveform.audioSample <-
function(audioSample, startTime=0, timeUnit='second') {
# Waveform.audioSample is a method for initializing a
# Waveform object from an audioSample object.  The
# audioSample class is defined in the 'audio' package.
# Arguments:
#   audioSample: An audioSample object.
#     startTime: A numeric that specifies the time of the
#                first sampled value of the audioSample
#                object.  Default is 0.
#      timeUnit: A character string specifying the unit of
#                measurement for startTime and endTime.
#                Legal values: c('second', 'millisecond',
#                   'microsecond', 'nanosecond').
#                Default is 'second'.
# Returns:
#   A Waveform object, which is a numeric vector whose
#   values represent the sampled values of the waveform,
#   augmented with the following attributes:
#        bitRate: The bitrate of the sampled waveform.
#     sampleRate: The sampling rate of the sampled waveform.
#   samplePeriod: The sampling period of the sampled
#                 waveform.
#              N: The number of samples in the waveform,
#                 excluding those values that are added to
#                 the waveform for zero-padding.
#      startTime: The time at which the first value of the
#                 waveform was sampled.
#        endTime: The time at which the last value of the
#                  waveform was sampled.
#       duration: The duration of the waveform.
#       timeUnit: The unit of measurement for startTime,
#                 endTime, and duration.

  # The audioSample object is a numeric vector that has a
  # 'bits' attribute and a 'rate' attribute for the bit rate and
  # sampling rate, respectively.
  waveform <- as.numeric(audioSample)

  # Set the 'bitRate' attribute of waveform to the value of
  # the 'bits' attribute of audioSample.
  waveform %@% 'bitRate' <- audioSample %@% 'bits'

  # Set the 'sampleRate' attribute of waveform to the value
  # of the 'rate' attribute of audioSample.
  waveform %@% 'sampleRate' <- audioSample %@% 'rate'

  # Set the 'samplePeriod' attribute of waveform.
  waveform %@% 'samplePeriod' <-
    1 / (waveform %@% 'sampleRate')

  # Set the 'N' (number of samples attribute of waveform.
  waveform %@% 'N' <- length(waveform)

  # Set the 'startTime' attribute of waveform from the
  # startTime argument.
  waveform %@% 'startTime' <- startTime

  # Set the 'endTime' attribute of waveform.
  time.lag.from.start <-
   ((waveform %@% 'N') - 1) / (waveform %@% 'sampleRate')
  waveform %@% 'endTime' <-
   time.lag.from.start + (waveform %@% 'startTime')

  # Set the 'duration' attribute of waveform.  The
  # (waveform %@% 'samplePeriod')  term is added in the
  # calculation below because each sampled point is a
  # treated as a semi-open interval (closed on the left,
  # open on the right) of duration equal to one sample
  # period.
  sampled.time.range <-
   (waveform %@% 'endTime') - (waveform %@% 'startTime')
  waveform %@% 'duration' <-
   sampled.time.range + (waveform %@% 'samplePeriod')

  # Set the 'timeUnit' attribute of waveform from the
  # timeUnit argument.
  waveform %@% 'timeUnit' <- timeUnit

  # Set the class of waveform.
  class(waveform) <- 'Waveform'

  # Return the Waveform object.
  return(waveform)
}

Waveform.character <-
function(waveFilepath, startTime=0, timeUnit='second') {
# Waveform.character is a method for initializing a Waveform
# object from the file path of a .wav file.
# Arguments:
#   waveFilepath: A character string specifying either the
#                 absolute or relative file path of a .wav
#                 file.
#      startTime: A numeric that specifies the time of the
#                 first sampled value of the waveform
#                 pointed to by waveFilepath.  Default is 0.
#       timeUnit: A character string specifying the unit of
#                 measurement for startTime and endTime.
#                 Legal values: c('second', 'millisecond',
#                 'microsecond', 'nanosecond').
#                 Default is 'second'.
# Returns:
#   A Waveform object, which is a numeric vector whose
#   values represent the sampled values of the waveform,
#   augmented with the following attributes:
#        bitRate: The bitrate of the sampled waveform.
#     sampleRate: The sampling rate of the sampled waveform.
#   samplePeriod: The sampling period of the sampled
#                 waveform.
#      startTime: The time at which the first value of the
#                 waveform was sampled.
#        endTime: The time at which the last value of the
#                 waveform was sampled.
#       duration: The duration of the waveform.
#       timeUnit: The unit of measurement for startTime,
#                 endTime, and duration.

  # Create an audioSample object by loading the wave file
  # pointed to by waveFilepath.
  audioSample <- load.wave(waveFilepath)

  # Dispatch the Waveform.audioSample method.
  Waveform(audioSample, startTime, timeUnit)
}

######################################
#  Methods to R's generic functions  #
######################################

plot.Waveform <- function(waveform, xAxisUnit='millisecond',
   type='l', col='orange',
   xlab=sprintf('Time (%s)', xAxisUnit), ylab='', ...) {
# plot.Waveform is a method for visualizing a Waveform
# object.
# Arguments:
#   waveform: A Waveform object.
#  xAxisUnit: A character string specifying the unit of the
#             time points plotted along the x-axis. Legal
#             values: c('second', 'millisecond',
#            'microsecond', 'nanosecond').
#             Default is 'millisecond'.
#       type: The type of line used to plot the values of
#             waveform. This value is passed to the
#             graphical parameter 'type'.
#        col: The color of the line used to plot the values
#             of waveform. This value is passed to the
#             graphical parameter 'col'.
#       xlab: The label on the x-axis.  This value is passed
#             to the graphical parameter 'xlab'. Default is
#             'Time (<xAxisUnit>)', where <xAxisUnit> is
#             replaced by the value of the xAxisUnit
#             argument.
#       ylab: The label on the y-axis.  This value is passed
#             to the graphical parameter 'ylab'.  Default is to
#             have no label.
#        ...: Other graphical parameters.
# Returns:
#   A plot of the Waveform object.

  # Make a vector of the time points at which the waveform's
  # samples occur.
  sample.times <- .ComputeSampleTimes(waveform)

  # Convert the unit of sample.times.
  wave.time.unit <-
     .ConvertUnitNameToMultiplier(waveform %@% 'timeUnit')
  x.axis.unit <- .ConvertUnitNameToMultiplier(xAxisUnit)
  sample.times <-
     sample.times * (x.axis.unit / wave.time.unit)

  # Grab just the sampled values of the waveform, excluding
  # the zero-padded values.
  wave.values <- as.numeric(waveform)
  sample.values <- wave.values[1:(waveform %@% 'N')]

  # Plot the Waveform object.
  plot(x=sample.times, y=sample.values,
       type=type, col=col, xlab=xlab, ylab=ylab, ...)
}

print.Waveform <- function(waveform) {
# print.Waveform is a method for reporting the attributes
# and visualizing a Waveform object.
# Arguments:
#   waveform: A Waveform object.
# Returns:
#   A report of the attributes of the waveform are printed
#   to the screen and a plot of the waveform is created.

  # Print the attributes of the Waveform object.
  message(sprintf(
     'Sampling rate:     %.2f', waveform %@% 'sampleRate'))
  message(sprintf(
     'Bit rate:          %d', waveform %@% 'bitRate'))
  message(sprintf(
     'Number of samples: %d', waveform %@% 'N'))
  message(sprintf(
     'Padded to:         %d', length(waveform)))
  message()
  message(sprintf(
     'Start time:        %f', waveform %@% 'startTime'))
  message(sprintf(
     'End time:          %f', waveform %@% 'endTime'))
  message(sprintf(
     'Duration:          %f', waveform %@% 'duration'))
  message(sprintf(
     'Time unit:         %s', waveform %@% 'timeUnit'))

  # Visualize the Waveform object.
  plot(waveform)
}

################################################
#  New generic functions and Waveform methods  #
################################################

FirstDifference <- function(...) {
  UseMethod('FirstDifference')
}

FirstDifference.Waveform <-
   function(waveform, coefficient=1) {
  # Make a delayed copy of the waveform that is scaled by
  # the coefficient.
  delayed.and.scaled <-
     c(0, waveform[1:(length(waveform)-1)]) * coefficient

  # Subtract the delayed and scaled copy from the waveform.
  preemphed.wave <- waveform - delayed.and.scaled

  # Return the pre-emphasized waveform.
  return(preemphed.wave)
}

TimeSlice <- function(...) {
# TimeSlice is a generic function for slicing a portion of a
# time series-like object according to time values, rather
# than indices.
# Methods available in Waveform package:
#   TimeSlice.Waveform
  UseMethod('TimeSlice')
}

TimeSlice.Waveform <- function(waveform, sliceFrom, sliceTo,
      centered=FALSE, duration,
      sliceUnit=(waveform %@% 'timeUnit')) {
# TimeSlice.Waveform is a method for slicing a portion of a
# Waveform object according to time values, rather than
# indices.
# Arguments:
#    waveform: A Waveform object.
#   sliceFrom: A numeric specifying the starting time of
#              the sliced portion of the Waveform object.
#     sliceTo: A numeric specifying the end time of the
#              sliced portion of the Waveform object.
#    centered: A boolean value.  If FALSE, then the values
#              of the sliceFrom and the sliceTo arguments
#              are used, and the duration argument is
#              ignored. If TRUE, then the duration argument
#              is used and the center portion of that
#              duration is sliced.
#    duration: A numeric specifying the duration of the
#              portion to be sliced if centered=TRUE.
#   sliceUnit: A character string specifying the unit of
#              time used to specify the sliceFrom and
#              sliceTo times. Legal values: c('second',
#              'millisecond', 'microsecond', 'nanosecond')
#              Default is the same time unit as the Waveform
#              object.
# Returns:
#   The portion of the Waveform object that falls between
#   the sliceFrom and sliceTo times, or a center portion of
#   the Waveform object if centered=TRUE.  If the Waveform
#   object had been zero-padded, then the zero-padding is
#   not appended to the sliced portion of the Waveform
#   object.

  # If the sliced portion is determined by sliceFrom and
  # sliceTo...
  if (! centered) {
    # Find the sample that occurs at the sliceFrom time.
    slice.from.index <- .FindSampleAtTime(waveform,
       timeOfSample=sliceFrom, timeUnit=sliceUnit)

    # Find the sample that occurs at the sliceTo time.
    slice.to.index <- .FindSampleAtTime(waveform,
       timeOfSample=sliceTo, timeUnit=sliceUnit)

    # Slice the waveform using the slice.from and slice.to
    # indices.
    sliced.wave <- as.numeric(waveform)
    sliced.wave <-
       sliced.wave[slice.from.index:slice.to.index]
  } else {
  # If the sliced portion is taken from the center of the
  # waveform...
    # Compute the time of the midpoint of the waveform, in
    # waveform time units.
    wave.midpoint <- waveform %@% 'startTime' +
       ((waveform %@% 'duration') / 2)

    # Convert wave.midpoint from waveform time units to the
    # time units in which the slice duration is specified.
    wave.unit.factor <- .
       ConvertUnitNameToMultiplier(waveform %@% 'timeUnit')
    slice.unit.factor <-
       .ConvertUnitNameToMultiplier(sliceUnit)
    conversion.factor <-
       slice.unit.factor / wave.unit.factor
    wave.midpoint <- wave.midpoint * conversion.factor

    # Compute the time at the beginning of the sliced
    # portion.
    slice.from.time <- wave.midpoint - (duration / 2)

    # Find the sample that occurs at slice.from.time.
    slice.from.index <- .FindSampleAtTime(waveform,
       timeUnit=sliceUnit, timeOfSample=slice.from.time)

    # Compute the time at the end of the sliced portion.
    slice.to.time <- wave.midpoint + (duration / 2)

    # Find the sample that occurs at slice.to.time.
    slice.to.index <- .FindSampleAtTime(waveform,
       timeOfSample=slice.to.time, timeUnit=sliceUnit)

    # Slice the waveform using the slice.from and
    # slice.to indices.
    sliced.wave <- as.numeric(waveform)
    sliced.wave <-
       sliced.wave[slice.from.index:slice.to.index]
  }

  # Copy the attributes of waveform over to those of
  # sliced.waveform.
  attributes(sliced.wave) <- attributes(waveform)

  # Update the 'startTime', 'endTime', 'duration', and 'N'
  # attributes of sliced.wave.
  # First, create a vector of the sample times for the
  # unsliced Waveform object.
  sample.times <- .ComputeSampleTimes(waveform)
  # Second, set the 'startTime' attribute of sliced.wave to
  # the time of the first sliced sample.
  sliced.wave %@% 'startTime' <-
     sample.times[slice.from.index]
  # Third, set the 'endTime' attribute of sliced.wave to the
  # time of the last sliced sample.
  sliced.wave %@% 'endTime' <- sample.times[slice.to.index]
  # Fourth, calculate the duration of sliced.wave.
  sliced.range <- (sliced.wave %@% 'endTime') -
     (sliced.wave %@% 'startTime')
  sliced.wave %@% 'duration' <- sliced.range +
     (sliced.wave %@% 'samplePeriod')
  # Lastly, update the 'N' attribute of sliced.waveform.
  sliced.wave %@% 'N' <- length(sliced.wave)
  # Return the sliced waveform.
  return(sliced.wave)
}

Zeropad <- function(...) {
# Zeropad is a generic function for padding zeroes to the
# end of a time series-like object.
# Methods available in Waveform package:
#   Zeropad.Waveform
  UseMethod('Zeropad')
}

Zeropad.Waveform <-
   function(waveform, lengthOut=(waveform %@% 'sampleRate')) {
# Zeropad.Waveform is a method for padding zeroes to the
# end of a Waveform object.
# Arguments:
#    waveform: A Waveform object.
#   lengthOut: An integer specifying the length of the
#              Waveform object after if has been padded with
#              zeroes.  Default is to pad the waveform to
#              the length equal to its sampling rate.
# Returns:
```

```
#   A Waveform object that is identical to the original
#   Waveform object, but with zeroes added to the end of it.

  # Check that the lengthOut of the padded waveform is
  # greater than the number of sampled values in the
  # waveform.
  if ((waveform %@% 'N') < lengthOut) {
    # If so, pad the waveform.
    # First, make a copy of just the sampled values of the
    # waveform.
    wave.values <- as.numeric(waveform)
    sample.values <- wave.values[1:(waveform %@% 'N')]
    # Second, create a vector of 0's to pad to the end of
    # the sampled values.
    num.zeroes.to.pad <- lengthOut - (waveform %@% 'N')
    zeroes.to.pad <- rep(0, times=num.zeroes.to.pad)
    # Third, pad the zeroes to the sampled values.
    padded.waveform <- c(sample.values, zeroes.to.pad)
    # Lastly, copy the attributes of the Waveform object
    # over to the padded waveform.
    attributes(padded.waveform) <- attributes(waveform)
  } else {
    # If the number of sampled values is greater than the
    # length that the waveform should be padded to, then
    # it cannot be padded.
    padded.waveform <- waveform
    # Print an error message.
    message(
       'You must pad the waveform to a length that is')
    message(
       'number of sampled values in the waveform.')
    message()
    message(sprintf(
       'Number of sampled values: %d', waveform %@% 'N'))
  }

  # Return the padded waveform.
  return(padded.waveform)
}
```

B Tapers.r

```
# Author:       Patrick Reidy
# Affiliations: The Ohio State University
#                Department of Linguistics
#                www.ling.ohio-state.edu
#                Learning To Talk
#                www.learningtotalk.org
# Email:         reidy@ling.ohio-state.edu
# Mail:          24A Oxley Hall
#                1712 Neil Ave.
#                Columbus, OH 43210-1298
# License:       GPL-3

#######################
#  Utility functions  #
#######################

`%@%` <- function(...) {
# %@% is a generic function for getting the value of an
# object’s attribute(s).
# Methods available in the Waveform package:
#   %@%.default
  UseMethod(’%@%’)
}

`%@%.default` <- function(object, attribute) {
# %@%.default is the default method for getting the value
# of an object’s attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
# Returns:
#   The value of the R object in the attribute slot of
#   object.
  attr(object, attribute)
}

`%@%<-` <- function(...) {
# %@%<- is a generic function for setting the value of an
# object’s attribute(s).
# Methods available in the Waveform package:
#   %@%<-.default
  UseMethod(’%@%<-’)
}

`%@%<-.default` <- function(object, attribute, value) {
# %@%<-.default is the default method for setting the value
# of an object's attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
#       value: Any R object.
# Returns:
#   Nothing. Instead, the value of object's attribute slot
#   is changed to value.
  `attr<-`(object, attribute, value)
}

###########################
#  Hamming taper methods  #
###########################

Hamming <- function(...) {
  UseMethod('Hamming')
}

Hamming.Waveform <- function(waveform) {
  # Get the number of samples in the Waveform object.
  num.of.samples <- waveform %@% 'N'

  # Generate the sequence of indices for the samples of the
  # Waveform object, that is a sequence of integers from 0
  # to (num.of.samples - 1).
  n.values <- seq(from=0, to=(num.of.samples -1 ))

  # Compute the values of the Hamming window from the
  # sequence of n values.
  hamming.values <- 0.54 -
     (0.46 * cos((2*pi*n.values) / (num.of.samples - 1)))

  # Pad the values of the Hamming window with the same
  # number of 0's that pad the Waveform object.
  zero.pad <-
     rep(0, times=(length(waveform) - num.of.samples))
  hamming.values <- c(hamming.values, zero.pad)

  # Multiply the Waveform object pointwise by the
  # zero-padded Hamming window.
  windowed.wave <- waveform * hamming.values

  # Set the attributes of the windowed waveform.
  attributes(windowed.wave) <- attributes(waveform)

  # Set an attribute to record how the waveform was
  # windowed.
  windowed.wave %@% 'taper' <- 'Hamming'

  # Return the windowed waveform.
  return(windowed.wave)
}
```

C `Periodogram.r`

```
# Author:        Patrick Reidy
# Affiliations: The Ohio State University
#               Department of Linguistics
#               www.ling.ohio-state.edu
#               Learning To Talk
#               www.learningtotalk.org
# Email:        reidy@ling.ohio-state.edu
# Mail:         24A Oxley Hall
#               1712 Neil Ave.
#               Columbus, OH 43210-1298
# License:      GPL-3

#######################
#  Utility functions  #
#######################

`%@%` <- function(...) {
# %@% is a generic function for getting the value of an
# object's attribute(s).
# Methods available in the Waveform package:
#   %@%.default
  UseMethod('%@%')
}

`%@%.default` <- function(object, attribute) {
# %@%.default is the default method for getting the value of
# an object's attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
# Returns:
```

```
#   The value of the R object in the attribute slot of
#   object.
  attr(object, attribute)
}

`%@%<-` <- function(...) {
# %@%<- is a generic function for setting the value of an
# object's attribute(s).
# Methods available in the Waveform package:
#   %@%<-.default
  UseMethod('%@%<-')
}

`%@%<-.default` <- function(object, attribute, value) {
# %@%<-.default is the default method for setting the value of
# an object's attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
#       value: Any R object.
# Returns:
#   Nothing. Instead, the value of object's attribute slot
#   is changed to value.
  `attr<-`(object, attribute, value)
}

###########################
#  Object initialization  #
###########################

Periodogram <- function(...) {
# Periodogram is a generic function for computing the
# ordinate values of the periodogram of a time series-like
# object.
  UseMethod('Periodogram')
}

Periodogram.Waveform <- function(waveform) {
# Periodogram.Waveform is a method for computing the
# ordinate values of the periodogram of a Waveform object.
# Arguments:
#   waveform: A Waveform object.
# Returns:
#   A Periodogram object, comprising the ordinate values of
#   the periodogram of the Waveform object.  If N is the
#   number of sampled values in the waveform x, then the
#   periodogram I of x is defined by
#     I(w_j) = (1/N) * |d(w_j)|^2,
#   where w_j = j/N is the j^th Fourier frequency (for j =
#   0, ..., N-1) and d(w_j) is the ordinate value of the
#   discrete Fourier transform at w_j, which is defined by
#     d(w_j) = \sum_{t=0}^{N-1} x_t * exp{-2 \pi i w_j t}.
#   A Periodogram object, furthermore, comprises the
#   following attributes:
#         nyquist: The Nyquist frequency of the Waveform
#                  object.
#               N: The number of sampled values in the
#                  Waveform object.
#        binWidth: The width of each frequency bin in the
#                  Periodogram object.
#    fourierFreqs: The hertz values of the Fourier
#                  frequencies.

  # Compute the ordinate values of the periodogram:
  # First, compute the ordinate values of the power spectrum.
  power.spectrum <- abs(fft(waveform))^2
  # Then, scale the power spectrum to get the periodogram.
  periodogram <- (1 / (waveform %@% 'N')) * power.spectrum

  # Keep only the ordinate values that lie on the upper half
  # of the unit circle.  That is, the ordinate values for
  # those frequencies that fall within [0, nyquist).
  nyquist.index <- floor(length(periodogram) / 2)
  periodogram <- periodogram[1:nyquist.index]

  # Set the 'nyquist' attribute of the periodogram, which is
  # equal to half the sampling rate of the Waveform object.
  periodogram %@% 'nyquist' <-
     (waveform %@% 'sampleRate') / 2

  # Set the 'N' attribute of the periodogram, which is equal
  # to the number of sampled values of the Waveform object.
  periodogram %@% 'N' <- waveform %@% 'N'

  # Set the 'binWidth' attribute of the periodogram, which
  # is equal to the sampling rate of the Waveform object,
  # divided by the number of values in the Waveform object
  # (including both sampled and zero-padded values).
  periodogram %@% 'binWidth' <-
```

```
     (waveform %@% 'sampleRate') / length(waveform)

  # Set the 'fourierFreqs' attribute of the periodogram.
  periodogram %@% 'fourierFreqs' <-
     seq(from=0, length.out=nyquist.index,
     by=(periodogram %@% 'binWidth'))

  # Set the class of the periodogram.
  class(periodogram) <-
     c('Periodogram', 'Spectrum', 'numeric')

  # Return the periodogram.
  return(periodogram)
}
```

D Multitaper.r

```
# Author:       Patrick Reidy
# Affiliations: The Ohio State University
#               Department of Linguistics
#               www.ling.ohio-state.edu
#               Learning To Talk
#               www.learningtotalk.org
# Email:        reidy@ling.ohio-state.edu
# Mail:         24A Oxley Hall
#               1712 Neil Ave.
#               Columbus, OH 43210-1298
# License:      GPL-3

# The Multitaper package depends on Karim Rahim's multitaper
# package.
library('multitaper', quietly=TRUE)

#######################
#  Utility functions  #
#######################

`%@%` <- function(...) {
# %@% is a generic function for getting the value of an
# object's attribute(s).
# Methods available in the Waveform package:
#   %@%.default
  UseMethod('%@%')
}
```

```
`%@%.default` <- function(object, attribute) {
# %@%.default is the default method for getting the value
# of an object’s attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
# Returns:
#   The value of the R object in the attribute slot of
#    object.
  attr(object, attribute)
}

`%@%<-` <- function(...) {
# %@%<- is a generic function for setting the value of an
# object’s attribute(s).
# Methods available in the Waveform package:
#   %@%<-.default
  UseMethod(’%@%<-’)
}

`%@%<-.default` <- function(object, attribute, value) {
# %@%<-.default is the default method for setting the value
# of an object’s attribute(s).
# Arguments:
#      object: Any R object.
#   attribute: A character string that names an attribute
#              slot of object.
#       value: Any R object.
# Returns:
#   Nothing.  Instead, the value of object’s attribute slot
# is changed to value.
  `attr<-`(object, attribute, value)
}

.ColumnMultiply <- function(numVector, numMatrix) {
# .ColumnMultiply is a utility function for multiplying each
# column of a matrix by a vector.
# Arguments:
#   vect: A numeric vector.
#   matr: A numeric matrix.
# Returns:
#   A matrix that has the same dimensions as matr.  Each
#   column is equal to the corresponding column of matr
#   multiplied by vect.

  # Multiply each column of matr by vect.
  new.matrix <- apply(numMatrix, 2, `*`, numVector)

  # Return the new matrix.
  return(new.matrix)
}

.NormalizeSum <- function(numVector, normalizeTo=1) {
# .NormalizeSum is a utility function for normalizing the
# values of a numeric vector so that they sum to a
# predetermined number.
# Arguments:
#     numVector: A numeric vector.
#   normalizeTo: A numeric vector of length 1.
# Returns:
#   The numeric vector that results from scaling the
#   elements of numVector so that they sum to sumTo.

  # Determine the scale factor.
  scale.factor <- normalizeTo / sum(numVector)

  # Multiply by the scale factor.
  normalized.vector <- numVector * scale.factor

  # Return the normalized vector.
  return(normalized.vector)
}

###########################
#  Object initialization  #
###########################

Multitaper <- function(...) {
# Multitaper is a generic function for computing the
# multitaper spectrum of a time series-like object.
  UseMethod('Multitaper')
}

Multitaper.Waveform <- function(waveform, k=(2*nw), nw=4) {
# Multitaper.Waveform is a method for computing the
# multitaper spectrum of a Waveform object.
# Arguments:
#   waveform: A Waveform object.
#          k: An integer specifying the number of DPSS
#             tapers to use in the computation of the
#             multitaper spectrum.  Default value is
#             k = (2 * nw). Since k and nw are constrained
#             to satisfy k <= 2*nw, the default value is the
#             maximum number of tapers that should be used
#             given a fixed value of nw.
#         nw: An integer specifying the time-bandwidth
#             parameter used to generate the DPSS tapers.
# Returns:
#   The k^th-order multitaper spectrum of the waveform,
#   computed using DPSS tapers generated using the
#   time-bandwidth parameter nw.  A Multitaper object,
#   furthermore, has the following attributes:
#         nyquist: The Nyquist frequency of the Waveform
#                  object.
#               N: The number of sampled values in the
#                 Waveform object.
#        binWidth: The width of each frequency bin in the
#                 Multitaper object.
#    fourierFreqs: The hertz values of the Fourier
#                 frequencies.
#               k: The number of DPSS tapers (equivalently,
#                   eigenspectra) used in the computation of
#                   the multitaper spectrum.
#              nw: The time-bandwidth parameter used to
#                  generate the DPSS tapers.

  # Generate the DPSS tapers using the dpss function from
  # the multitaper package.  dpss creates a named list whose
  # 'v' element is a matrix, each column of which is a DPSS
  # taper.
  dpss.taper.matrix <-
     dpss(n=(waveform %@% 'N'), k=k, nw=nw)$v

  # Zero-pad the DPSS tapers to the length of the waveform,
  # since the waveform is zero-padded by the difference
  # between length(waveform) and (waveform %@% 'N'):
  # First, determine how many zeroes were padded on the end
  # of the waveform.
  pad.length <- length(waveform) - (waveform %@% 'N')
  # Second, create a matrix of 0's that has pad.length rows
  # and k columns.
  zeropad.matrix <- matrix(data=0, nrow=pad.length, ncol=k)
  # Lastly, row bind the zeropad.matrix to the bottom of the
  # dpss.taper.matrix.
  dpss.taper.matrix <- r
     bind(dpss.taper.matrix, zeropad.matrix)

  # Make k tapered copies of the sampled waveform by
  # windowing it by each DPSS taper, using the
  # .ColumnMultiply function.
  tapered.wave.matrix <-
     .ColumnMultiply(waveform, dpss.taper.matrix)

  # Compute the k eigenspectra of the waveform.  The k^th
  # "eigenspectrum" of the waveform is the periodogram of
  # the waveform after it has been windowed by the k^th
  # DPSS taper.
  eigenspectra <- abs(fft(tapered.wave.matrix))^2

  # For each eigenspectrum, keep only the ordinate values
  # for the frequencies in the [0, nyquist) range.
  nyquist.index <- floor(length(waveform) / 2)
  nyquist.eigenspectra <- eigenspectra[1:nyquist.index, ]

  # Compute the k^th order multitaper spectrum by averaging
  # the k eigenspectra, pointwise.
  if (k == 1) {
    multitaper <- nyquist.eigenspectra
  } else {
    multitaper <- rowMeans(nyquist.eigenspectra)
  }

  # Set the 'nyquist' attribute of the multitaper spectrum,
  # which is equal to half the sampling rate of the
  # Waveform object.
  multitaper %@% 'nyquist' <-
     (waveform %@% 'sampleRate') / 2

  # Set the 'N' attribute of the multitaper spectrum, which
  # is equal to the number of sampled values of the Waveform
  # object.
  multitaper %@% 'N' <- waveform %@% 'N'

  # Set the 'binWidth' attribute of the multitaper spectrum,
  # which is equal to the sampling rate of the Waveform
  # object, divided by the number of values in the Waveform
  # object (including both sampled and zero-padded values).
  multitaper %@% 'binWidth' <-
```

```
      (waveform %@% 'sampleRate') / length(wavefofrm)

   # Set the 'fourierFreqs' attribute of the multitaper
   # spectrum.
   multitaper %@% 'fourierFreqs' <-
      seq(from=0, length.out=nyquist.index,
      by=(multitaper %@% 'binWidth'))

   # Set the 'k' attribute of the multitaper spectrum.
   multitaper %@% 'k' <- k

   # Set the 'nw' attribute of the multitaper spectrum.
   multitaper %@% 'nw' <- nw

   # Set the class of the multitaper spectrum.
   class(multitaper) <- c('Multitaper', 'Spectrum', 'numeric')

   # Return the multitaper spectrum.
   return(multitaper)
}
```