

TECHNICAL PAPER

Time-Frequency Analysis Methods and Applications in SAS®

Last update: July 2024



Contents

Contents	2
Introduction	1
Preliminary Background	1
Convolution and Rotating Phasors	1
Fast Fourier Transform (FFT).....	2
Sampling.....	3
Hilbert Transform.....	4
Short-Time Fourier Transform	4
Methodology.....	5
UNWRAP Function	7
Methodology.....	7
INSTANTTFA Function	9
Methodology.....	9
Hilbert Method	9
First Moment Method	11
Hilbert-Huang Transform	11
Methodology.....	12
Empirical Mode Decomposition (EMD).....	12
Hilbert Spectrum	15
Peak Detection	16
Methodology.....	16
Analyzing EEG Signals	18
Instrument-Based Music Decomposition	22

Queen Bee Piping Example 1.....	30
Queen Bee Piping Example 2.....	39
Conclusion	42
References.....	43

Relevant Products and Releases

- SAS® Viya® 4
 - SAS® IML

Introduction

Digital signal processing (DSP) is the methodological and technical discipline that is concerned with manipulating, analyzing, and transforming signals through digital means. In DSP, signals are represented as discrete sequences of numerical values, which are subject to processing by using algorithms that are implemented on digital computing systems or specialized DSP hardware platforms. This field finds extensive application across diverse domains, including audio and speech processing, image and video processing, telecommunications, radar, sonar, biomedical engineering, and beyond. DSP facilitates crucial tasks such as noise reduction, signal enhancement, data compression, spectral analysis, modulation, demodulation, and various forms of signal synthesis, all conducted within the digital realm. Its methodologies encompass a wide spectrum of operations, ranging from basic filtering and convolution to complex Fourier analysis and sophisticated modulation techniques, enabling comprehensive manipulation and understanding of both the frequency and time domains. SAS offers many digital signal processing tools. This paper covers some of the tools that have been most recently added to SAS IML software ([SAS IML: Language Reference](#)), such as the TFSTFT function, UNWRAP function, INSTANTTFA function, EMD call, HHT call, HHTSPECTRUM call, PEAKLOC function, and PEAKINFO call.

The paper reviews the methodology behind these tools and presents real-world examples to demonstrate how to use them. The audience is data scientists and engineers who are looking to analyze high-frequency data by using signal processing techniques.

Preliminary Background

This section presents the information that you need in order to understand the subsequent sections of the paper.

Convolution and Rotating Phasors

The concept of convolution, which is useful later in this section, is described as a mathematical operation that combines two functions to form a third function by way of integrating the product of the two functions. It is represented as follows:

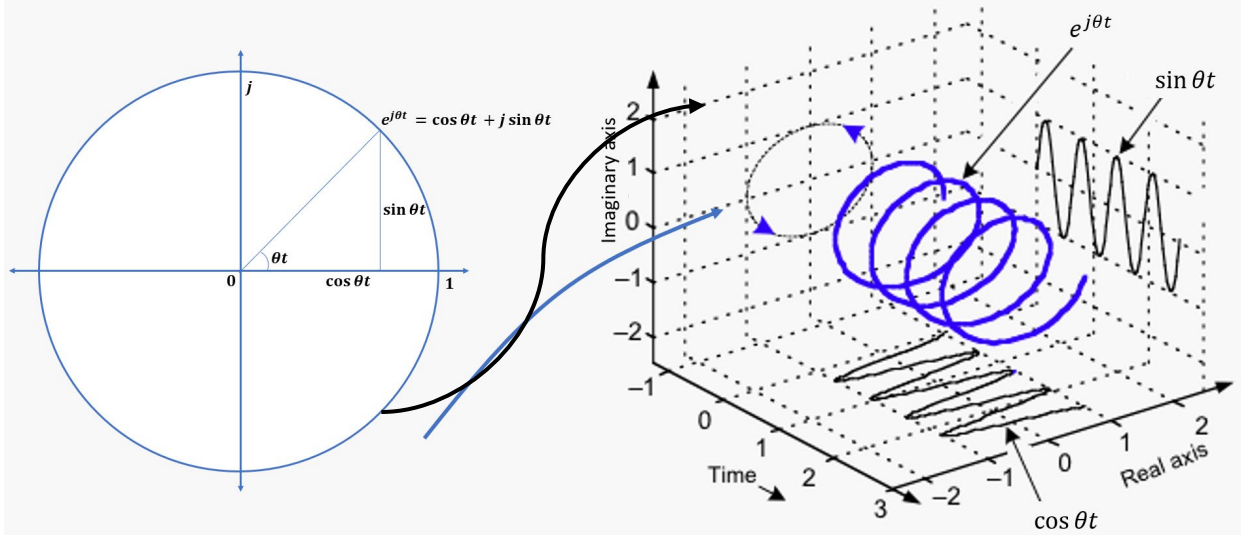
$$f(t) * g(t) = \int_{-\infty}^{\infty} f(t)g(t)dt$$

To explain how a function can go from the time domain to the frequency domain, we use Euler's formula, which states that a complex exponential for any real value t can be expressed as

$$e^{j\theta t} = \cos \theta t + j \sin \theta t$$

where $\theta = 2\pi\omega$ and ω represents frequency. From this point on, we refer to this exponential term as a three-dimensional phasor or complex rotating phasor. The left side of **Figure 1** illustrates Euler's formula by using polar coordinates. This formula is an important concept for convolution and the Fourier transform. As t increases, $e^{j\theta t}$ rotates clockwise in a circle with a radius of 1. This unit circle is shown as a projection on the right of the figure, where you see how this illustration conveys that $e^{j\theta t}$ is constructed from $\cos \theta t$ on the real and time axes and from $\sin \theta t$ on the imaginary and time axes.

Figure 1. Example of Using a Unit Circle to Illustrate Euler's Formula



Fast Fourier Transform (FFT)

To obtain the fast Fourier transform (FFT), convolution is performed using a function, $f(t)$, and using the rotating phasors that are made from sinusoids. This convolution allows the function $f(t)$ to be expressed as a set of sinusoids, which then easily translates to the frequency spectrum through the use of a spectrogram.

Then the Fourier transform, $F(\omega)$, of function $f(t)$ is given by the following equation:

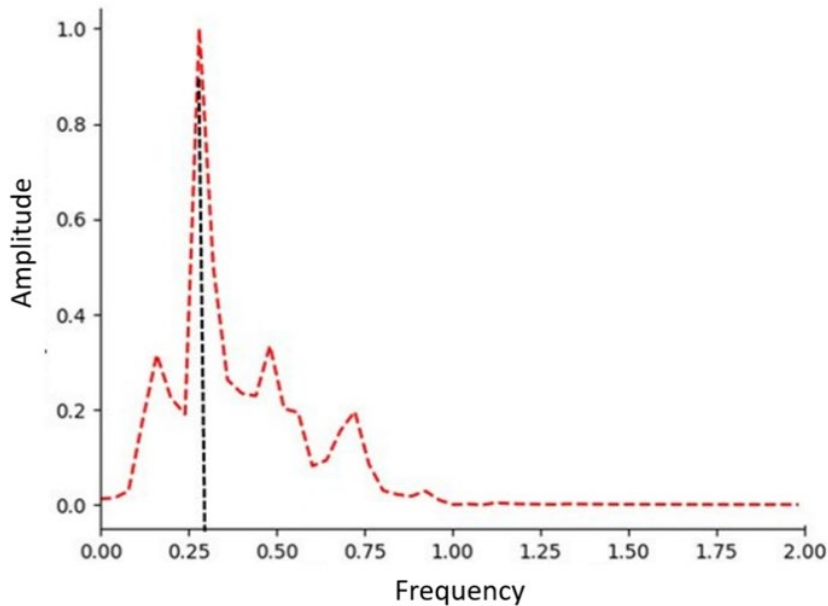
$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\theta t} dt$$

To change back to the time domain, we use the inverse Fourier transform, which is given by a similar equation:

$$f(t) = \int_{-\infty}^{\infty} F(t)e^{j\theta t} d\omega$$

The output of the Fourier transform is a set of complex numbers. You can also visualize the output by taking the magnitude squared, which represents the power present at each frequency. **Figure 2** shows an example of the power of the Fourier transform.

Figure 2. Illustration of Magnitude Found by the Fourier Transform



Sampling

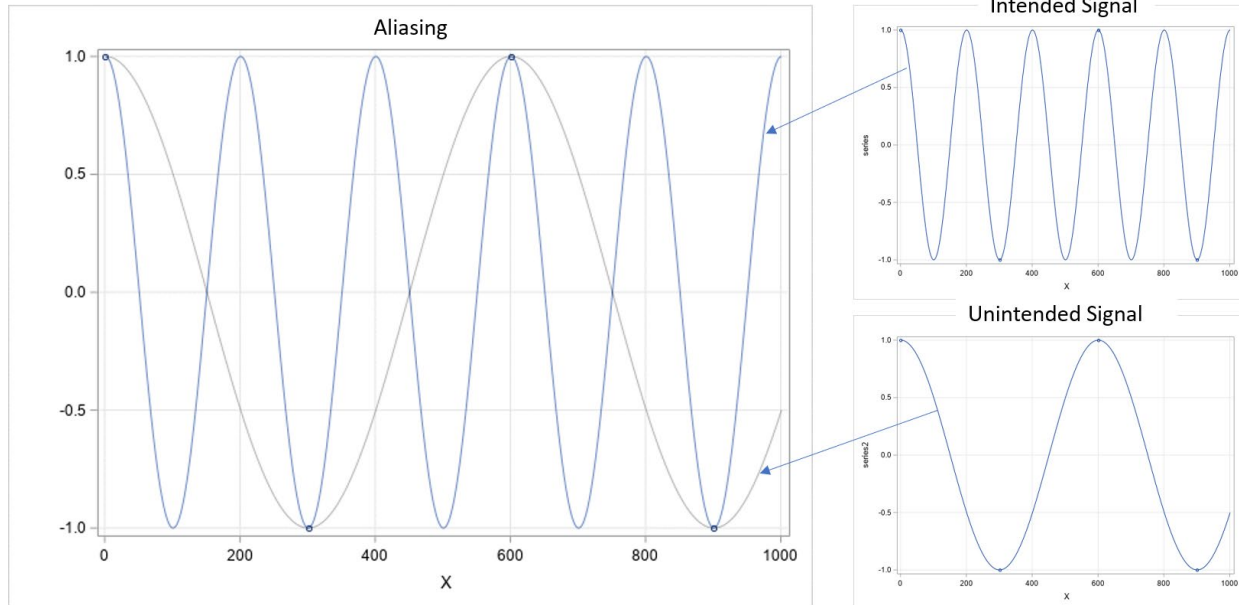
When you digitize an analog time series signal so that it can be processed in a digital space, assumptions about continuous time no longer apply. In the analog domain, you can assume that there are infinite time points between $x(t = 0)$ and $x(t = 1)$. But in the digitized or discrete domain, you are limited to working with a set of discrete sampled points.

When you convert to the digital domain, the number of evenly spaced discrete points that are sampled per second from an analog signal is called the sampling frequency. In digital signal processing, the sampling frequency is often measured in hertz (Hz). A higher sampling frequency provides better fidelity in representing the original signal, but it also requires more storage space and greater processing power. For example, if you convert an analog signal that is measured at 60 samples per second, the sampling frequency of the newly obtained digital signal is 60 Hz.

The range of frequencies from which information can be determined depends on the sampling frequency. For a sampling frequency f_s , the range of frequencies available is $[0, f_s/2]$, where $f_s/2$ is referred to as the Nyquist frequency (Grenander 1959). Sampling a signal below its Nyquist frequency allows for a signal to be distorted, and it might change the signal in unintended ways. This distortion is referred to as aliasing. Because all frequencies are mapped between 0 and the Nyquist frequency, high-frequency components that exceed the Nyquist frequency are changed to lower-frequency components.

Figure 3 shows an example of how a sinusoidal signal is transformed into another sinusoidal signal of a lower frequency as a result of insufficient sampling.

Figure 3. Example of Aliasing



Hilbert Transform

The Hilbert transform produces the analytic signal of the input time series vector. The instantaneous amplitude, phase, and frequency are then determined from that analytical signal. The analytical signal of an input signal $x(t)$ is defined as $z(t) = x(t) + i\hat{x}(t)$, where $\hat{x}(t)$ is the Hilbert transform of $x(t)$ and is defined as follows:

$$\hat{x}(t) = 2 \int_0^{\infty} X(f)e^{2\pi ift} df$$

where $X(f)$ is the Fourier transform of $x(t)$.

The analytical signal is a tool that is created to remove redundant negative frequencies from the Fourier spectrum of a real signal. The analytical signal can be preferable in some applications, because it can prevent unwanted cross-term artifacts from having both positive and negative frequencies, and in some cases it can reduce estimation biases. The Hilbert transform is used in the section [INSTANTTFA](#) Function.

Short-Time Fourier Transform

The first function that we delve into is the TFSTFT function; TFSTFT stands for time-frequency short-time Fourier transform. This function is designed to calculate the short-time Fourier transform of an input signal. The short-time Fourier transform (STFT) represents a methodological adaptation of the conventional Fourier transform that is specifically tailored for analysis within a moving-window framework. You can see the syntax of the TFSTFT function in the [SAS IML documentation](#).

Methodology

The short-term Fourier transform is the application of the Fourier transform to windowed data. The Fourier transform represents a time signal as a summation of sinusoidal functions. This transformation allows a signal to be changed from the time domain to the frequency domain.

The Fourier transform equation, shown in the section Fast Fourier Transform (FFT), essentially performs convolution with a time series function and a rotating phasor, $e^{-j\theta t}$, so that the function can be expressed as a set of sinusoids whose frequencies can easily be read in the frequency domain.

The short-term Fourier transform is a Fourier transform to which a moving window, $w(\tau)$, is applied. By applying a moving window, you can track how the frequency of a time series function changes through time.

The following equation is of the STFT. Here we see that the STFT is the integral of the original function, $f(t)$, with a moving time window, $w(t - \tau)$, and a rotating phasor, $e^{-j\theta t}$.

$$S(t, \omega) = \int_{-\infty}^{\infty} f(t)w(t - \tau)e^{-j\theta t} dt$$

The discrete version of the short-time Fourier transform for a time series $x[0], x[1], \dots, x[L - 1]$ and a window $w[0], \dots, w[m]$ is given by

$$\text{STFT}_x(n, f) = \sum_{k=0}^{m-1} x[n + k]w[k]e^{-j2\pi kf}$$

Let $k = \left\lfloor \frac{\text{series length} - \text{overlap}}{\text{window length} - \text{overlap}} \right\rfloor + 1$ and $S = \text{window length} - \text{overlap}$.

Then the STFT consists of the computation of $\text{STFT}_x(n, f)$ for $n = 0, S, 2S, \dots, kS$ and for

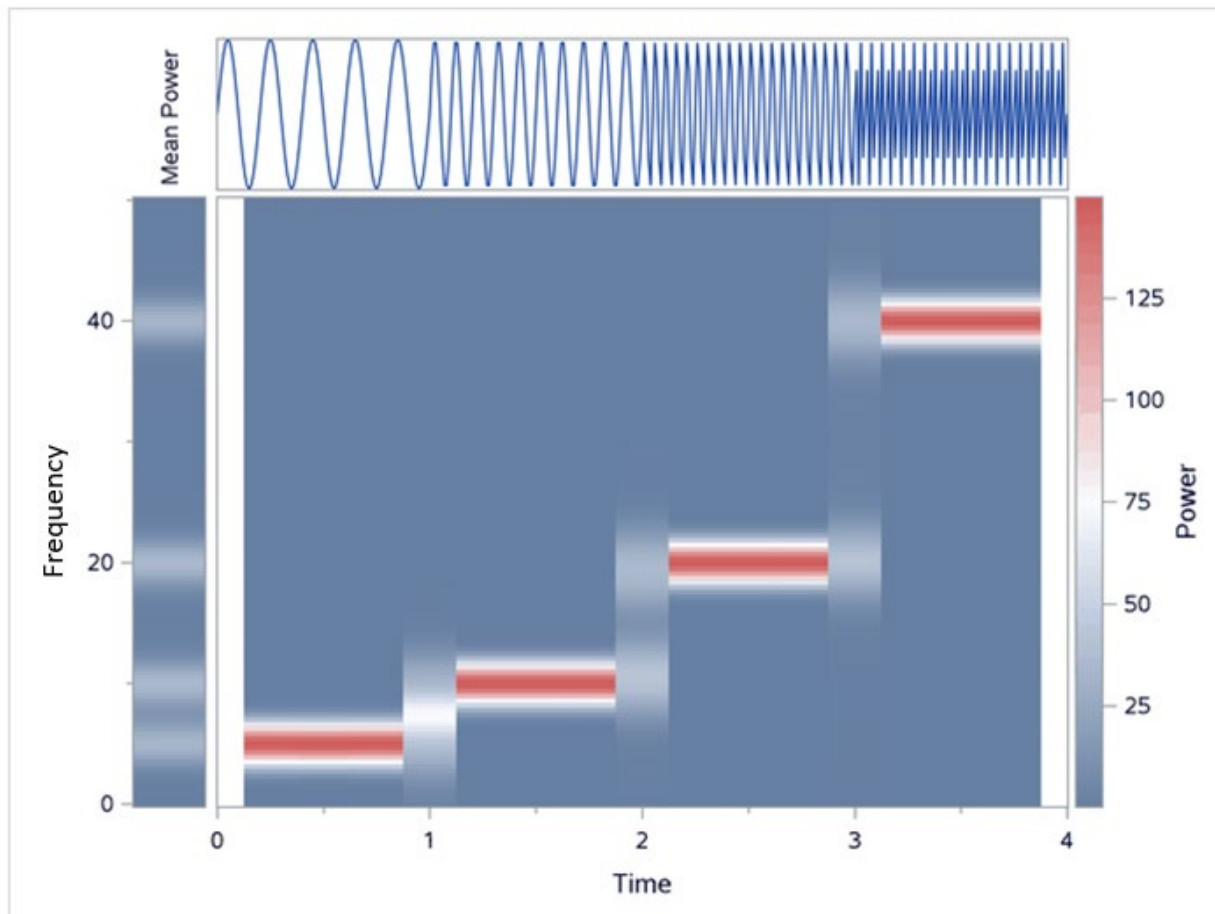
$$f = 0, \frac{1}{\text{fftlen}}, \dots, \frac{\text{fftlen} - 1}{\text{fftlen}}$$

where fftlen refers to the total number of frequency indices between 0 and the maximum frequency that the function can express, which is given by half the sampling frequency.

The range of frequencies from which information can be determined depends on the sampling frequency. For a sampling frequency f_s , the range of frequencies available is $[0, f_s/2]$, where $f_s/2$ is the Nyquist frequency (Grenander 1959).

Another way to think of the preceding equation is to understand that the STFT is essentially the Fourier transform that has been computed for every window $w[k]$. Thus, the output of the STFT is the same as the Fourier transform with the added dimension of time. **Figure 4** is an example of the output of the STFT that is plotted using SAS IML's [SPECTROGRAM](#) call. The spectrogram also includes mean power (at top) and the frequencies without a temporal component (at left). Unlike the image in **Figure 2**, which shows the output of the Fourier transform, the image in **Figure 4** represents power intensity on a color scale (at right).

Figure 4. Output of the SPECTROGRAM Call



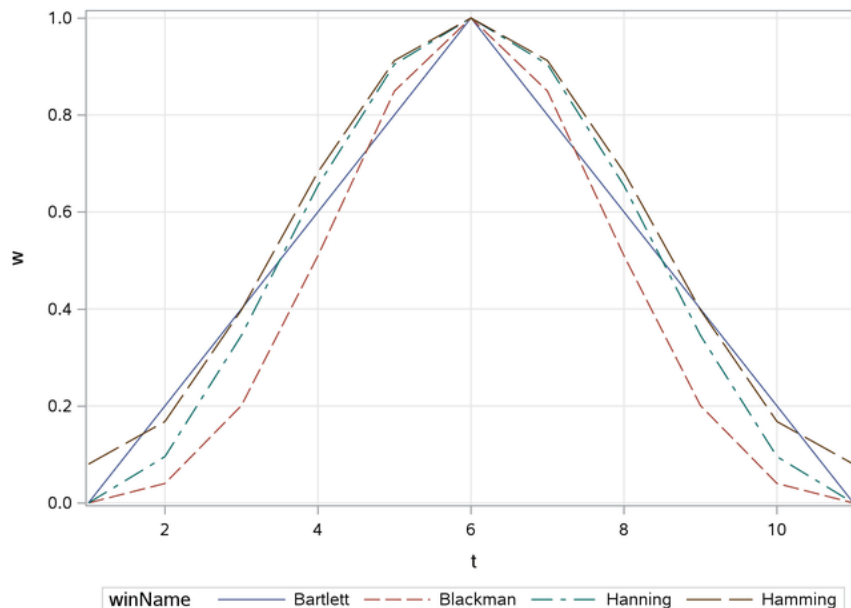
SAS IML's TFSTFT function offers the option to specify the *fftlen*, which refers to the total number of indexes in the range $[0, f_s/2]$. A higher *fftlen* value increases the resolution of the frequency at every window. The TFSTFT function restricts the *fftlen* value to be the same size as or larger than the specified moving time window, which is measured by window length (referred to as *window_len* in the SAS IML documentation).

A rectangular window has boundaries that change drastically as they go from 0 to 1 at the beginning of the window, then back down to 0 at the end of the window. Recall that the STFT includes a windowing function in its convolution. When you use a rectangular window, spectral leakage, which has a negative impact on output frequencies, is introduced at both the beginning and end of the segment (Nuttall 1981).

The TFSTFT function offers multiple alternative windows that help minimize the boundary effect of a rectangular window when you use the STFT. Each of these windows offers a unique way to attenuate frequencies at the window's boundaries, so choosing the correct window depends on the signal-specific application and the desired frequency results.

Figure 5 presents a depiction of these windows. (For more information about the windows that SAS IML offers, see the [TFWINDOW function documentation](#).)

Figure 5. The Various Windows SAS IML Offers through the TFWINDOW Function



For a real-world example of how the STFT is used, see Queen Bee Piping Example.

UNWRAP Function

The UNWRAP function takes the phase of a signal that is represented by a vector of angles and unwraps the phase when the difference between two consecutive angles exceeds a particular threshold. You can see the syntax of this function in the [UNWRAP function documentation](#). The phase of a real signal that is produced from the FFT (see Fast Fourier Transform (FFT)) is related to the start time of that signal. This phase can be understood as a signal's angles and can be mapped to a range of $-\pi$ to π . This type of phase is referred to as a wrapped phase. In some cases, such as finding instantaneous phase in frequency (see INSTANTTFA Function), a wrapped phase can produce undesired results. This problem is remedied with phase unwrapping. Phase unwrapping takes the phase angle and corrects it by adding multiples of $\pm 2\pi$. Mappings below a wrapped phase are detected when the difference between two consecutive angles exceeds the threshold. The default threshold is 2π .

Methodology

In digital signal processing (DSP), phase refers to the relative timing or displacement of one waveform with respect to another waveform. It characterizes the temporal alignment or misalignment between two signals at a particular frequency or over a range of frequencies.

More formally, phase can be understood as the angle component of a complex representation of a sinusoidal signal. Whereas the period of a signal can be described as the measure of time t , where that signal has completed an entire repetition, the phase is the measure of degrees or radians at any point in time, t .

Consider a simple sinusoidal function $A\sin(\theta)$, where θ is considered the phase. To break down exactly how the phase is determined, you can specify $\theta = \omega t$.

The unit circle is a tool that is often used to visualize how a phase angle changes. A full period is referred to as 2π and is represented as an entire rotation around the unit circle. An example of a unit circle is shown in **Figure 6**.

Phase unwrapping becomes useful whenever the phase exceeds 2π . The value of a periodic function is the same for every multiple of 2π . In other words, for the periodic function $f(x)$ and a scalar value x ,

$$f(t + 2\pi x) = f(t)$$

This equation leads the function to return a phase value between 0 and 2π , even if the phase has a correct value of over 2π . Phase unwrapping returns a value that reflects an angle that exceeds 2π .

For example, take a signal with a phase of $\frac{5\pi}{2}$, as shown in **Figure 7**. The phase that is represented on the unit circle is one full rotation plus a quarter rotation. After a full rotation, the phase repeats itself on the same unit circle; this repetition can cause the loss of important phase information in some applications. Without phase unwrapping, the phase output is $\frac{\pi}{2}$, but with phase unwrapping, the output includes the extra rotation on the unit circle.

Figure 6. Unit Circle Rotating Counterclockwise to Show Phase

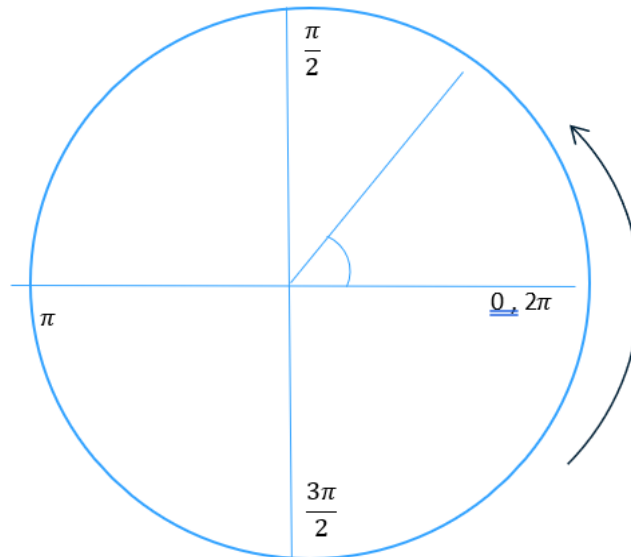
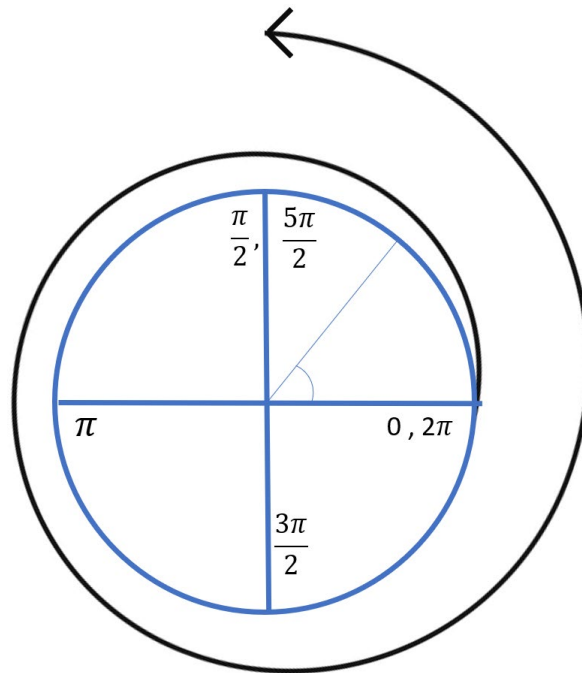


Figure 7. A Unit Circle with a Phase That Is More Than One Full Rotation



INSTANTTFA Function

The INSTANTTFA function calculates the instantaneous frequency, phase, and amplitude of a signal. The name of the function is shorthand for instantaneous time-frequency analysis. You can see its syntax in the [INSTANTTFA function documentation](#).

Methodology

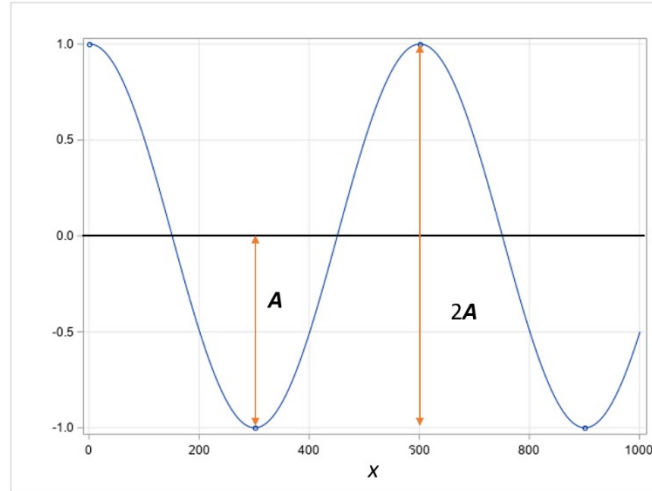
Hilbert Method

Recall the analytical signal from the earlier section Hilbert Transform. The analytical signal of an input signal $x(t)$ is defined as $z(t) = x(t) + i\hat{x}(t)$, where $\hat{x}(t)$ is the Hilbert transform of $x(t)$.

The analytical signal is used in the INSTANTTFA function to determine the instantaneous amplitude, phase, and frequency.

A time-frequency sinusoid can be represented by $A \sin(2\pi ft)$, where A is the amplitude. The amplitude is a nonnegative scalar value that is given by the distance between the X axis and the signal's peak. An example of the amplitude is shown in **Figure 8**.

Figure 8. Amplitude of a Sinusoid



The following equation calculates the amplitude, $\text{amp}(t)$, from the analytical signal:

$$\text{amp}(t) = \sqrt{x(t)^2 + \hat{x}(t)^2}$$

To calculate the amplitude in the discrete domain, you can use the discrete time equivalent to the preceding equation, where time is evenly sampled into N discrete points and the signal $x(t)$ is represented by $x[n]$, where $n = 0, 1, \dots, N - 1$. The equation is as follows:

$$\text{amp}[n] = \sqrt{x[n]^2 + \hat{x}[n]^2}$$

This equation works, because one property of an analytical signal is that the real part and the imaginary components must be orthogonal to one another. In other words,

$$\sum_{n=0}^{N-1} x[n]\hat{x}[n] = 0$$

The phase of a signal can be understood as the measure of the signal in degrees or radians at any point in time, t . The instantaneous phase of the analytical signal at time t , $\theta(t)$, is calculated using the ATAN2 function as follows:

$$\theta(t) = \text{atan2} \frac{\hat{x}(t)}{x(t)}$$

The discretized version of this equation is as follows:

$$\theta[n] = \text{atan2} \frac{\hat{x}[n]}{x[n]}$$

The INSTANTTFA function outputs the unwrapped version of $\theta[n]$. (See the [UNWRAP Function](#).)

Frequency is the number of cycles or repetitions that a signal completes per unit time. To determine the instantaneous frequency, f_H , of a signal by using the Hilbert method, you simply take the gradient of the phase, $\theta(t)$. Optionally, you can also multiply it by the scalar $f_s/2\pi$, where f_s is the sampling frequency, to get the frequency in hertz:

$$f_H = \frac{f_s}{2\pi} \frac{d\theta}{dt}$$

When you use the INSTANTTFA function, you can output amplitude, frequency, or phase, depending on the input specification. You can also obtain the instantaneous frequency by using the first moment method.

First Moment Method

Moments are statistical measurements that describe a set of characteristics of a function. Note that the first-, second-, third-, and fourth-order moments give the mean, variance, skewness, and kurtosis of a distribution, respectively.

The central moment of function $Z(f)$ is defined as follows:

$$\mu_m = \frac{\int_{-\infty}^{\infty} f^m |Z(f)|^2 df}{\int_{-\infty}^{\infty} |Z(f)|^2 df}$$

By setting $m = 1$ and setting $|Z(f)| = S_x(n, f)$, which is the frequency spectrum of the analytical signal from using the short-term Fourier transform, you can set up the function to find the first moment in the frequency. These substitutions provide μ_1 , the mean value of the frequencies from 0 to $f_s/2$:

$$\mu_1 = \frac{\int_0^{f_s/2} f S_x(n, f) df}{\int_0^{f_s/2} S_x(n, f) df}$$

When you use the INSTANTTFA function and specify the moment method, you can also specify the *winLen* argument. This argument enables you to adjust the segment that the INSTANTTFA function uses in its calculation of the STFT.

For a real-world example of how to use the INSTANTTFA function, see [Queen Bee Piping Example](#).

Hilbert-Huang Transform

The Hilbert-Huang transform (HHT) provides time-frequency information about a signal. It was developed specifically for nonlinear and nonstationary processes. The HHT offers higher resolution than the short-time Fourier transform when applied to nonstationary signals. However, the HHT can be sensitive to noise and can produce spurious components in noisy data. Careful preprocessing is often necessary. You can see the syntax of the HHT call in the [SAS IML documentation](#).

The HHT is a combination of two processes: empirical mode decomposition and Hilbert spectral analysis.

Methodology

Empirical Mode Decomposition (EMD)

Empirical mode decomposition (EMD) decomposes a time series into multiple intrinsic mode functions (IMFs) and a residual. An IMF is a collection of empirical features that have an oscillatory nature with a narrow frequency band. A residual is noise or trend present in the time series. The EMD algorithm in the EMD call in SAS IML uses a sifting process to iteratively find IMFs. IMFs with higher frequencies are generated earlier, and IMFs with lower frequencies are generated later. Thus each IMF represents a different scale or frequency present in the data.

Every potential IMF goes through the process of sifting. The first potential IMF, S_i , is set equal to the input signal, $x(t)$, where i is the current total of IMFs thus far. In other words,

$$S_i = x(t), \quad i = 1$$

After the first IMF has been found using the sifting process, the potential IMF, S_i , is set to IMF _{i} :

$$S_i = \text{IMF}_i$$

Every subsequent potential IMF is determined by the equation

$$S_{i+1} = x(t) - \text{IMF}_i, \quad i > 1$$

A potential IMF might need to go through multiple iterations of the sifting process; therefore, we denote each iteration as k and potential IMFs as $S_{i,k}$. Upon successful validation, a potential IMF, $S_{i,k}$, is promoted to an IMF _{i} in accordance with the prescribed criteria that are presented later in this paper. The iteration count k is then reset to 0, thereby marking the beginning of a new iteration cycle. This iterative refinement process underscores the dynamic nature of IMF identification within the EMD framework.

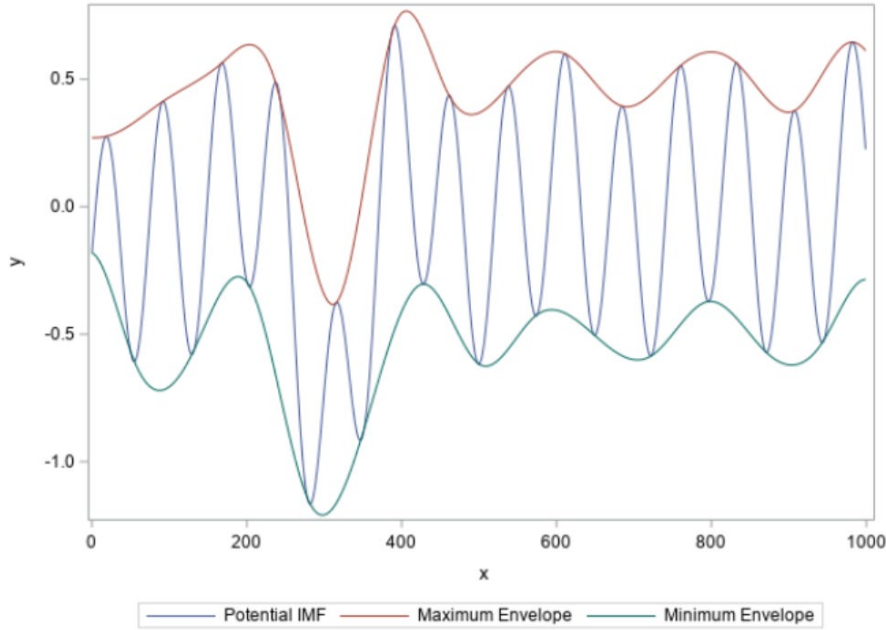
The steps of the sifting process are as follows:

1. For every potential IMF, $S_{i,k}$, identify all local extrema.
2. Use a cubic spline to do the following:
 - a. Create an upper envelope of the local maxima.
 - b. Create a lower envelope of the local minima.
3. Use the upper and lower envelopes to create a mean of the envelopes, m_k .
4. Subtract the mean of the envelopes, m_k , from the original input signal, $X(t)$, to define the next iteration of a potential IMF. In other words,

$$S_{i,k+1} = S_{i,k} - m_k$$

An illustration of an example IMF's minimum and maximum envelopes is shown in **Figure 9**.

Figure 9. Upper and Lower Envelopes of a Signal



From here, you check whether $S_{i,k+1}$ meets one of the following criteria:

- A predetermined number of maximum sifts has been reached. You can specify this number by using the EMD call in SAS IML with the optional parameter *maxSift*. In other words, the EMD call sets $S_i = \text{IMF}_i$ if $k \geq \text{maxSift}$.
- The sifting stopping criterion, SC , which is determined by a Cauchy type of convergence test, becomes smaller than a predetermined threshold value. As with *maxSift*, you can specify this value with an optional parameter, *sqdiff*, when calling the EMD call:

$$SC = \frac{\|S_{i,k-1}(t) - S_{i,k}(t)\|_2}{\|S_{i,k}\|_2} < sqdiff$$

If one of the conditions is met, then $S_{i,k+1}$ is upgraded to an IMF. Before the sifting process begins again to find the next IMF, the next potential IMF is determined as follows:

$$S_{i+1,k=0} = X(t) - \text{IMF}_i$$

When newly derived IMFs become uninformative, a stopping criterion is essential to halt the IMF discovery process and prevent the extraction of redundant components. This criterion ensures computational efficiency and guards against overfitting, enhancing the accuracy and interpretability of the decomposition results. If one of the following conditions is met, the IMF discovery process stops:

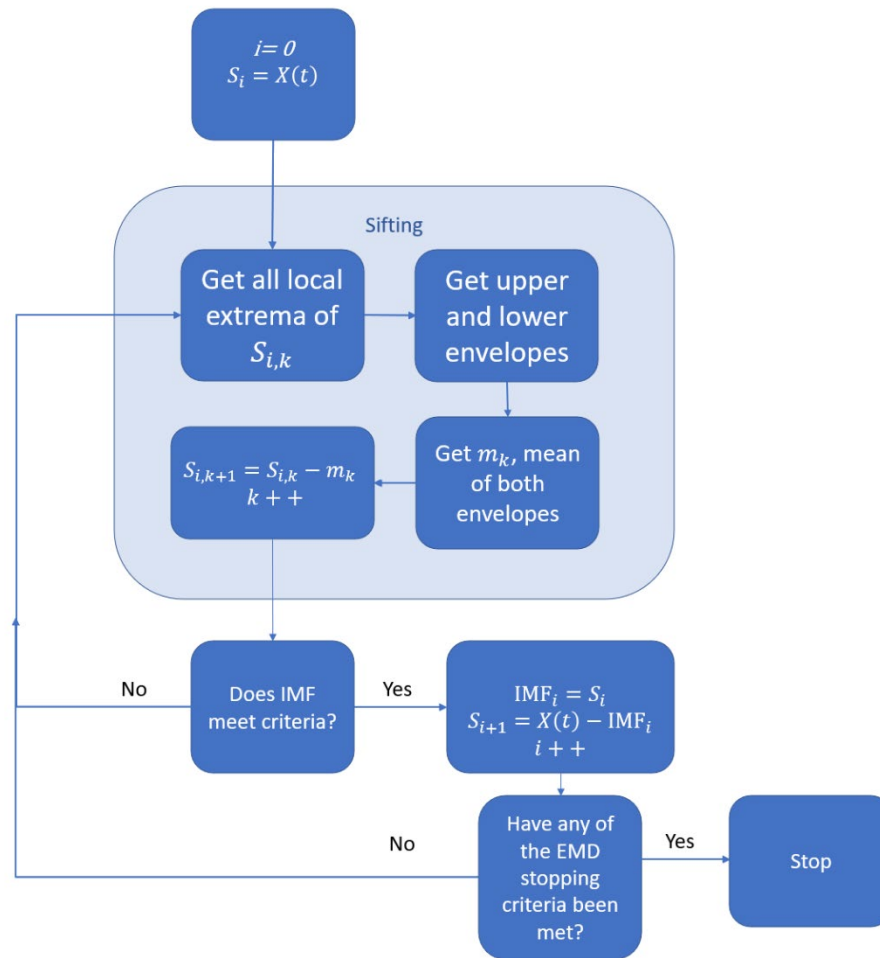
- The total number of extrema and the total number of zero crossings are equal or at most differ by one. Zero crossings are specific points within the signal's temporal domain where the signal waveform intersects the horizontal axis (zero amplitude).

- The total number of IMFs i has reached a preset maximum number of IMFs, i_{\max} . You can specify this value in the EMD call by using the optional parameter *maxIMF*.
- The current energy ratio is smaller than the termination condition. You can specify the termination condition in the EMD call by using the optional parameter *powerR*. The energy ratio (*ER*) is defined as the ratio of the energy at the beginning of the sifting process to the average envelope energy, as follows:

$$ER = \frac{\|S_i(t)\|_2}{\|X(t)\|_2} < powerR$$

The entire process is illustrated in **Figure 10**, where i refers to the i th possible IMF and k is the current and total number of times that S_i has been sifted.

Figure 10. Flow Chart of the Process of Sifting to Determine Whether an IMF Has Been Found



Recall from the Hilbert Transform section that this transform is applied to each IMF to obtain the instantaneous amplitude and frequency of the signal. This makes it possible to analyze the time-frequency characteristics of the data.

The Hilbert transform returns the analytic signal of the input time series vector. The instantaneous amplitude, phase, and frequency are then determined from that analytical signal. Recall that the analytical signal of an input signal $x(t)$ is defined as $z(t) = x(t) + i\hat{x}(t)$, where $\hat{x}(t)$ is the Hilbert transform of $x(t)$:

$$\hat{x}(t) = 2 \int_0^{\infty} X(f)e^{2\pi ift} df$$

where $X(f)$ is the Fourier transform of $x(t)$. (For more information about the Fourier transform, see the section [Short-Time Fourier Transform](#).)

After the analytical function is applied, you can determine the instantaneous amplitude, frequency, and phase. For information about how to calculate these values from the analytical signal, see the section [INSTANTTFA Function](#).

The transform is commonly expressed in the following format, where $\text{amp}(t)$ is the instantaneous amplitude and $\theta(t)$ is the instantaneous phase:

$$\text{amp}(t)e^{-i\theta(t)}$$

The instantaneous energy, e_{inst} , is computed by taking the absolute square of the amplitude:

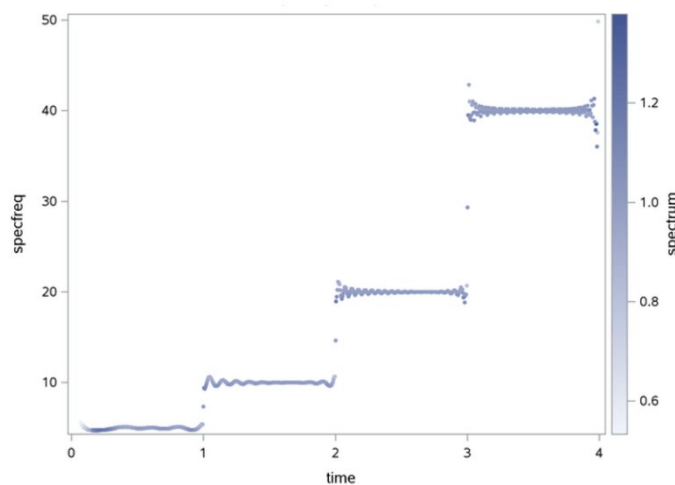
$$e_{\text{inst}} = |\text{amp}(t)|^2$$

Hilbert Spectrum

The Hilbert spectrum is a two-dimensional representation of the data, where time is on one axis; frequency is on the other axis; and the color intensity represents the strength of the signal, which is expressed as various time-frequency locations.

An example of the output of the HHTSPECTRUM call in SAS IML is shown in **Figure 11**. See the [HHTSPECTRUM call documentation](#) to see how this example was generated.

Figure 11. Example of HHTSPECTRUM Call Output



The HHT has been applied in a wide variety of fields, including geophysics, finance, biomedical signal analysis, and climate science. It is used for tasks as diverse as detecting earthquakes, analyzing heart rate variability, and predicting stock market trends.

SAS provides several tools for using the HHT. The EMD call yields IMFs, and the HHT call returns amplitude, phase, frequency, and timestamp information. Additionally, the HHTSPECTRUM call generates frequency spectra by using the HHT. These tools are highly recommended for anyone seeking to implement the HHT effectively.

For examples of how to use the EMD, HHT, and HHTSPECTRUM calls, see the sections [Analyzing EEG Signals](#) and [Instrument-Based Music Decomposition](#).

Peak Detection

A peak in a series is defined as the index of a local maximum. Finding a peak is a popular preprocessing step. The PEAKLOC function returns the peaks of an input series, and the PEAKINFO call returns valuable information about each peak in a signal. You can see their syntax in the [PEAKLOC function documentation](#) and [PEAKINFO call documentation](#).

A peak-finding function is considerably useful in signal processing because of its ability to identify critical points or features within a signal. Peaks represent local maxima or minima, significant transitions, or points of interest that often convey essential information about the underlying signal characteristics. By accurately detecting peaks, signal processing systems can extract valuable insights, facilitate pattern recognition, and enable informed decision-making in various applications.

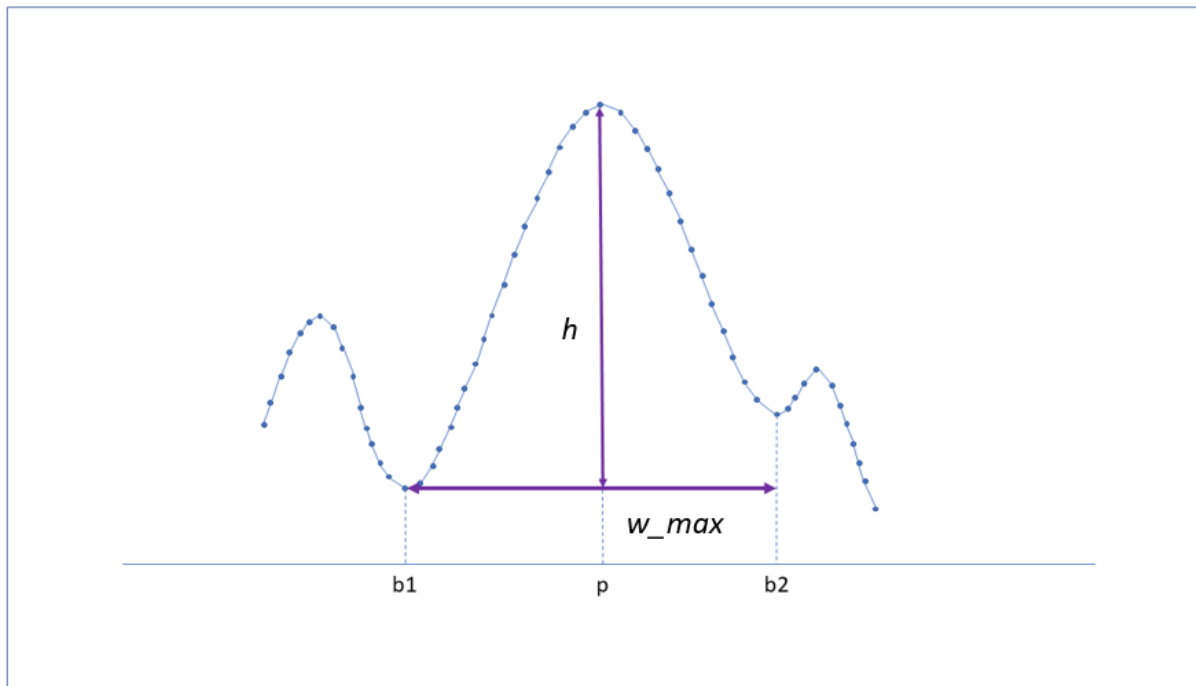
In fields such as biomedical engineering, identifying peaks in physiological signals like electrocardiograms (ECGs) or electroencephalograms (EEGs) helps diagnose abnormalities, monitor patient health, and guide medical interventions. Similarly, in audio processing, peak detection helps identify prominent frequencies, facilitating tasks such as pitch detection, instrument recognition, and sound classification. Moreover, in environmental monitoring or industrial processes, peak-finding functions enable the detection of anomalies or critical events in sensor data, allowing for timely intervention or preventive maintenance. Peak detection also plays a crucial role in data compression, where identifying significant signal features can help reduce data redundancy and optimize storage or transmission efficiency. Additionally, in signal denoising or filtering applications, distinguishing true signal peaks from noise or artifacts enables the preservation of important information while suppressing unwanted disturbances.

Overall, the incorporation of a peak-finding function enriches the analytical capabilities of signal processing systems, empowering researchers, engineers, and analysts to extract meaningful insights, enhance signal fidelity, and unlock the full potential of diverse signal sources across numerous domains.

Methodology

The PEAKLOC function can find peaks by using the following criteria: minimum height, minimum width, maximum width, minimum prominence height, threshold, minimum distance, and maximum number of peaks. The width of a peak is defined as the length between the two bases of a peak, and the height of a peak is the distance between the base and the highest point. **Figure 12** shows an example of a peak, with its height h , and width, w_{max} .

Figure 12. Peak with Its Labeled Height, h , and Width, w_max



Peak prominence refers to the relative significance or importance of a peak within a signal. It quantifies the distinction of a peak from its neighboring points, reflecting its prominence above the surrounding baseline or background noise level. Essentially, prominence measures how much a peak stands out from its surroundings in terms of magnitude or amplitude. For more information about how the PEAKLOC function and PEAKINFO call determine prominence, see the [PEAKLOC function documentation](#).

The criterion min distance in the PEAKLOC function refers to the minimum distance that is required between two local maxima in order for a peak to be detected. The threshold refers to the difference between the value of a peak and the value of the points on its immediate left and right in the series.

The PEAKINFO call returns various information about the peaks in a signal. The function enables you to specify particular peaks as input, thereby providing output exclusively for the designated peaks of interest.

The output of the PEAKINFO call can be categorized into general information about the peak and prominence-related information. It provides the following general information about the peaks:

- peak index, as specified by the user
- index of the beginning of a peak
- index of the end of a peak (which is equal to the beginning if a peak is not a plateau)
- height of a peak
- absolute height of a peak

- width of a peak
- left base point of a peak
- right base point of a peak
- distance between the current peak and its closest neighbor to the right
- status of a peak in regard to its being a plateau

The PEAKINFO call provides the following information about the prominence of the peaks:

- prominence value
- prominence width
- index of the left base, used for prominence calculations
- index of the right base, used for prominence calculations
- parent peak
- prominence-based ranking

For more information about the output of this call, see the [PEAKINFO call documentation](#).

For examples of how to use the PEAKLOC function and PEAKINFO call, see Queen Bee Piping Example 1.

Analyzing EEG Signals

In this example, we use the [EMD call](#) to analyze data from electroencephalography (EEG). More specifically, we use EMD data to decompose EEG data into multiple signals from which we can extract statistical features that can be further used to train models.

EEG data are the electrical activity that is recorded from the scalp that reflects the neural activity of the brain. EEG is a noninvasive technique that measures and analyzes brain activity in real time. EEG data are captured using electrodes that are placed on the scalp, which detect the electrical signals that are generated by neurons firing in the brain. These signals, known as brain waves, are associated with various cognitive processes, emotions, and states of consciousness. EEG data are widely used in neuroscience research, clinical diagnosis of neurological disorders, and applications such as brain-computer interfaces and cognitive enhancement technologies. Analyzing EEG data provides insights into brain function, enabling researchers and clinicians to study brain dynamics, identify abnormalities, and understand cognitive processes and relationships between the brain and behavior. EEG data are a time-series representation of the electrical potentials that are generated by the activity of neurons within the brain.

EEG data analysis involves preprocessing to remove noise, filtering to focus on specific frequency bands, and feature extraction to quantify various aspects of the EEG signal. Signal processing techniques such as spectral analysis and event-related potential (ERP) analysis are commonly used to extract information from EEG data.

In this example, we illustrate how to use the EMD call for data preprocessing, with the aim of facilitating subsequent feature extraction processes. Additionally, we prepare the data for use in the HHT call. The data set in this example is sourced from Kaggle and consists of 31.5 seconds of EEG data that were collected from 14 participants. These participants watched music videos and reported on the emotions that were evoked during the viewing experience. The code for this example can be found at [this GitHub link](#).

The following PROC IML code reads the data set `eeg` from the `work` library. We download the input data file, `features_raw.csv`, from [here](#) and import the data as `work.eeg`. The website that the data came from states that the sampling rate is 256 Hz, so the `FS=` option is set to 256. Using the sampling rate, we can create a time vector. The first 1,000 samples of the EEG signal are then plotted as shown in **Figure 13**.

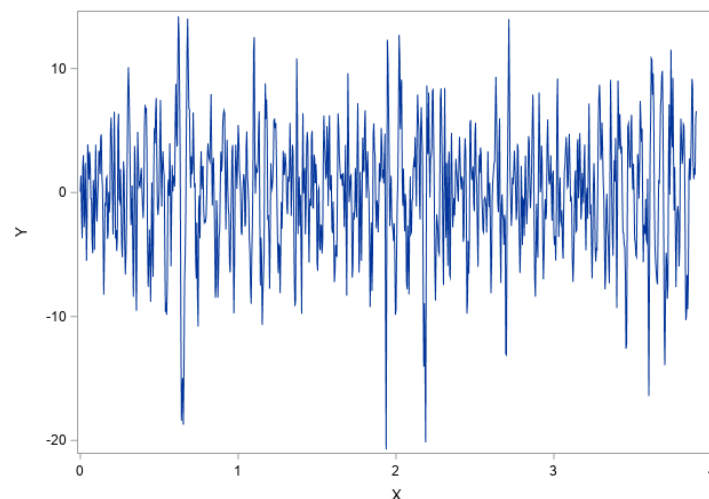
```
proc iml;
    use work.eeg;          /* open the data set */
    read all var {"Fp1" }; /* read 1st vars into vectors */

    fs = 256;
    timestep = 1/fs;
    eeg_rows = nrow(Fp1);

    endtime = eeg_rows/fs;
    time = do(0, endtime-timestep, timestep) `;
    time_rows = nrow(time);

    call series(time[1:1000], Fp1[1:1000]);
```

Figure 13. EEG Time Series Data



The following SAS IML code invokes the EMD call and outputs multiple IMFs. Because we did not specify the maximum number of IMFs for the function to return, the default value of 10 is used. We then double-check the total number of IMFs that the function has output by using the `TOTALIMFS=` option value `NCOL` in case the output

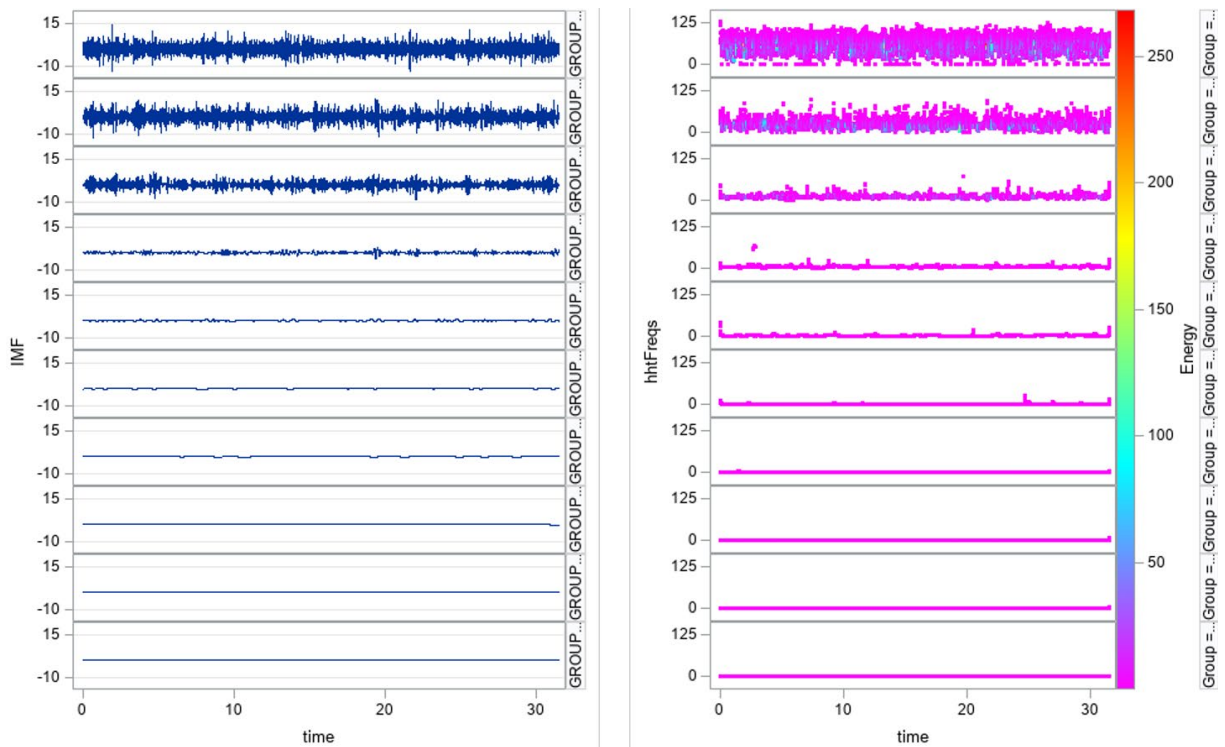
is less than 10. After the total number of IMFs has been determined, that number is saved by the TOTALIMFS= option and is used to plot all IMFs by using the PANELSERIES call. **Figure 14** shows the output of the code.

```
call EMD(IMF, residual, Fp1);

totalIMFs = ncol(IMF);

title "IMFs of the EEG Time Series";
call panelSeries(time, IMF) grid="y" label={"time" "IMF"} NROWS=
totalIMFs;
```

Figure 14. (Left) Plot of All IMFs of Time Series EEG Data Output by EMD Call. (Right) Plot of the Instant Frequencies of Each IMF of EEG Data Output by HHT Call.



In **Figure 14**, you can see that most information has been contained in the first four or five IMFs. For this reason, we use the first five IMFs for feature extraction and time-frequency analysis. In this figure, the right plot contains the frequency information for each of the IMFs seen in the left plot. (Later in this section, we explain how to generate the right plot.)

To confirm that most of the spectral information is in the first five IMFs, you can use the PANELSERIES call to visually display the frequency information for each IMF.

In the image, you can see that the first five IMFs contain most of the information about the signal. As a result, we discard the last five IMFs. By not using IMFs that have redundant information, we also save time and processing power.

Many predictive models are trained on features. The IMFs that are extracted from the EEG signal can be used for feature extraction. For this example, we use basic statistical features such as mean, standard deviation, and variance. We extract these features for each of the five IMFs.

The following code produces the mean, variance, and standard deviation of the EEG signal. We use a five-second window with 50% overlap. We then plot the mean for all IMFs.

```
usefulIMFs = 5;
win_size = 5 * fs;
overlap = 2.5 * fs;

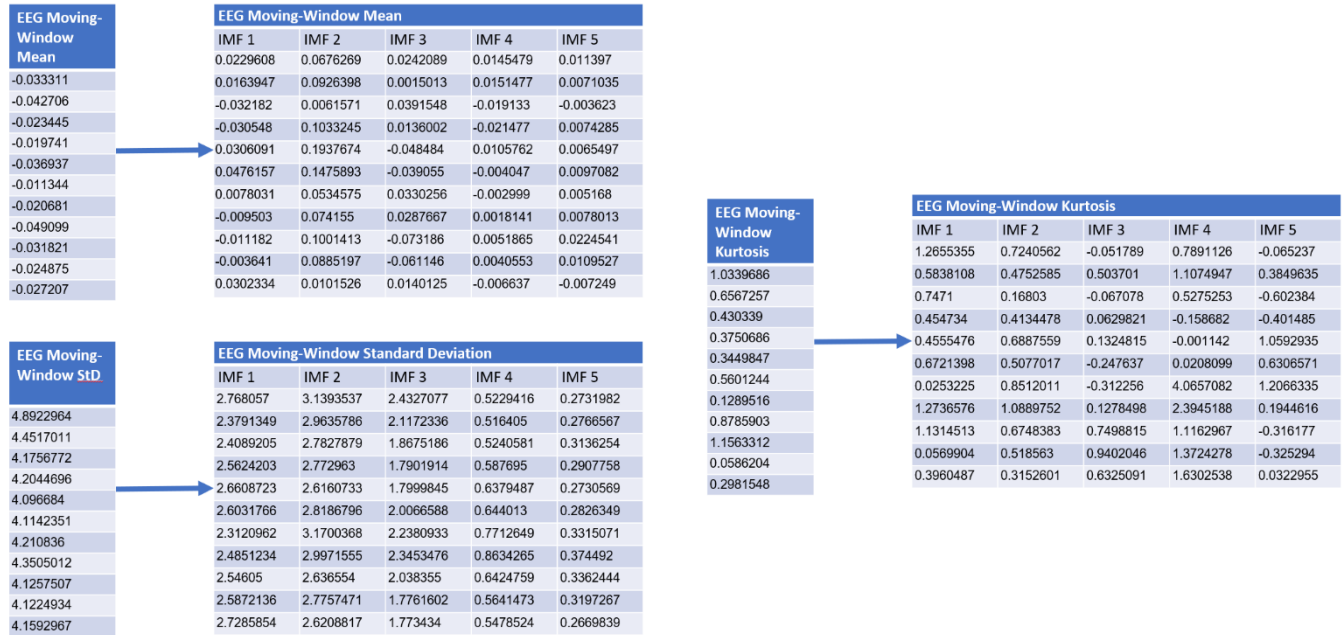
meanrows = floor((eeg_rows/overlap)-1);
eeg_mean = j(meanrows,usefulIMFs);
eeg_std = j(meanrows,usefulIMFs);
eeg_kur = j(meanrows,usefulIMFs);

do j = 1 to usefulIMFs;
    m_st = 1;
    m_end = win_size;
    currIMF = IMF[ ,j];

    do I = 1 to meanrows;
        eeg_mean[i,j] = mean(currIMF[m_st:m_end]);
        eeg_std[i,j] = sqrt(var(currIMF[m_st:m_end]));
        eeg_kur[I,j] = kurtosis(currIMF[m_st:m_end]);
        m_st = m_st + overlap;
        m_end = m_end + overlap;
    end;
end;
```

Using EMD enables us to increase the dimensionality of our features. We went from having features from a single one-dimensional time series to being able to extract features from multiple subbands that are generated as IMFs. This technique is especially useful for signals that are more difficult to interpret. There is much ongoing research into understanding and interpreting EEG signals, so the process of breaking them down before feature extraction might prove to be helpful for training models. The output of this increase in dimension is shown in **Figure 15**. Notice that the IMFs increased the dimensionality of the EEG moving-window features, mean, standard deviation, and kurtosis.

Figure 15. Example of the Features Extracted



The following code creates the plot on the right side of Figure 14:

```
call HHT(Amp, Phase, hhtFreqs, time, IMF, fs);
Energy = Amp##2;

title "Instantaneous Frequencies of IMFs";
call PanelScatter(time, hhtFreqs)
  colorvar=Energy
  colorramp="Rainbow"
  nrows = 10
  option="markerattrs=(symbol=CircleFilled size=3)";
```

The following section presents a more detailed example of using the HHT function.

Instrument-Based Music Decomposition

In this example, a time-varying signal, indicative of a recording that includes various musical instruments, undergoes a process of decomposition. This decomposition procedure is enacted with the specific objective of discerning individual instruments within the recording. The outcome yields a series of distinct music files, each representing a single instrument that is isolated from the original recording.

We start with two sound files of two different musical instruments. We first analyze them separately. This enables us to have a ground truth of the correct frequencies that we should see in the results. We combine two signals into

a single .wav file by using a third-party tool, and then we use empirical mode decomposition (EMD) to decompose the music file into multiple IMFs. The HHT is then used to analyze the IMFs in the frequency domain. The IMFs then separate each instrument from the combined file. Next, we use the HHT to analyze the IMFs in the frequency domain to check whether our results are correct.

Before getting started, you need to download the [file](#) and rename it flute.wav. Then do the same with the next [file](#) and rename it bass.wav. Download combo.wav and the %read_wav and %write_wav macros from the [SAS GitHub location](#).

We analyze the two signals separately. By looking at them individually, we can extrapolate which frequencies we are already looking for when they are combined. Many real-world problems do not present ground truth, and often separation has to occur with limited information about each individual signal. This often requires trial and error, but for this example, we simplify the process by having a source of ground truth for each signal.

We start by converting the .wav sound file into a SAS data set. This can be done as follows, by using the %read_wave macro available in the [GitHub repository that accompanies this paper](#):

```
proc iml;
  %read_wav(file=yourfilelocation\flute.wav, _out=work.flute);
  %read_wav(file=yourfilelocation\bass.wav, _out=work.bass);
quit;
```

The original signal has a sampling rate of 44.1 kHz. Subsequent examination reveals that the uppermost frequencies inherent in the instruments are approximately 500 Hz. Consequently, downsampling from 44.1 to 4.4 kHz can be seamlessly executed without compromising pertinent information. So the downsampling factor is 10. It is imperative to filter the signal to eliminate any extraneous noise that might exist beyond the frequencies of interest. Failing to filter beforehand can result in distortion through a phenomenon known as aliasing; this makes it more challenging to manage unwanted frequencies, which are referred to as noise. To filter any frequencies below 4.41 kHz, we set the cutoff frequency to be the inverse of the downsampling rate, as shown in the following code. We use the DFDESIGN call to create a lowpass Butterworth filter that meets the specifications that we are looking for, and we use the DFSOSFILT function to filter the signal. Then we use the DO function to perform uniform downsampling. The filtered and downsampled signal is saved in a new data set.

```
%let down_sample_factor = 10;
%macro down_sample_data(factor);
  %let cutoff_freq = %sysevalf(1.0/(&factor));
  %put &cutoff_freq;

  proc iml;
    use work.flute;
    read all var {time, channel1};
    close work.flute;
    t = time;
    x = channel1;

    filter_name = "butter";
    filter_type = "lowpass";
    n = 10;
    Wc = &cutoff_freq;

    call dfdesign(b, a, z, p, k, filter_name, filter_type, n, Wc);
```

```

y = dfsosfilt(x, z, p, k);

dim = dimension(y);
len = dim[1];

idx = do(1, len, &factor);
time = t[idx];
channell = y[idx];

create work.flute_data_new var {time, channell};
append;
close work.flute_data_new;
quit;

proc sgplot data=work.flute_data_new;
  title "Downsampled Data";
  series x=time y=channell;
  xaxis label= "Time (second)";
  yaxis label= "Amplitude";
run;
%mend;

%down_sample_data(&down_sample_factor)

```

To downsample the bass, use the preceding code, replace the data set `work.flute` with `work.bass`, and create the data set `work.bass_data_new`.

The following code opens the new SAS data set, `work.flute_data_new`. We double-check that the sampling frequency is the expected 4.41 kHz by using the MEAN function. We use the EMD call to decompose the signal into multiple IMFs and plot them as shown in **Figure 16**. By default, the EMD function outputs the first 10 IMFs, but that number can possibly be increased by using the *maxIMFs* option in the EMD function. We then use the HHT to plot the frequency information by using the PANELSCATTER call on the signal for each IMF; the plot is displayed in **Figure 17**.

```

proc iml;
  use work.flute_data_new;
  read all var {time channell};
  close;
  signal = channell;
  music_rows = nrow(signal);
  timesteps = mean((time[2:music_rows]-time[1:music_rows-1]));
  fs = 1/timesteps;

  call EMD(IMF, residual, signal);

  totalIMFs = ncol(IMF);

  title "IMFs of the Musical Time Series";
  call panelSeries(time, IMF) grid="y" label={"time" "IMF"} NROWS=
totalIMFs;

  freqRes = .;
  freqRange = .;
  energyThreshold = 1;

```

```

format = 'SPARSE';

call HHT(Amp, Phase, hhtFreqs, time, IMF, fs);
Energy = Amp##2;

title "Instantaneous Frequencies of IMFs";
call PanelScatter(time, hhtFreqs[,1:totalIMFs])
  colorvar=Energy[,1:totalIMFs]
  colorramp=palette("YLORRD",7)
  nrows = 1
  option="markerattrs=(symbol=CircleFilled size=3)";

title "Instantaneous Frequencies of IMFs";
call HHTSPECTRUM(spectrum, specfreq, time, IMF, fs,
  freqRes, freqRange, energyThreshold, format);

quit;

```

Figure 16. IMFs of a Decomposed Time Series Musical Signal

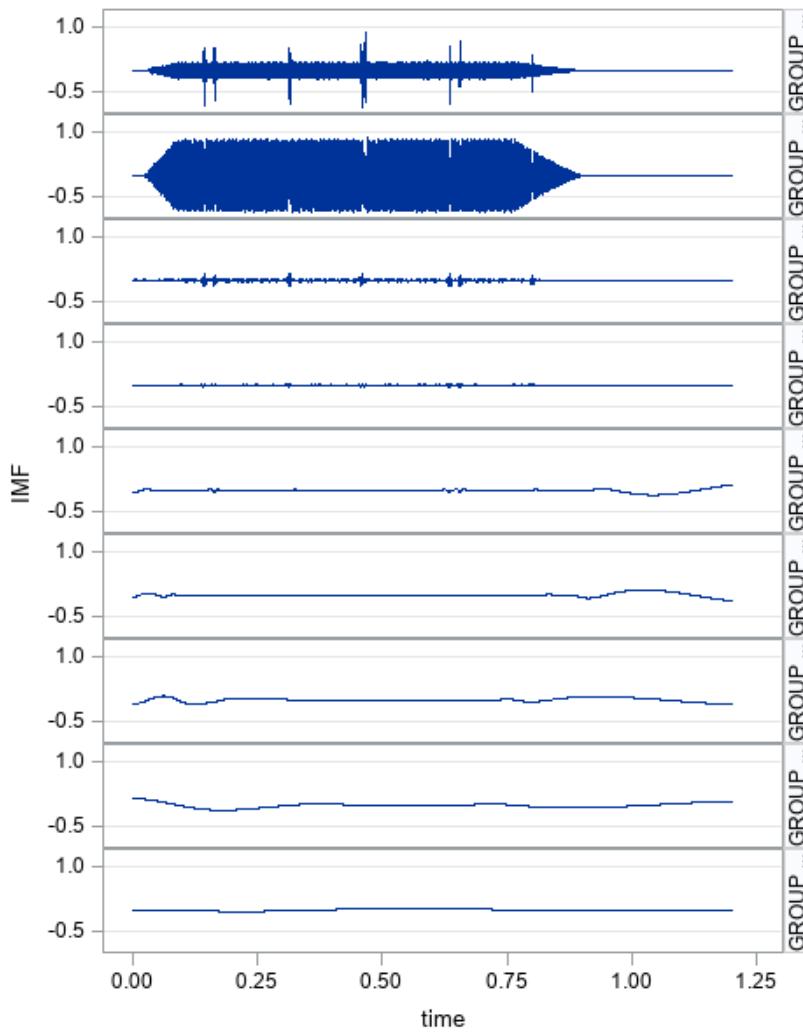
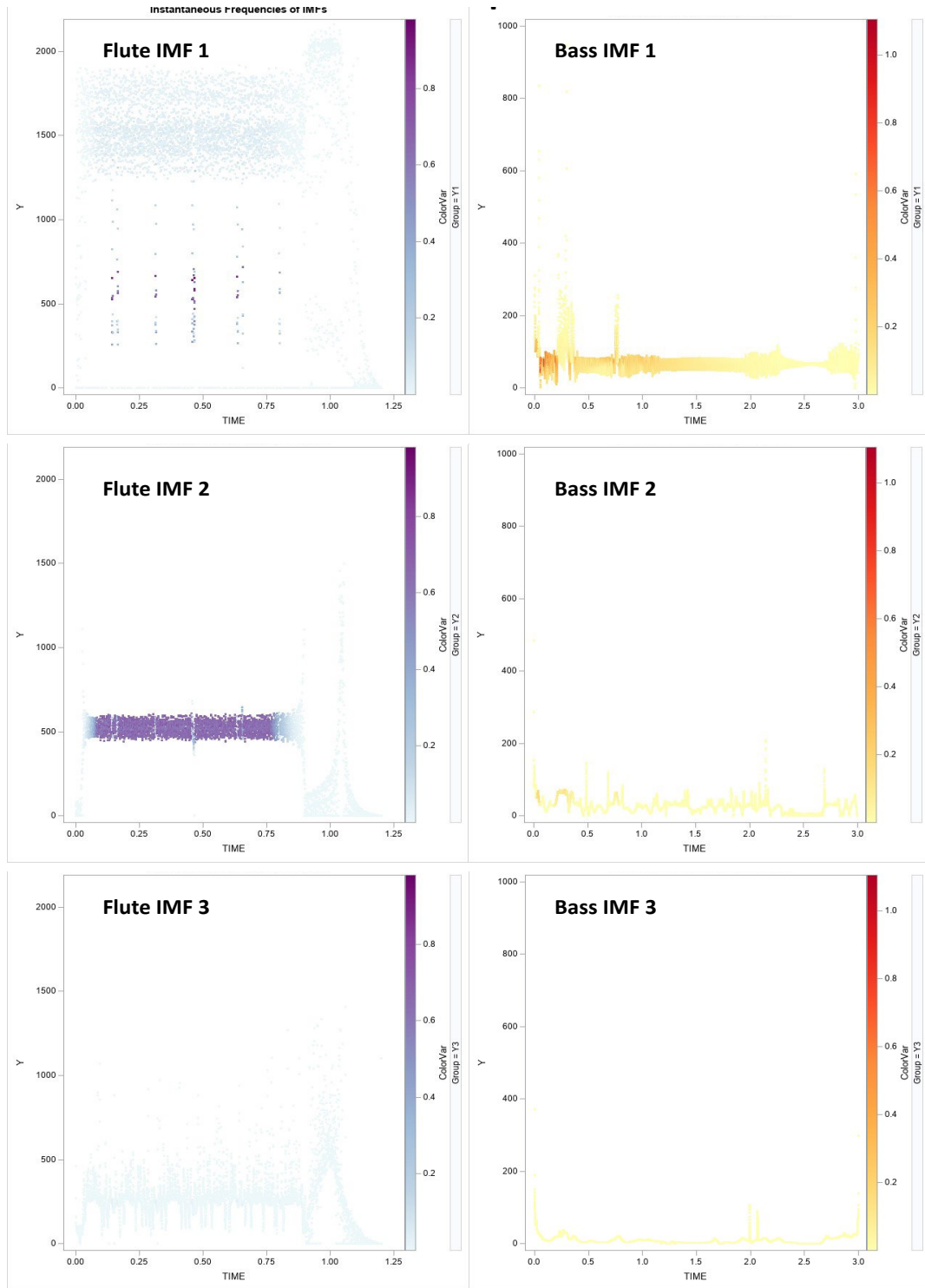


Figure 17. (Left) Frequencies of the First Three IMFs from a Musical Signal Produced by a Flute. (Right) Frequencies of the First Three IMFs from a Musical Signal Produced by a Bass Instrument



We repeat this process for the bass file and for the combined .wav file that has both instruments playing simultaneously. **Figure 17** shows the frequency information for the first three IMFs of the flute (left) and the bass (right). The rest of the subsequent IMFs did not contain useful frequency information.

Figure 18 shows the frequency information for the first six IMFs of the combined signal. We can observe that we were able to get a decomposition of both the higher-frequency flute signal and the lower-frequency bass signal from the combined signal by using the EMD and HHT calls. From here we can play with the reconstruction to separate the signals. Notably, IMF2 prominently encapsulates the high-frequency components that are attributed to the flute, extending for approximately 0.9 seconds. IMF1 also exhibits residual high-frequency elements inherent in the flute's spectrogram. Additionally, IMF2, IMF3, and IMF4 beyond the 0.9-second mark encapsulate the bass-related information within the signal.

The beginning of the following code has been removed because it is similar to the code that was already presented in this example. You can assume that the beginning of the code is the same, except for the `combo.wav` data set that is used, as well as variable names and data set names. The entirety of the code for the combined .wav file, called `combo.sas`, is located [here](#).

To determine exactly when IMF2 switches from low frequencies to high frequencies, we look at the energy in IMF2. To do so, we save the energy values that are associated with IMF2 in `work.energy` in SAS. You can see the code in the following program. Looking at this file, we can see that the change in energy happens at index 5441. We save this index as the variable `fluteend` in the following code:

```
varnames = {'time', 'channel1'};
e = (Energy[,2]);
create energy from e[colname=varnames];
append from e;
close energy;

fluterange = j(music_rows,2);
bassrange = fluterange;
fluteend = 5441;
```

Now we have all the information that is needed to re-create the separated signals from the IMFs and the observations that have been made so far. The following code creates the data sets `work.fluteout` and `work.bassout` from the combination of IMFs that were mentioned previously. We also split IMF2 so that the appropriate information goes to the correct instrument.

```
temp1 = IMF[1:fluteend,1]+IMF[1:fluteend,2];
fluterange[1:fluteend,1] = time[1:fluteend,1];
fluterange[1:fluteend,2] = temp1;

create fluteout from fluterange[colname=varnames];
append from fluterange;
close fluteout;

temp2 = IMF[,3]+IMF[,4];
temp2[fluteend:music_rows] =
temp2[fluteend:music_rows]+IMF[fluteend:music_rows,2];
bassrange[1:music_rows,1] = time;
bassrange[1:music_rows,2] = temp2;
```

```
create bassout from bassrange[colname=varnames];
append from bassrange;
close bassout;
```

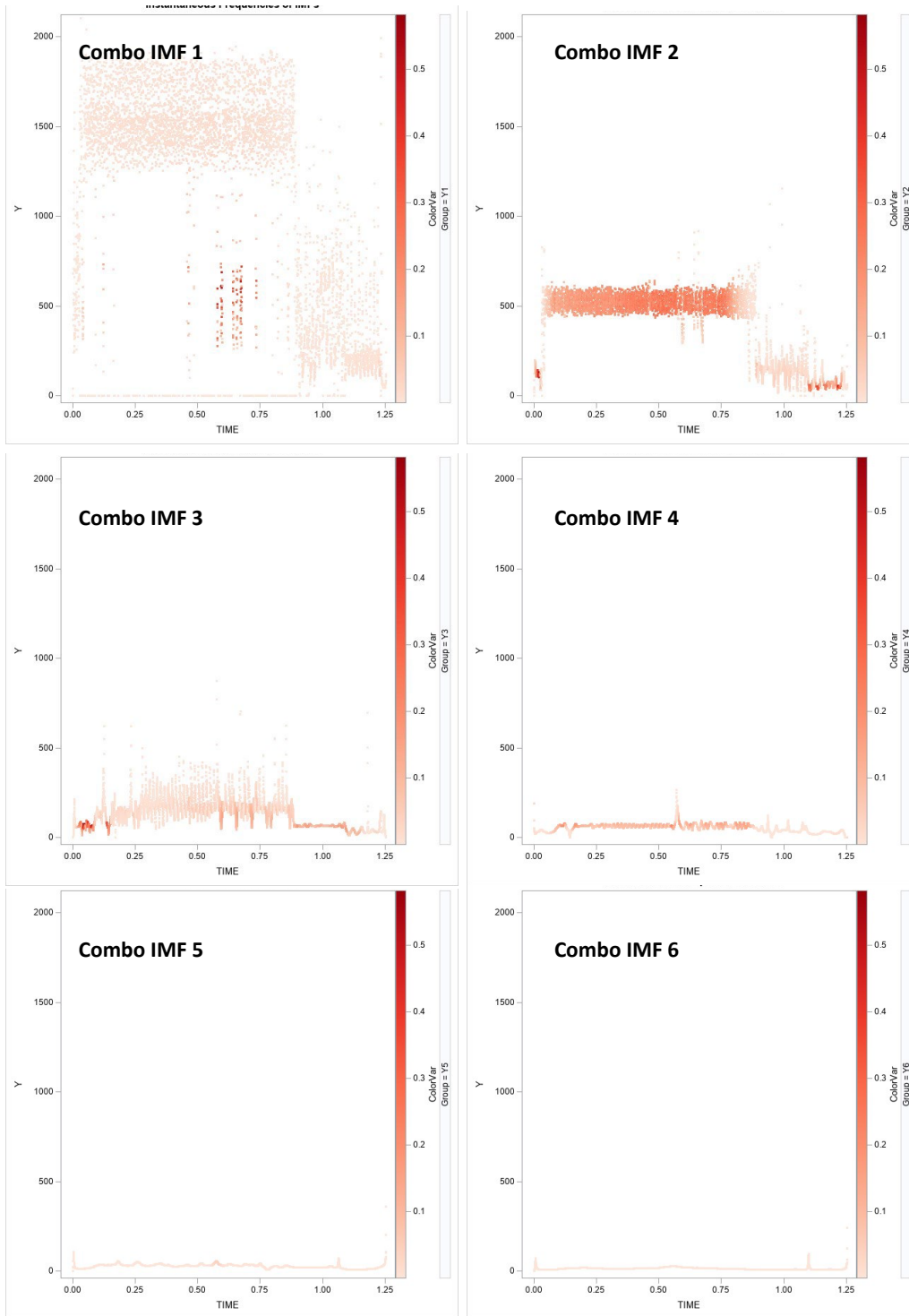
After the data sets have been created, we can use the %write_wave macro to export work.fluteout and work.bassout to brand-new .wav sound files as follows, and we can audibly compare how close our reconstructions were to the original files by using any media player that supports .wav files:

```
%write_wav(indata=work.fluteout,sampling_frequency=5530,_out=%nrquote(C:
\fluteout.wav) );
```

```
%write_wav(indata=work.bassout,sampling_frequency=5530,_out=%nrquote(C:
bassout.wav) );
```

In this instance, we conducted an analysis on three distinct musical signals: one from a flute, another from a bass instrument, and a third that represents a combination of these instruments. Using empirical mode decomposition (EMD) and Hilbert-Huang transform (HHT) methodologies, we decomposed and examined these signals to gain insights into their frequency characteristics. Notably, for the composite signal, we successfully segregated both instruments into individual files. Although you can also achieve decomposition through other methods, such as independent component analysis (ICA), it is important to note that unlike the HHT, ICA requires the number of signals to match the number of sources. With the HHT, we had the flexibility to initiate decomposition directly from the composite signal without prior analysis of individual files. However, in real-world scenarios, experimentation and parameter adjustment can be indispensable for achieving optimal separation and reconstruction, especially in the absence of ground-truth data that pertain to the source signals. In our case, having access to the ground truth obviated the need for extrapolating frequency ranges for each instrument, thereby expediting the analysis process.

Figure 18. Frequencies of the First Six IMFs from a Musical Signal Produced by a Bass Instrument and a Flute Playing Simultaneously



Queen Bee Piping Example 1

In this example, we show how you can use the [PEAKLOC function](#) and [PEAKINFO call](#) in SAS IML to monitor the health of a beehive by demonstrating how it can potentially detect the piping sounds that the queen bee makes.

Bees play a crucial role as pollinators, contributing significantly to agricultural productivity. Without them, humans would lose approximately 75% of all produce. Among the important bee species, honeybees stand out. However, loss of honeybee colonies has been escalating every year because of factors like inadequate nutrition; bee viruses linked to mites; and various human causes, such as pesticide use and climate change. Consequently, monitoring beehive health has garnered considerable attention among beekeepers. Previously, SAS introduced a comprehensive pipeline for monitoring hive health, which includes detecting queen bee piping signals (Liao 2020).

Piping serves as a sophisticated form of communication within a beehive, facilitating interaction between worker bees and the queen bee, often signaling significant events that involve the queen. Specifically, the queen uses this communication method to announce her intention to separate from the hive; the result is that about 50% of its worker bees follow her and form a new hive. In instances where the queen bee experiences mortality or illness, worker bees are compelled to prepare for the emergence of a new queen. In the absence of a discernible queen, the worker bees emit piping sounds, awaiting a responsive reaction. Beekeepers, who traditionally inspect their hives every two weeks because of the invasive nature of such assessments, stand to benefit from using technology for nonintrusive hive monitoring. This technological approach enables more frequent and unobtrusive surveillance of beehive health, empowering keepers to promptly identify and address issues in support of ailing hives. Queen bee piping has been found to have a frequency range of approximately 300–500 Hz. In this example, we demonstrate one possible way to detect the piping of a queen bee by using the signal processing tools available in SAS IML.

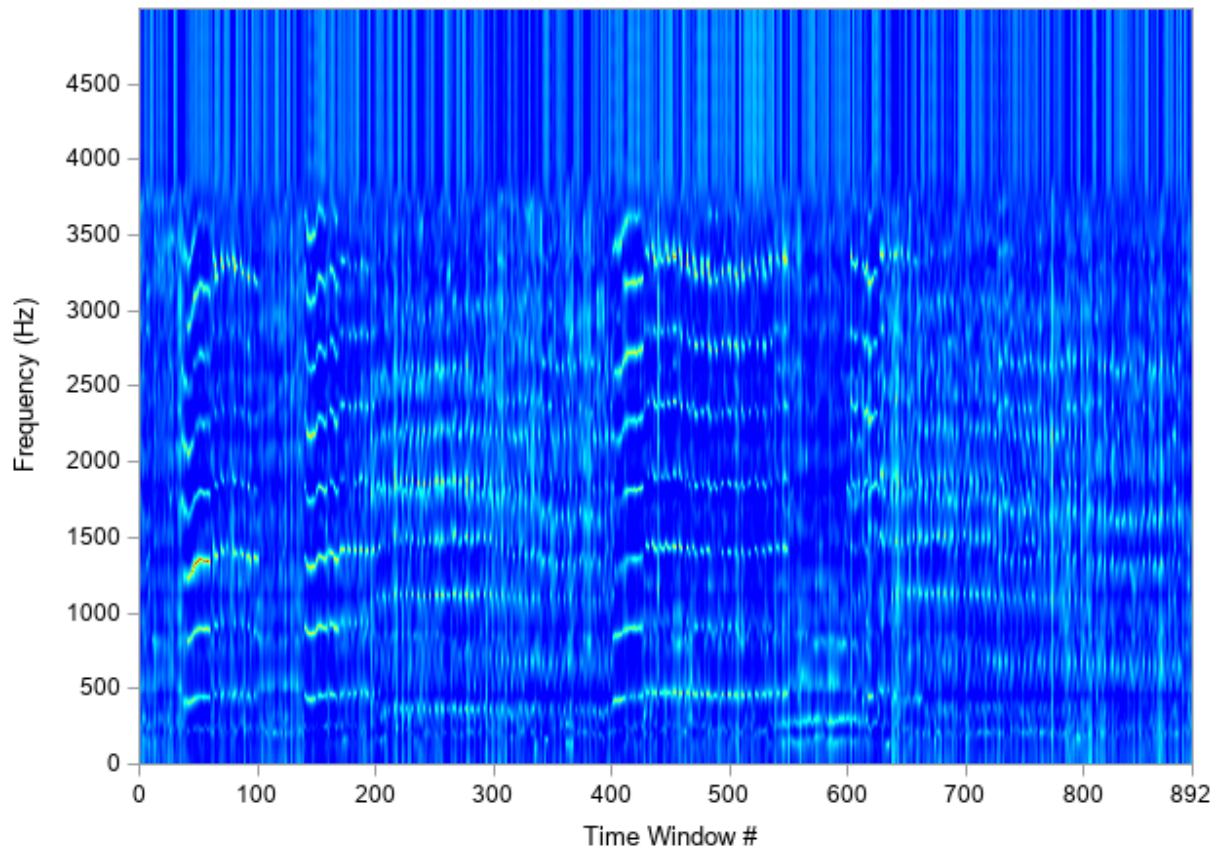
In this example, an audio recording of a beehive is created using specialized but inexpensive audio equipment. The audio is sampled at 11,025 Hz. The beekeeper has purposely triggered a queen bee event by removing the current reigning queen from the hive, compelling the worker bees to initiate the process of generating a new queen. Subsequently, upon the emergence of the new queen, the distinct sound of queen bee piping becomes audible. To showcase the practical application of the frequency analysis, we present an approach that uses the PEAKLOC function to detect and characterize the fundamental frequency of the queen bee piping, along with its accompanying harmonics.

The following code takes the spectrogram data of queen bee piping. The data have already been filtered and windowed, and the magnitude has been adjusted to produce the results shown in **Figure 19**. This work can be found [here](#). To learn more about the previous steps, see the [previous work](#).

```
%let piping_data_new = bee;

%let win_len_t = 0.15;
%let order = 50;
```

Figure 19. Heat Map of Queen Bee Piping. Lighter-colored segments represent the fundamental frequency and the harmonics.



The following code loads the data set `myWork.Spectral_adj`, which contains the spectrum information:

```
proc iml;
  use myWork.spectral_adj;
  read all var _NUM_ into spectral;
  close myWork.spectral_adj;

  use myWork.Fs;
  read all var _NUM_ into Fs;
  close myWork.Fs;
```

The `myWork.Spectral_adj` data set contains spectrum information in a two-dimensional format. Here, the columns correspond to the total count of windows, and the rows contain frequency data. Each entry represents the magnitude in decibels (dB) of the signal for a specific frequency and window. To determine the total number of windows, specified in the `NUM_WINS=` option, and the total number of frequencies, specified in the `NUM_FREQ=` option, we use the `DIMENSION` function.

```

ds = dimension(spectral);
num_freq = ds[1];
num_wins = ds[2];

```

The total count of frequencies, along with the sampling frequency denoted as F_s , determines the frequency values for each row. The following code generates a vector of frequencies that range from one up to the Nyquist frequency, which is half the sampling frequency ($F_s/2$):

```

freq_x = do(1, num_freq, 1)*Fs/(2*num_freq);

```

Before we use the PEAKLOC function, we configure optional parameters such as the minimum height (minH) and the minimum distance between peaks (minD) as in the following code. The minimum height is established as the mean of the magnitude. Harmonics, which are integer multiples of the fundamental frequency, are considered. The minimum distance ensures that no two peaks are detected if they fall within a 100 Hz range of each other. Given that the fundamental frequency typically ranges from 300 to 500 Hz, any peak that is detected within 100 Hz of another peak is deemed too close and thus disregarded.

```

minH = mean(spectral[:]);
minD = round(100*2*num_freq/Fs);

```

Because the fundamental frequency of the queen bee's piping is usually between 300 and 500 Hz, the following code identifies the starting and ending indices in the frequency vector, `freq_x`, that satisfy this criterion. Subsequently, `idx_seg` is generated as a vector of indices that is used to reference the frequencies within this range.

```

minFidx = 300*2*num_freq/Fs;
maxFidx = 500*2*num_freq/Fs;
idx_seg = do(1, num_freq, 1);

```

A vector named "filter" is constructed as follows; it serves as the optional parameter for the PEAKLOC function. This function uses the provided values to exclusively identify peaks that adhere to the criteria that are outlined within the optional parameter. In this case, only the minimum height and minimum distance parameters are explicitly defined. All other parameters remain at their default settings, indicated by the use of the "." character.

```

filter = minH // . // . // . // . // minD;

```

The heat map shown in **Figure 19** provides a visual indication of distinct instances of queen bee piping across multiple segments. The initial segment spans approximately from 30 to 75. Subsequently, these identified time windows are iteratively input into the following DO loop. Within each iteration, the current time window, which is denoted as "cur_seg," is passed into the PEAKLOC function along with the optional parameter vector, "filter."

```

do i = 30 to 75;
  cur_seg = spectral[, i];

  idx = peakloc(cur_seg, filter);

```

The J function creates a new column that is the size of `cur_seg` as follows. The values of the peak magnitudes at the peak locations that are returned by the PEAKLOC function cause the vector that is denoted as "val" to have all empty values except at the peak locations.

```

val = j(1, max(nrow(cur_seg), ncol(cur_seg)), .);

```

```
val[idx] = cur_seg[idx];
```

The vector “val” is then established to find candidates of the fundamental frequency by finding all indices in the 300–500 Hz range. If there is no piping in the segment, cur_seg, then a fundamental frequency might not be found. The IEMPTY function looks for any possible candidates for the fundamental frequency. If none are found, the code jumps to the end of the loop, through the label CONTINUE, so that the code can continue to the next segment.

```
idx_F = loc(idx < maxFidx & idx > minFidx);

if isempty(idx_F) then do;
    print "empty fundamental frequency";
    print i;
    goto CONTINUE;
end;
```

After the maximum magnitude value is identified from the list of potential fundamental frequencies, the code designates this value as the index of the fundamental frequency, as follows. Specifically, the index of the fundamental frequency within the candidate list, denoted as “idx_F,” is stored as the starting index, which is referred to as “start_idx.” Moreover, the index of the fundamental frequency, represented by “F_idx,” along with its corresponding magnitude value, “F_val,” is preserved with respect to all detected peaks. After this, the actual frequency value, denoted as “F0,” is computed using the index “F_idx,” the sampling frequency “Fs,” and the total number of frequencies “num_freq.”

```
temp_idx = loc(max(val[idx_F]));
start_idx = idx_F[temp_idx];
F_idx = idx[start_idx];
F_val = val[F_idx];

F0 = F_idx*Fs/(2*num_freq);
```

A tolerance, called “tolerance,” of about 80 Hz is established as follows so that any harmonics found are not within 80 Hz of each other. An empty vector, “harmonics_idx,” is created to store the indices of the harmonics.

```
tolerance = 80 * 2*num_freq/Fs;
nidx = max(nrow(idx),ncol(idx));
harmonics_idx = j(nidx, 1, 0);
```

The following DO loop finds the indices of the harmonics of the fundamental frequency present. The variable “r” is the remainder (calculated by the MOD function) of the index that is being evaluated to see whether it is a harmonic divided by the index of the fundamental frequency. The variable “r_c” is the index of that fundamental frequency subtracted from that remainder. If either the remainder variable r or the variable r_c is less than the tolerance, then that peak index is considered a harmonic.

```
hflag = 0;
do j= start_idx+1 to nidx;
    cur_idx = idx[j];
    r = mod(cur_idx, F_idx);
    r_c = F_idx - r;
    if (r < tolerance) | (r_c < tolerance) then do;
        harmonics_idx[j] = 1;
        hflag = 1;
    end;
end;
```

The following code examines whether any harmonics have been detected. If no harmonics are identified, this suggests that the fundamental frequency might not be associated with piping. This inference is drawn from the observation that during the evaluation of queen bee piping, all instances of piping typically have accompanying harmonics. Therefore, if no harmonics are detected, the code proceeds to the next window.

```

if (hflag<1) then do;
  print "no harmonics found: hflag 0";
  print i;
  goto CONTINUE;
end;

```

The next section of code uses the binary vector, "harmonics_idx," to find the index of the harmonics that pertain to the peak indices, "h_idx"; the frequency value, "h_freq"; and magnitude value, "h_val," at the location of the found harmonics:

```

temp_idx = loc(harmonics_idx>0);
h_idx = idx[temp_idx];
h_val = val[h_idx];
h_freq=h_idx*Fs/(2*num_freq);

```

Next, the vectors relevant to plotting the fundamental frequency and harmonics are consolidated into a data set named `plotData`. Subsequently, this data set is used in PROC SGPLOT to visualize the results. By using the SUBMIT statement, you can invoke a procedure within PROC IML. PROC SGPLOT enables you to generate a series plot that features two distinct scatter plots overlaid. The series plot displays all frequency and magnitude values within the current window segment, `cur_seg`. The first overlaid scatter plot showcases the harmonics, which are denoted by green markers positioned at the peaks. Finally, the fundamental frequency is depicted as the third scatter plot, represented by a red circle. The ENDSUBMIT statement signifies the end of the PROC SGPLOT run, allowing the code to revert to PROC IML statements.

```

create plotData var {freq_x, cur_seg, F0, F_val, h_freq, h_val};
  append;
close plotData;

title "F0 and Harmonics";

submit;
proc sgplot data=plotData;
  series x=freq_x y=cur_seg / lineattrs=(thickness=2)
legendlabel="Spectral Curve";
  scatter x=h_freq y=h_val / filledoutlinedmarkers
markerfillattrs=(color=green)
markeroutlineattrs=(color=green thickness=2)
markerattrs=(symbol=circlefilled size=13)
legendlabel="Harmonics";
  scatter x=F0 y=F_val / filledoutlinedmarkers
markerfillattrs=(color=red)
markeroutlineattrs=(color=red thickness=2)
markerattrs=(symbol=circlefilled size=13)
legendlabel="Fundamental Frequency";
  xaxis label="Frequency (Hz)";
  yaxis label="Magnitude (dB)";

```

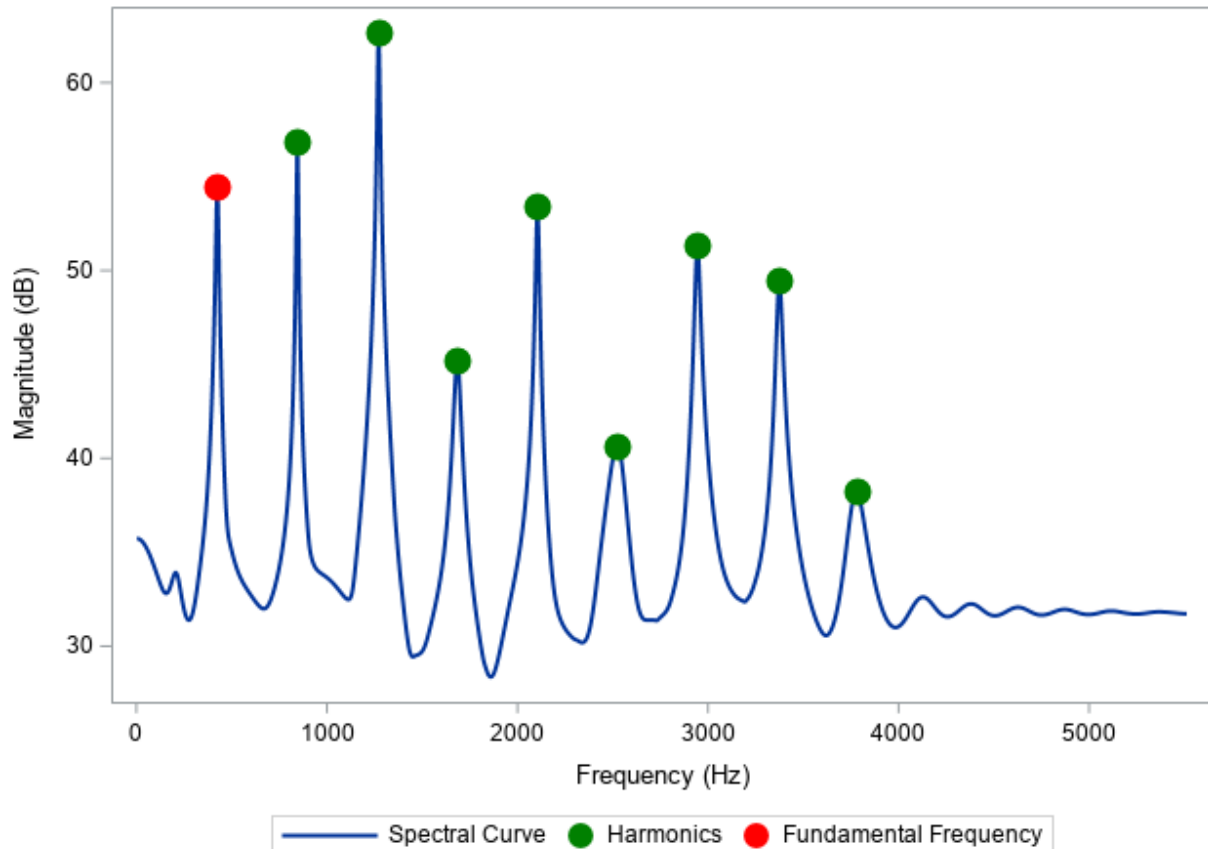
```

        title "Current Window";
    run;
    endsubmit;
CONTINUE:
end;
quit;

```

The results of the plotting code for the 45th time window are shown in **Figure 20**.

Figure 20. Fundamental Frequency of the Queen Bee Piping and Its Harmonics



We can use the PEAKINFO call to further improve this example. **Table 1** shows some example plots of time windows. The first does not have a clear fundamental frequency with harmonics, so it would be useful to omit that time window. We will use the PEAKINFO call to determine a good prominence threshold, and then we will use the same call again to omit the time windows in which the fundamental frequency does not meet that prominence threshold. We can compare the prominence of the fundamental frequencies for other time windows not shown in **Table 1** to get similar results. By comparing the prominence of the clear fundamental frequencies and harmonics, we can estimate that a prominence threshold of 5 or greater might work for unplotted examples.

We can edit the following code that is referenced earlier:

```

temp_idx = loc(max(val[idx_F]));
start_idx = idx_F[temp_idx];
F_idx = idx[start_idx];

```

```
F_val = val[F_idx];

F0 = F_idx*Fs/(2*num_freq);
```

When the index of the fundamental frequency has been found, we can use the PEAKINFO call to check the prominence and skip to the next window if the prominence threshold is not met. The preceding code now becomes the following code:

```
temp_idx = loc(max(val[idx_F]));
start_idx = idx_F[temp_idx];
F_idx = idx[start_idx];

/* check if prominence is high enough to be a fundamental frequency */
min_prom= 5;
call peakinfo(output1, labels1, cur_seg,F_idx);
if(output1[1,9]<min_prom) then do;
    print "F0 prominence too low";
    goto CONTINUE;
end;

F_val = val[F_idx];
F0 = F_idx*Fs/(2*num_freq);
```

Here the minimum prominence threshold, `min_prom`, is set to 5. The PEAKINFO call is then used, with the optional parameter of the peak location index that we are interested in. In this example, that location is the fundamental frequency index, `F_idx`. From the PEAKINFO call documentation, we know that the ninth column has the prominence information of the peak, so we use an IF statement to check whether the prominence meets our criteria. If it is too low, the code goes to the CONTINUE label, in order to skip to the next window.

In **Table 1**, the last row contains a clear fundamental frequency with harmonics, but notice that the last harmonic has a much lower amplitude than the others. It is unclear whether it is a true harmonic or just part of the trailing high-frequency noise present. We can use the PEAKINFO call again to remove the instance where a harmonic is incorrectly labeled.

To do this, we edit the code at the point where `h_idx` is first defined, as follows. We first check to see whether `h_idx` is empty, because we do not want to use a PEAKINFO call with an empty vector. The PEAKINFO call then returns all the prominence values of the harmonics. The PRINT statement displays the information. **Figure 21** shows the output.

```
h_idx = idx[temp_idx];

if isempty(h_idx) then do;
    print "empty harmonics: h_idx empty";
    goto CONTINUE;
end;

call peakinfo(h_output, labels, cur_seg,h_idx);
print (h_output[,{1 9}]`)[r=labels c=('Peak1':'Peak10') format=best5.];
```

Figure 21. Output of the PRINT Statement Showing the Prominence of the Harmonics from the PEAKINFO Call

	Peak1	Peak2	Peak3
userPeakIndex	164	509	845
prominence	7	8	1.222
parentPeak	509	164	1241

As previously stated, the ninth column pertains to the prominence, so we save the prominence values to the vector `h_prom` as in the following code. We then calculate the average of all the prominence values to create a threshold value, `minh_prom`. We set `minh_prom` to be 15% of the found average. The value of 15% was chosen after much trial and error with several windows that displayed incorrect trailing harmonics.

```
h_prom = h_output[,9];  
minh_prom = h_prom[:]*.15;
```

The LOC function is then used as follows to return all harmonic indices relative to `h_prom` that meet our threshold, `minh_prom`. We also use the LOC function to find the harmonics that did not meet the threshold and store them in the vector `rmv_idx`. If all harmonics meet the criteria, we use the ISEMPY function to verify this and notify us; otherwise the harmonics that we remove are printed to output. Finally, we save the harmonics that passed the criteria back to the vector `h_idx`.

```
temp_idx = loc(h_prom>minh_prom);  
rmv_idx = loc(h_prom<=minh_prom);  
  
if isempty(rmv_idx) then  
    print "no harmonics removed";  
else  
    print(rmv_idx);  
    h_idx = h_output[temp_idx,1];
```

Figure 22 shows window 39 from **Table 1** after the harmonics were tested against the prominence threshold. Notice that the last trailing harmonic was removed. The output now shows the correct harmonics plotted.

Table 1. Plots with Corresponding Window and Prominence. Row 1: Correctly Omitted Fundamental Frequency (F0) Plot. Rows 2–3: Correctly Included Fundamental Frequency Plots.

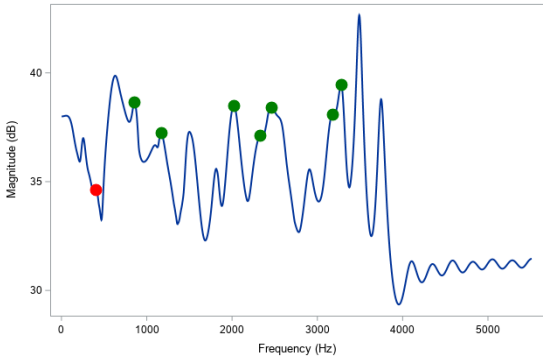
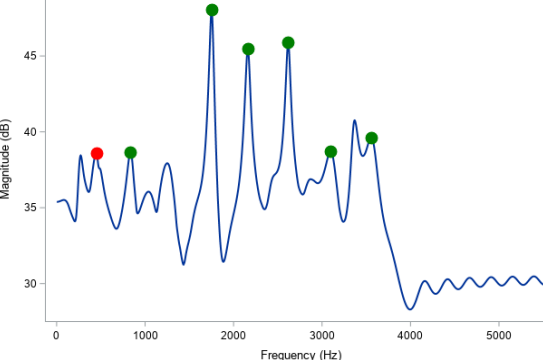
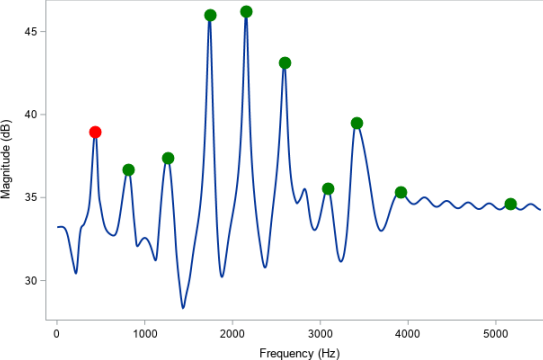
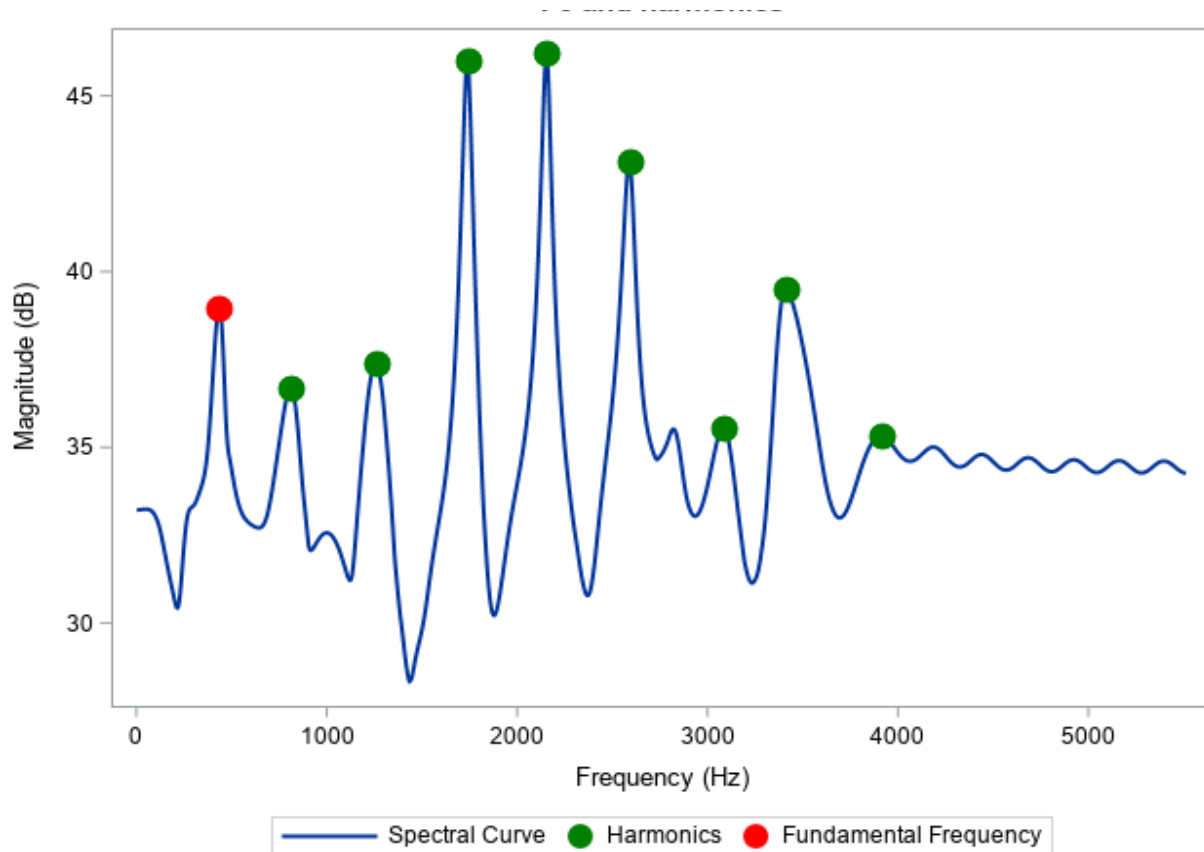
Plot of Detected Fundamental Frequency and Harmonics	Window #	Fundamental Frequency Peak Prominence
	30	0
	38	4
	39	8

Figure 22. Fundamental Frequency and Harmonics of Window 39, after a Harmonic Was Removed for Not Meeting the Prominence Threshold



Queen Bee Piping Example 2

In this example, we show how you can use the [INSTANTTFA](#) function to monitor the health of a beehive by demonstrating how it can potentially detect queen bee piping sounds.

This work is the continuation of Queen Bee Piping Example 1, in which the fundamental frequency and harmonics are found when queen bee piping is detected. Our goal in the current example is to use the INSTANTTFA function to show how the fundamental frequency of the queen bee piping changes over time and to trace the conversation between the queen bee and the hive's worker bees by using the heat map shown in **Figure 19**.

The spectral data set that is used in this example mirrors that of the previous example, with a minor modification. In that example, spectral data underwent an adjustment based on magnitude, ensuring parity in power between higher-frequency harmonics and their lower-frequency counterparts. In contrast, for the current analysis, we do not include the detection of harmonics. Rather, the objective here is to discern the most prominent frequency within each time window and evaluate its correspondence with the fundamental frequencies characteristic of the queen bee piping sound. To facilitate this investigation, the data set in this example lacks the magnitude adjustment that was previously applied to higher frequencies, thereby allowing for a more accurate assessment of

frequency concentration. You can access the data set by using the following code:

```
libname myWork "\YourPathtoBeeData\";
```

The following code opens the data set `myWork.spectral_data`, which contains the spectrum information:

```
proc iml;
  use myWork.spectral_data;
  read all var _NUM_ into spectrum;
  close myWork.spectral_data;

  Fs = 11025;
```

Like the data set `myWork.Spectral_adj` that is used in the previous example, `myWork.spectral_data` is a two-dimensional data set, where the columns represent the total number of windows and the rows represent frequency information. Each observation is the magnitude in dB of the signal for each listed frequency and window. The `DIMENSION` function is used to determine the total number of windows, `num_wins`, and the total number of frequencies, `num_freq`. As in the previous example, we adhere to the predetermined sampling frequency (`Fs`) of 11,025 Hz.

The following code creates two empty variables that will later be used to store the instantaneous frequency and the window of time when the instantaneous frequency occurred:

```
all_wins = .;
all_freqs = .;

spectrows = nrow(spectrum[, 1]);
```

Because we do not have the original phase information, we set the phase to zero before using the IFFT function. Setting the phase to zero causes a delay in the time domain, but because we care only about frequency information, this setting is inconsequential for this example. The following code creates an empty vector that is the size of the total number of rows or total number of frequency bins in each window:

```
phase_zero = j(spectrows, 1, 0);
```

The following code creates multiple variables that will be used for filter design. We create a seventh-order ($n = 7$) bandpass Butterworth filter. The low and high cutoff frequencies, `Wc`, are set to 300 and 500 Hz, respectively, because those are the frequency ranges where we know that queen bee piping occurs. We use the `DFDESIGN` function to get the correct zeros, poles, and gain (`z`, `p`, `k`) and the filter coefficients (`b`, `a`) that meet the filter specifications.

```
/* filter desig vars */
filter_name = "butter";
filter_type = "bandpass";
n = 7;
Wc = (300/Fs)*2 || (500/Fs)*2;
call dfdesign(b, a, z, p, k, filter_name, filter_type, n, Wc);
Ne = spectrows;
```

The following function, DFFREQZZPK, obtains the frequency response of the filter. This function serves a valuable purpose in plotting the filter, facilitating visual confirmation of whether the filter design aligns with the specified criteria.

```
h = DFFreqzZPK(z, p, k, Ne);
```

In the next section of the code, we iterate through windows 40 to 60, which we have identified from the previous example as windows where queen bee piping occurs. For each window, we extract the frequency data and apply a specific formula to reformat the data appropriately for inverse fast Fourier transform (IFFT) usage. The IFFT operation uses the inverse Fourier transform to convert the frequency spectrum data back to the time domain. Because the original phase information is unavailable, we set the phase to zero and incorporate the zero phase into the formula.

```
do i = 40 to 60;
    cur_seg = spectrum[, i];
    new_z = cur_seg # cos(phase_zero) || cur_seg#sin(phase_zero);
    new_signal =ifft(new_z);
```

After applying the IFFT, we then apply the filter that we previously designed as follows. We use a zero-phase filter to avoid introducing any time delay. This enables us to plot the instantaneous frequency against the fundamental frequencies that are found in the previous example more easily.

```
y = dffilt(new_signal, b, a, "zerophase");
```

Now that we have applied the filter, we are ready to use the INSTTTFA function to find the instantaneous frequency of the current time window. In the following code, we set the METHOD= option to 1 to use the first moment method (for more information, see the section First Moment Method). We set the OPT= option to include the sampling frequency Fs, the method, and the window length that we want to use. We use a fairly large window size because the data are already segmented into windows, where each iteration in the DO loop is working on a window of about 0.15 seconds, so there is no advantage in using a smaller window size.

```
method = 1; /* Set method to first moment */
opt = Fs // method // (win_len-1);
moment_freqs1 = instantTFA(y, "freq", opt);
```

The following code appends the instantaneous frequency output to a vector, along with the current window. These vectors are then saved to the data set Mywork.INSTTTFA_Freq so that we can compare results from the INSTTTFA function to the fundamental frequency results from the previous example, as shown in **Table 2**.

```
all_wins = all_wins // i;
all_freqs = all_freqs // moment_freqs1[1,1];
end;

create Mywork.INSTTTFA_Freq var { all_wins, all_freqs};
append;
close Mywork.INSTTTFA_Freq;
quit;
```

Table 2. Comparison of Fundamental Frequencies Using the PEAKLOC and INSTANTTFA Functions

Time Window	Fundamental Frequency with the PEAKLOC Function	Instantaneous Frequency with INSTANTTFA Function
40	419.8975	414.13
41	425.2808	432.9773
42	419.8975	426.3677
43	417.2058	418.9091
44	419.8975	423.9296
45	425.2808	427.4438
46	430.6641	428.1907
47	433.3557	432.0843
48	438.739	436.4742
49	446.814	439.4499
50	449.5056	435.3967
51	452.1973	434.0282
52	452.1973	427.0931
53	457.5806	424.1487
54	457.5806	433.1863
55	452.1973	439.0628

Conclusion

This paper has presented a number of digital signal processing tools that have been recently added to SAS IML software. We discussed the methodology behind the tools and showed how to use them, and we presented several real-world examples. Empirical mode decomposition (EMD) was used for feature extraction with EEG data. EMD and the Hilbert-Huang transform (HHT) were used to decompose musical signals. We demonstrated how the PEAKLOC function and PEAKINFO call can be used to detect queen bee piping for monitoring beehive health and then expanded on that example by using the INSTANTTFA function. These examples demonstrate how you can apply powerful signal processing techniques to a diverse set of problems.

References

Champion, J. 2024. *Alesis Sanctuary QCard AcousticBas C2*. Free Wave Samples. Available at <https://freewavesamples.com>.

Champion, J. 2024. *1980s Casio Flute C5*. Free Wave Samples. Available at <https://freewavesamples.com>.

Grenander, U. 1959. "Probability and Statistics: The Harald Cramér Volume."

Liao, Y. 2020. "Noninvasive Beehive Monitoring through Acoustic Data Using SAS Event Stream Processing and SAS Viya." *Proceedings of the SAS Global Forum 2020 Conference*. Cary, NC: SAS Institute Inc. <https://support.sas.com/resources/papers/proceedings20/4509-2020.pdf>.

Nikolas. 2024. *EEG Dataset*. Kaggle. Available at <https://www.kaggle.com/datasets/samnikolas/eeg-dataset>.

Nuttall, A. 1981. "Some Windows with Very Good Sidelobe Behavior." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29:84–91.

SAS Institute Inc. (2024). *SAS IML: Language Reference*. https://pubshelpcenter.unx.sas.com/test/doc/en/pgmsascdc/v_052/casimllang/titlepage.htm.

Release Information Content Version: 1.0 July 2024.

Trademarks and Patents SAS Institute Inc. SAS Campus Drive, Cary, North Carolina 27513

SAS[®] and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. R indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

To contact your local SAS office, please visit: sas.com/offices

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.
® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © SAS Institute Inc. All rights reserved.

