2

De-Noising Audio Signals Using MATLAB Wavelets Toolbox

Adrian E. Villanueva- Luna[1], Alberto Jaramillo-Nuñez[1],
Daniel Sanchez-Lucero[1], Carlos M. Ortiz-Lima[1],
J. Gabriel Aguilar-Soto[1], Aaron Flores-Gil[2] and Manuel May-Alarcon[2]
[1]Instituto Nacional de Astrofisica, Optica y Electronica (INAOE)
[2]Universidad Autonoma del Carmen (UNACAR)
Mexico

1. Introduction

Based on the fact that noise and distortion are the main factors that limit the capacity of data transmission in telecommunications and that they also affect the accuracy of the results in the signal measurement systems, whereas, modeling and removing noise and distortions are at the core of theoretical and practical considerations in communications and signal processing. Another important issue here is that, noise reduction and distortion removal are major problems in applications such as; cellular mobile communication, speech recognition, image processing, medical signal processing, radar, sonar, and any other application where the desired signals cannot be isolated from noise and distortion.

The use of wavelets in the field of de-noising audio signals is relatively new, the use of this technique has been increasing over the past 20 years. One way to think about wavelets matches the way how our eyes perceive the world when they are faced to different distances. In the real world, a forest can be seen from many different perspectives; they are, in fact, different scales of resolution. From the window of an airplane, for instance, the forest cover appears as a solid green roof. From the window of a car, the green roof gets transformed into individual trees, and if we leave the car and approach to the forest, we can gradually see details such as the trees branches and leaves. If we had a magnifying glass, we could see a dew drop on the tip of a leaf. As we get closer to even smaller scales, we can discover details that we had not seen before. On the other hand, if we tried to do the same thing with a photograph, we would be completely frustrated. If we enlarged the picture "closer" to a tree, we would only be able to see a blurred tree image; we would not be able to spot neither the branch, nor the leaf, and it would be impossible to spot the dew drop. Although our eyes can see on many scales of resolution, the camera can only display one at a time.

In this chapter, we introduce the reader to a way to reduce noise in an audio signal by using wavelet transforms. We developed this technique by using the wavelet tool in MATLAB. A Simulink is used to acquire an audio signal and we use it to convert the signal to a digital format so it can be processed. Finally, a Graphical User Interface Development Environment (GUIDE) is used to create a graphical user interface. The reader can go through this chapter systematically, from the theory to the implementation of the noise reduction technique.

We will introduce in the first place the basic theory of an audio signal, the noise treatment fundamentals and principles of the wavelets theory. Then, we will present the development of noise reduction when using wavelet functions in MATLAB. In the foreground, we will demonstrate the usefulness of wavelets to reduce noise in a model system where Gaussian noise is inserted to an audio signal. In the following sections, we will present a practical example of noise reduction in a sinusoidal signal that has been generated in the MATLAB, which it is followed by an example with a real audio signal captured via Simulink. Finally, the graphic noise reduction model using GUIDE will be shown.

2. Basic audio theory

Sound is the vibration of an elastic medium, whether gaseous, liquid or solid. These vibrations are a type of mechanical wave that has the capability to stimulate human ear and to create a sound sensation in the brain. In air, sound is transmitted due to pressure variations at a rate of change that is called frequency. The difference between the extreme values of pressure represents its amplitude. Pressure variations in the range of 20 Hz to 20 kHz produce the sound which is audible to the human ear and this is more receptive when it is between 1 kHz to 4 kHz. In physical terms, the sound is a longitudinal wave that travels through the air due to vibration of the molecules. Similar to light, sound waves can be reflected, absorbed, diffracted, or refracted.

Audio signals, which represent longitudinal variations of pressure in a medium, are converted into electrical signals by piezoelectric transducers. Transducers convert the energy of a mechanical displacement into an electrical signal, either voltage or current. The main advantage of converting an audio signal into an electrical signal is that the signal can now be processed. An example is an analog signal obtained from the transducer that can be converted into an encoded digital data stream by using an analog-digital converter (ADC) and constitutes digital processing of analog signals. Alternatively, if a digital-analog converter (DAC) is applied to a digital data stream, the audio signal transmits through an amplifier and a speaker. The process is show schematically in Figure 1, which identifies the important steps in digital audio processing.

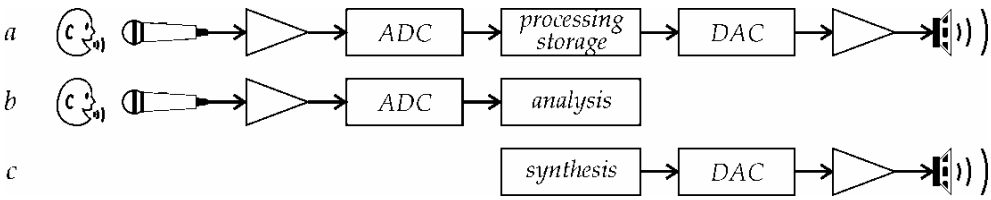


Fig. 1. Shows the process of digital processing of three types of audio signal. Part (*a*) represents a complete digital audio processing comprising (from left to right) a microphone, amplifier, ADC, digital processing material, DAC, amplifying section and speaker; an audio recognition system in (*b*), and a set of audio synthesis (*c*)

2.1 Recording audio signals in Simulink/MATLAB

Once in the digital domain, these signals can be processed, transmitted or stored. We found that the Audio Device block in Simulink enables experimentation and processing of digital signals. The From Audio Device block reads audio data from an audio device in real time.

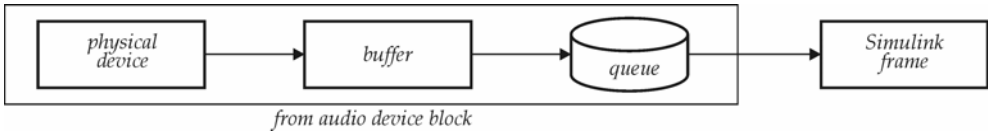


Fig. 2. Shows the process identifies the main steps in a digital audio processing system based in Simulink software

The From Audio Device block buffers the data from the audio device by means of using the process illustrated by Figure 2. We selected the block MATLAB Simulink audio and multimedia file block in order to save the audio acquired by a given time. Figure 3 shows the From Audio Device GUI, where we selected a 5 second queue period. At the start of the simulation, the audio device writes input data to a buffer. When the buffer is full, the From Audio Device block writes the contents of the buffer to the queue. The size of this queue can be specified in the queue duration (seconds) parameter. As the audio device appends audio data to the bottom of the queue, the From Audio Device block pulls data from the top of the queue to fill the Simulink frame. We used this file to make our de-noise method using wavelets.

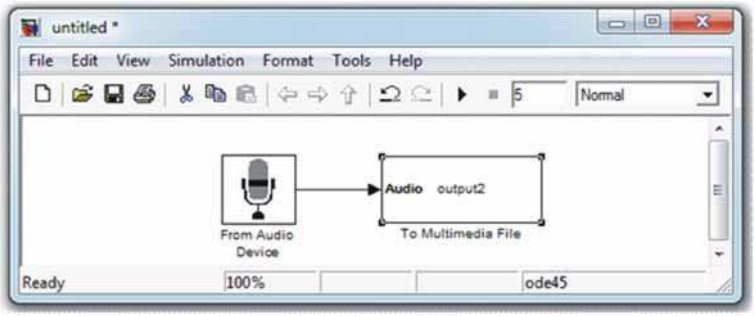


Fig. 3. Shows block diagram of audio acquire

We used the *wavread* function. It loads a: WAVE file specified by the string filename, returning the sampled data in *y*. If the filename does not include an extension, *wavread* appends a .wav extension. Example code is: *[x,Fs,nbits]= wavread('filename') where the function returns the filename with* the number of bits per sample (nbits).

```
[x,Fs,nbits]= wavread ('voice');
y = awgn(x,10,'measured');
```

For example, *wavwrite(y,Fs,'filename')* writes the data stored in the variable *y* to a WAVE file called 'filename'. The data have a sample rate, Fs, in Hz and is assumed 16-bit.

```
wavwrite(y,Fs,'noisyvoice')
wavwrite(xd,Fs,'voice5')
```

3. Basic noise theory

Noise is defined as an unwanted signal that interferes with the communication or measurement of another signal. A noise itself is an information-bearing signal that conveys information regarding the sources of the noise and the environment in which it propagates.

There are many types and sources of noise or distortions and they include:

1. *Electronic noise* such as thermal noise and shot noise,
2. *Acoustic noise* emanating from moving, vibrating or colliding sources such as revolving machines, moving vehicles, keyboard clicks, wind and rain,
3. *Electromagnetic noise* that can interfere with the transmission and reception of voice, image and data over the radio-frequency spectrum,
4. *Electrostatic noise* generated by the presence of a voltage,
5. *Communication channel* distortion and fading and
6. *Quantization noise* and lost data packets due to network congestion.

Signal distortion is the term often used to describe a systematic undesirable change in a signal and refers to changes in a signal from the non-ideal characteristics of the communication channel, signal fading reverberations, echo, and multipath reflections and missing samples. Depending on its frequency, spectrum or time characteristics, a noise process is further classified into several categories:

1. *White noise*: purely random noise has an impulse autocorrelation function and a flat power spectrum. White noise theoretically contains all frequencies in equal power.
2. *Band-limited white noise*: Similar to white noise, this is a noise with a flat power spectrum and a limited bandwidth that usually covers the limited spectrum of the device or the signal of interest. The autocorrelation of this noise is sinc-shaped.
3. *Narrowband noise*: It is a noise process with a narrow bandwidth such as 50/60 Hz from the electricity supply.
4. *Coloured noise*: It is non-white noise or any wideband noise whose spectrum has a non-flat shape. Examples are pink noise, brown noise and autoregressive noise.
5. *Impulsive noise*: Consists of short-duration pulses of random amplitude, time of occurrence and duration.
6. *Transient noise pulses*: Consist of relatively long duration noise pulses such as clicks, burst noise etc.

3.1 Signal to noise ratio

The signal-to-noise ratio (*SNR*) is commonly used to assess the effect of noise on a signal. This measurement is based on an additive noise model, where the quantized signal $x_q[n]$ is a superposition of the unquantized, undistorted signal $x[n]$ and the additive quantization error $e[n]$. The ratio between the signal powers of $x[n]$ and $e[n]$ defines the *SNR*. To capture the wide range of potential *SNR* values and to consider the logarithmic perception of loudness in humans, *SNR* generally given in a logarithmic scale, in decibels (dB)

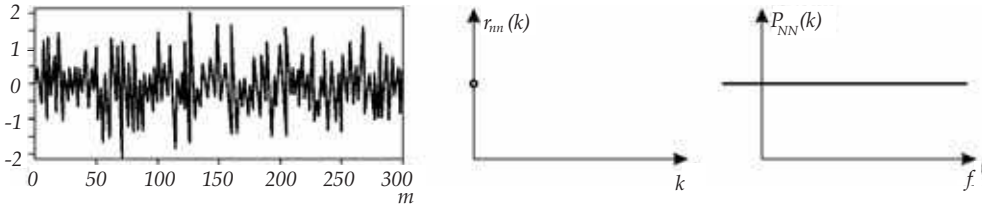
$$SNR_{dB} = 10 * \log_{10} \frac{\sigma_x^2}{\sigma_e^2}, \tag{1}$$

Where, $\sigma 2x$ and σ^2_e are the powers of $x[n]$, and $e[n]$, respectively. Specifically for the assessment of quantization noise, *SNR* is often labeled as the signal to quantization-noise ratio (SQNR).

3.2 White noise

Shown in Figure 4, white noise is defined as an uncorrelated random noise process with equal power at all frequencies. Random noise has the same power at all frequencies in the range of ∞ it would necessarily need to have infinite power, and it is therefore an only a

theoretical concept. However, a band-limited noise process with a flat spectrum covering the frequency range of a band-limited communication system is practically considered a white noise process.



(a) (b) (c)

Fig. 4. Shows an illustration of (a) white noise time-domain signal, (b) its autocorrelation function is a delta function, and (c) its power spectrum is a constant function of frequency

3.3 Additive White Gaussian Noise Model (AWGN)

In classical communication theory, it assumed that the noise is a stationary additive white Gaussian (AWGN) process. Although for some problems this is a valid assumption and leads to mathematically convenient and useful solutions, in practice, the noise is often time-varying, correlated and non-Gaussian. This is particularly true for impulsive-type noise and for acoustic noise, which is non-stationary and non-Gaussian and hence cannot be modeled using the AWGN assumption.

3.4 Adding noise using MATLAB

How to do it with MATLAB: In the Communication Toolbox in MATLAB is possible to find the function *awgn*. This function means adding Gaussian white noise to signal.

Syntax
y = awgn(x,snr)
y = awgn(x,snr,'measured')

y = awgn(x,snr) adds white Gaussian noise to the vector signal *x*. The scalar *SNR* specifies the signal-to-noise ratio per sample, in dB. If x is complex, *awgn* adds complex noise. This syntax assumes that the power of *x* is 0 dBW.
y = awgn(x,snr,'measured') is the same as *y = awgn(x,snr)*, except that *awgn* measures the power of *x* before adding noise.
Figure 5 shows one example of how to add white noise to the sinusoidal signal using the communication toolbox of MATLAB. We generate the interval of the signal (k), considering some frequency (w), then we generate a sinusoidal function with these parameters, considering this function with the *x* vector. Now we generate a Gaussian white noise and add to the sinusoidal function and plot.

```
k = 0:9.0703e-005:5;
w=500*pi;
h=w.*k;
x = sin(h); % Create sinus signal.
y = awgn(x,10,'measured'); % Add white Gaussian noise.
figure(1)
plot(k,y)
```

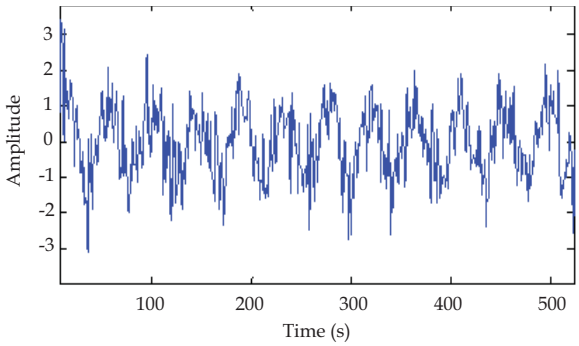


Fig. 5. Shows adding Gaussian white noise to sine signal

4. Wavelets theory

Wavelets are used in a variety of fields including physics, medicine, biology and statistics. Among the applications in the field of physics, there is the removal of noise from signals containing information. There are different ways to reduce noise in audio. (Johnson et al., 2007) demonstrated the application of the Bionic Wavelet Transform (BWT), an adaptive wavelet transform derived from a non-linear auditory model of the cochlea, to enhance speech signal. (Bahoura & Rouat, 2006) proposed a new speech enhancement method based on time and scale adaptation of wavelet thresholds. (Ching-Ta & Hsiao-Chuan Wang, 2003 & 2007) proposed a method based on critical-band decomposition, which converts a noisy signal into wavelet coefficients (WCs), and enhanced the WCs by subtracting a threshold from noisy WCs in each subband. Additionally, they proposed a gain factor in each wavelet subband subject to a perceptual constraint. (Visser et al., 2003) has presented a new speech enhancement scheme by spatial integration and temporal signal processing methods for robust speech recognition in noisy environments. It further de-noised by exploiting differences in temporal speech and noise statistics in a wavelet filter bank. (Képesia & Weruaga, 2006) proposed new method for time–frequency analysis of speech signals. The analysis basis of the proposed Short-Time Fan-Chirp Transform (FChT) defined univocally by the analysis window length and by the frequency variation rate, that parameter predicted from the last computed spectral segments. (Li et al., 2008) proposed an audio de-noising algorithm based on adaptive wavelet soft-threshold, based on the gain factor of linear filter system in the wavelet domain and the wavelet coefficients teager energy operator in order to progress the effect of the content-based songs retrieval system. (Dong et al., 2008) has proposed a speech de-noising algorithm for white noise environment based on perceptual weighting filter, which united the spectrum subtraction and adopted auditory perception properties in the traditional Wiener filter. (Shankar & Duraiswamy, 2010) proposed an audio de-noising technique based on biorthogonal wavelet transformation.

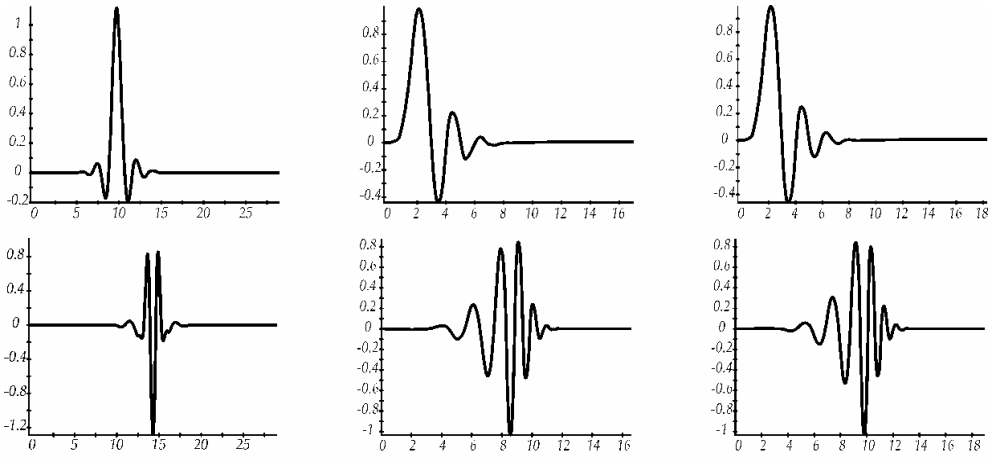
Wavelets are characterized by scale and position, and are useful in analyzing variations in signals and images in terms of scale and position. Because of the fact that the wavelet size can vary, it has advantage over the classical signal processing transformations to simultaneously process time and frequency data. The general relationship between wavelet scales and frequency is to roughly match the scale. At low scale, compressed wavelets are used. They correspond to fast-changing details, that is, to a high frequency. At high scale,

the wavelets are stretched. They correspond to slow changing features, that is, to a low frequency.
The wavelet only has a time domain representation as the wavelet function ψ (*t*). The mother wavelet is scaled (or dilated) by a factor of *a* and translated (or shifted) by a factor of *b* to give

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right). \tag{2}$$

Wavelets are defined by the wavelet function ψ (*t*) (i.e. the mother wavelet) and scaling function φ (*t*) (also called father wavelet) in the time domain. The wavelet function is in effect a band-pass filter and scaling it for each level halves its bandwidth. This creates the problem that in order to cover the entire spectrum, an infinite number of levels would be required. The scaling function filters the lowest level of the transform and ensures that the entire spectrum is covered. For a wavelet with compact support, φ (*t*) can be considered finite in length and is equivalent to the scaling filter *g*.
Wavelets can be scaled and copied (known as "daughter wavelets") of a finite-length or fast-decaying oscillating waveform (known as the "mother wavelet"). Wavelet transforms have advantages over traditional Fourier transforms for representing functions that have discontinuities and sharp peaks, and for accurately deconstructing and reconstructing finite, non-periodic and/or non-stationary signals.
Wavelet transforms are classified into two broad groups: discrete wavelet transforms (DWTs) and continuous wavelet transforms (CWTs). Note that both DWT and CWT are continuous-time (analog) transforms and can represent continuous-time (analog) signals. CWTs operate over every possible scale and translation whereas DWTs use a specific subset of scale and translation values or representation grid. When wavelet-functions coefficients are expressed as z-transform, then the number of zeros at π corresponds to the number of vanishing moments. S, o *p* zeros in π means *p* vanishing moments.
Having *p* vanishing moments means that wavelet-coefficients for *p-th* order polynomial will be zero. That is, any polynomial signal up to order *p-1* can be represented completely in scaling space. In theory, more vanishing moments meaning that scaling function can represent more signals that are complex accurately, *p* which it is also called as the accuracy of the wavelet.
Daubechies wavelets are a family of orthogonal wavelets defining a discrete wavelet transform and they are characterized by a maximal number of vanishing moments for some given support. With each wavelet type of this class, a scaling function (also called father wavelet) generates an orthogonal multi-resolution analysis.
In general the Daubechies wavelets are chosen to have the highest number *A* of vanishing moments, (this does not imply the best smoothness) for given support width *N=2A*, and between the 2^{A-1} possible solutions the number one is chosen whose scaling filter has extreme phase. The wavelet transform is also easy to put into practice by using the fast wavelet transform. Daubechies wavelets are widely used in solving a broad range of problems, e.g. self-similarity properties of a signal or fractal problems, signal discontinuities, etc.
Coiflets are discrete wavelets designed by Ingrid Daubechies, at the request of Ronald Coifman, to have scaling functions with vanishing moments. The wavelet is near symmetric

their wavelet functions have $N/3$ vanishing moments and scaling functions $N/3-1$, they are used in many applications using Calderón-Zygmund Operators.
Db 10 and Db 9 are asymmetric, orthogonal and biorthogonal, Coif 5 is near symmetric, orthogonal and biorthogonal. Figure 6 shows the scale and wavelet function of Coiflet 5, Daubechies 9 and 10.



(a) (b) (c)

Fig. 6. Shows the scale function (left) and wavelet function (right) of (a) Coiflet 5, (b) Daubechies 9 and (c) Daubechies 10

4.1 One stage filtering: Approximations and details

For many signals, the low-frequency content is the most important part because it gives to the signal its identity. The high-frequency content, on the other hand, imparts nuance. Consider the human voice. If high-frequency components are removed, the voice sounds different, but the words are still audible and clearly recognized. However, if enough low-frequency components are removed, the resulting audio signal is not clear. In wavelet analysis, approximations are the high-scale, low-frequency components and the details are the low-scale, high-frequency components of the signal. Figure 7 shows the steps for signal decomposition.

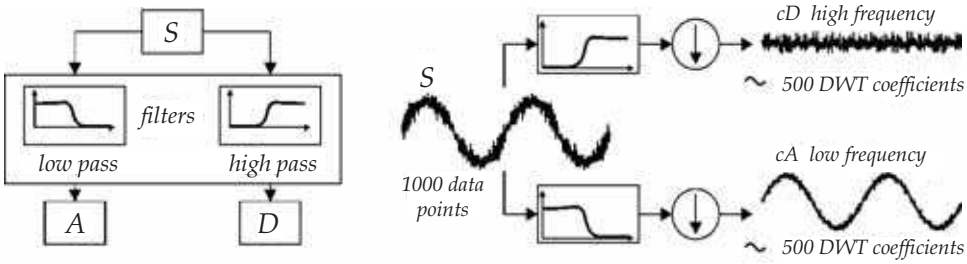


Fig. 7. Show the way to decompose a signal

4.2 Multiple-level decomposition

The decomposition process can be iterated, with successive approximations being decomposed in turn, so that one signal is broken down into many lower resolution components. This is called the wavelet decomposition tree that is shown on figure 8.

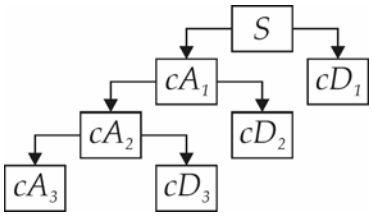


Fig. 8. Shows the process of performing, the multiple decomposition, which it is called wavelet decomposition tree

4.3 Number of levels

Since the analysis process is iterative, in theory it can be continued indefinitely. Realistically, the decomposition can proceed only until the individual details consist of a single sample or pixel. Moreover, the processes include selecting a suitable number of levels based on the nature of the signal, or on a suitable criterion such as *entropy*.

4.4 Wavelet reconstruction

Discrete wavelet transform can be used to analyze, or decompose, signals and images in a process called decomposition or analysis. Conversely, reconstruction or synthesis is the process of assembling hose components back into the original signal without loss of information. While being this transformation, it is desirable to establish its investment, i.e. to return to the original signal from the output tree. This methodology follows reasoning in the opposite direction, i.e. from the coefficients while depending on the number of levels and considering the high frequency (*H'*) and low (*L'*) bands that must be obtained from the reconstructed signal S, shown in figure 9. The mathematical manipulation that affects synthesis is called: the *inverse discrete wavelet transforms* (IDWT). In order to synthesize a signal by using Wavelet Toolbox software, we reconstruct it from the wavelet coefficients.

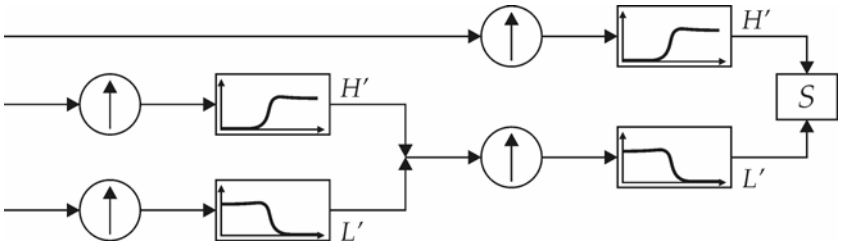


Fig. 9. Show how to reconstruct a signal using wavelets

4.5 Wavelet using MATLAB

We will describe a simple method for wavelet transform of a given signal. You must choose a wavelet function, which is the mother wavelet, as well as determining the value of the

signal wavelet scale, using the command *wname*. In this example we use a wavelet *coif5* level 10.

```
wname = 'coif5'; lev = 10;
```

In most of the signals of low frequency components that give the signal most of its information, while the high frequency components are responsible for incorporating particular features, that is why we subdivide the components of a signal into two categories: approaches (low frequencies) and details (high frequencies). To perform this decomposition the *wpdec* command is used which is responsible for creating a tree that consists of a vector in this case it is the signal to break down "*y*" the level to be decomposed *lev = 10* and the type of wavelet used, which in this case is coiflet 5 and it is stored as a tree in the *tree* variable:

```
tree = wpdec(y,lev,wname);
```

We generate the coefficients of the decomposition of the signal using the *wpcoef* command where needed as the variable information and the number of tree node that has a better performance in the tree that was generated, which in this case is two.

```
det1 = wpcoef(tree,2);
```

We determine noise threshold by using *wpbmpen* command that returns a global threshold *THR* for de-noising. *THR* obtained by a wavelet packet coefficients selection rule by means of using a penalization method provided by Birge-Massart. T (in our case tree) is a wavelet packet tree corresponding to the wavelet packet decomposition of the signal or image to be de-noised. *SIGMA* is the standard deviation of the zero mean Gaussian white noise in the de-noising model. *alpha* is a tuning parameter for the penalty term, which must be a real number greater than 1. The sparsity of the wavelet packet representation of the de-noised signal or image grows with *alpha*. Typically *alpha* = 2.

```
sigma = median(abs(det1)) /0.6745;
alpha = 2;
thr = wpbmpen(tree,sigma,alpha);
```

We use command *wpdencmp* that performs a de-noising or compression process of a signal, when using wavelet packet. The ideas and the procedures for de-noising and compression by using wavelet packet decomposition are the same as those used in the wavelets framework. *Wpdencmp (TREE,SORH,CRIT,PAR,KEEPAPP)* has the same output arguments, using the same options as above, but which were obtained directly from the input wavelet packet tree decomposition *TREE* of the signal to be de-noised or compressed. *SORH* ('*s*' *or* '*h*') stands for soft or hard thresholding. Best decomposition performed using entropy criterion defined by string *CRIT* and parameter *PAR*. Threshold parameter is also PAR. In addition, if *CRIT* = '*nobest*' no optimization is done, and the current decomposition is thresholded. If *KEEPAPP= 1*, approximation coefficients cannot be thresholded; otherwise, they can be:

```
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
```

All those parameters help us to de-noise our signals by using wavelets.

5. Sine signal example

With MATLAB, it is possible to process noisy signals containing certain information, such as an audio one, in order to reduce the quantity of noise contained in it. Non-periodicity

characterizes an audio signal, which is composed by a large number of different frequencies signals. This feature allows the use of conventional methods such as digital filters to eliminate noise mixed into the signal.
In this section, we will introduce the treatment of noise in audio signals by using wavelet transforms, which are included as a tool in MATLAB. To do this we will start by using a sine wave signal, which is periodic and presents a well-known behavior.
First, it is necessary to define the time interval *k*, and to define the angular frequency *w*.

```
k = 0:9.0703e-005:5;
w=500*pi;
```

The variable *h* represents the argument of the sine function *x* in the next two instructions

```
h=w.*k;
x = sin(h);
```

Once we have defined the signal to be treated *x*, it is necessary to generate the noise source in order to be mixed with the original signal. One of the most common sources of noise is Gaussian white noise, which contains all frequencies. Thus to generate white Gaussian noise we use the *awgn* instruction described in section 3.4.

```
y = awgn(x,0,'measured');
wname = 'coif5'; lev = 10;
tree = wpdec(y,lev,wname);
det1 = wpcoef(tree,2);
sigma = median(abs(det1)) /0.6745;
alpha = 2;
thr = wpbmpen(tree,sigma,alpha);
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
```

Correlation between both signals, original and filtered one, is the parameter to compare them. We used the command *crosscorr*, which computes and plots the sample cross-correlation function *XCF* between two univariate, stochastic time series. To plot the *XCF* sequence without the confidence bounds, set nSTDs = 0.

```
D=crosscorr(x,xd);
Finally, we plot three figures.
figure(1)
plot(k,xd)
hold on
plot(k,y,'k')
hold on
plot(k,x,'g')
legend('Denoise signal','Signal with AWGN','Original signal');
figure(2)
plot(z,D)
legend('Correlation @ 0');
```

The sinusoidal waveform *x* represents the signal to which is added white Gaussian noise; this will subsequently be reduced to recover the sine wave.
We evaluate three wavelets families, Symlets 2 to 8, Daubechies 2 to 10 and Coiflet 1 to 5 by using the program presented above. Our results show that Coiflet 5 and Daubechies 9 and 10 are the best options for better de-noised in our signals. We choose cross correlation to

evaluate the best performance; the cross correlation number that we select is the maximum at zero. Figure 10 shows the original signal sinus waveform. Part (*b*) is an extension of two periods of part (*a*). We choose a frequency of 250 Hz due to voice is in that range.

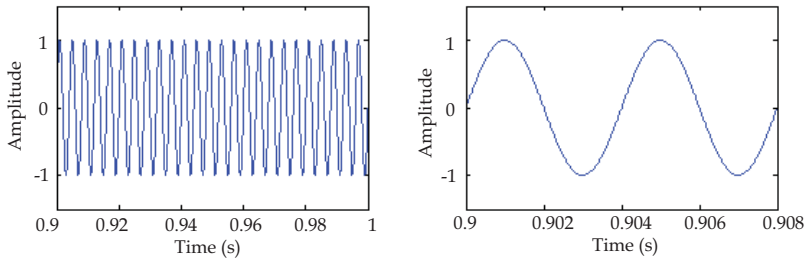


Fig. 10. Shows original sinus waveform. The right one shows a zoom view of left figure

We add white Gaussian noise to sine waveform represented in Figure 11 and the resulting graph is the plot shown in Figure 11. We set *SNR* to *AWGN=10*

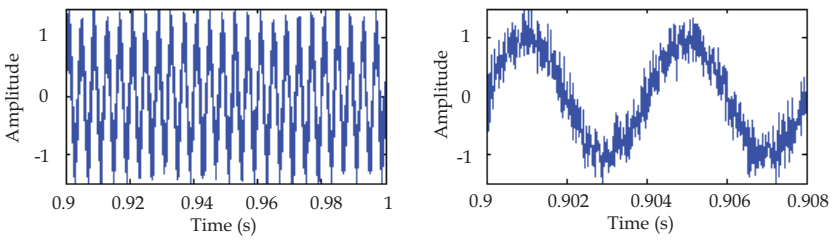


Fig. 11. Shows sine waveform with *AWGN=10*.

After running the program, we find the cross correlation *D* between original signal and signal de-noise. Figure 12 presents the result

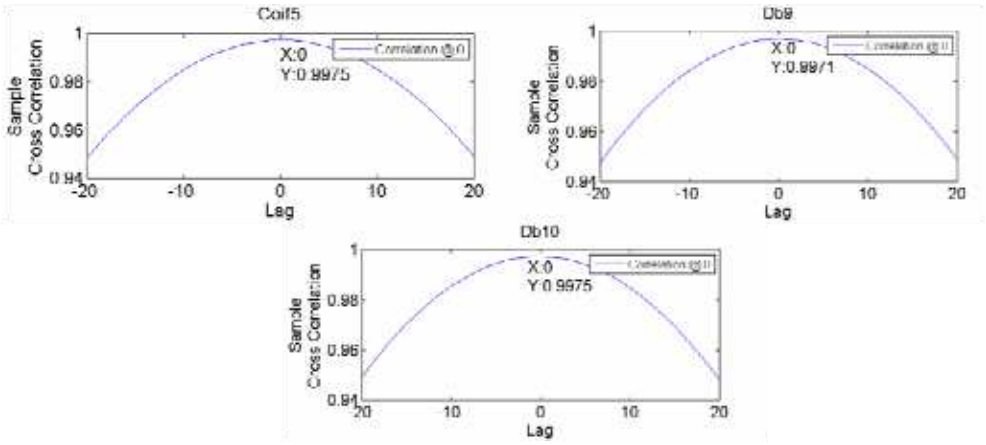


Fig. 12. Shows cross correlation of original signal and signal de-noise

Figure 13 shows the plot of a signal after its processing method using Coiflet 5.

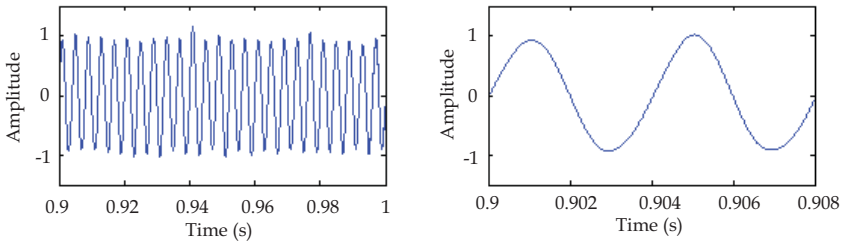


Fig. 13. Shows signal after processing method of de-noise using Coiflet 5

Evaluation of the resultant waveform *xd* presented in Figure 14 is compared with the original sine function *x* form Figure 11 gives numbers higher than 99% as presented in Figure 13. These results support the reliability of the proposed method for de-noising.

6. Audio signal example

As it has been described in section 2.1, to obtain an audio signal, we select the audio block and we connect it to the multimedia file block. This information is saved in a user-specified file with extension .wav and then we use the program to extract this information and process it with wavelets *coif5, db9* and *db10*. Figure 14 shows the original audio signal, audio signal with noise and the recovered signal using the wavelet *coif5*. Figure 15 shows the best correlation using the above wavelets.

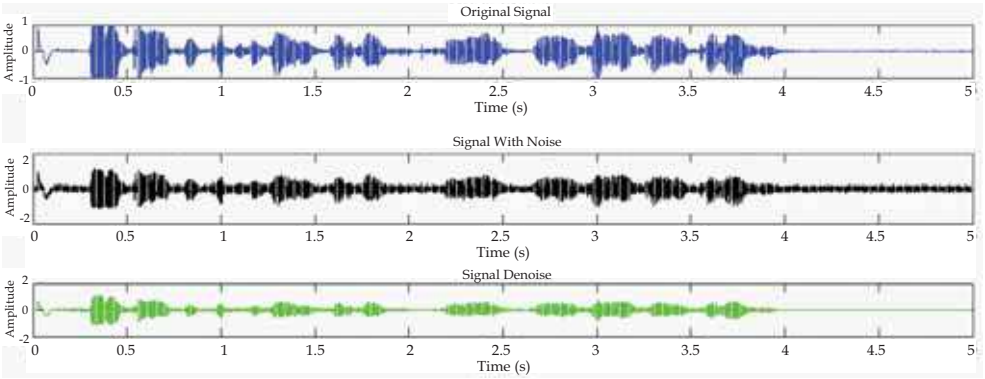


Fig. 14. Shows original audio signal, audio signal with noise and the recovered signal

To make the signal processing we use the next code:

```
clc,close all,clear all
 k = 0:9.0703e-005:5;
w=500*pi;
h=w.*k;
[x,Fs,nbits]= wavread ('voice');
y = awgn(x,10,'measured'); % Add white Gaussian noise.
```

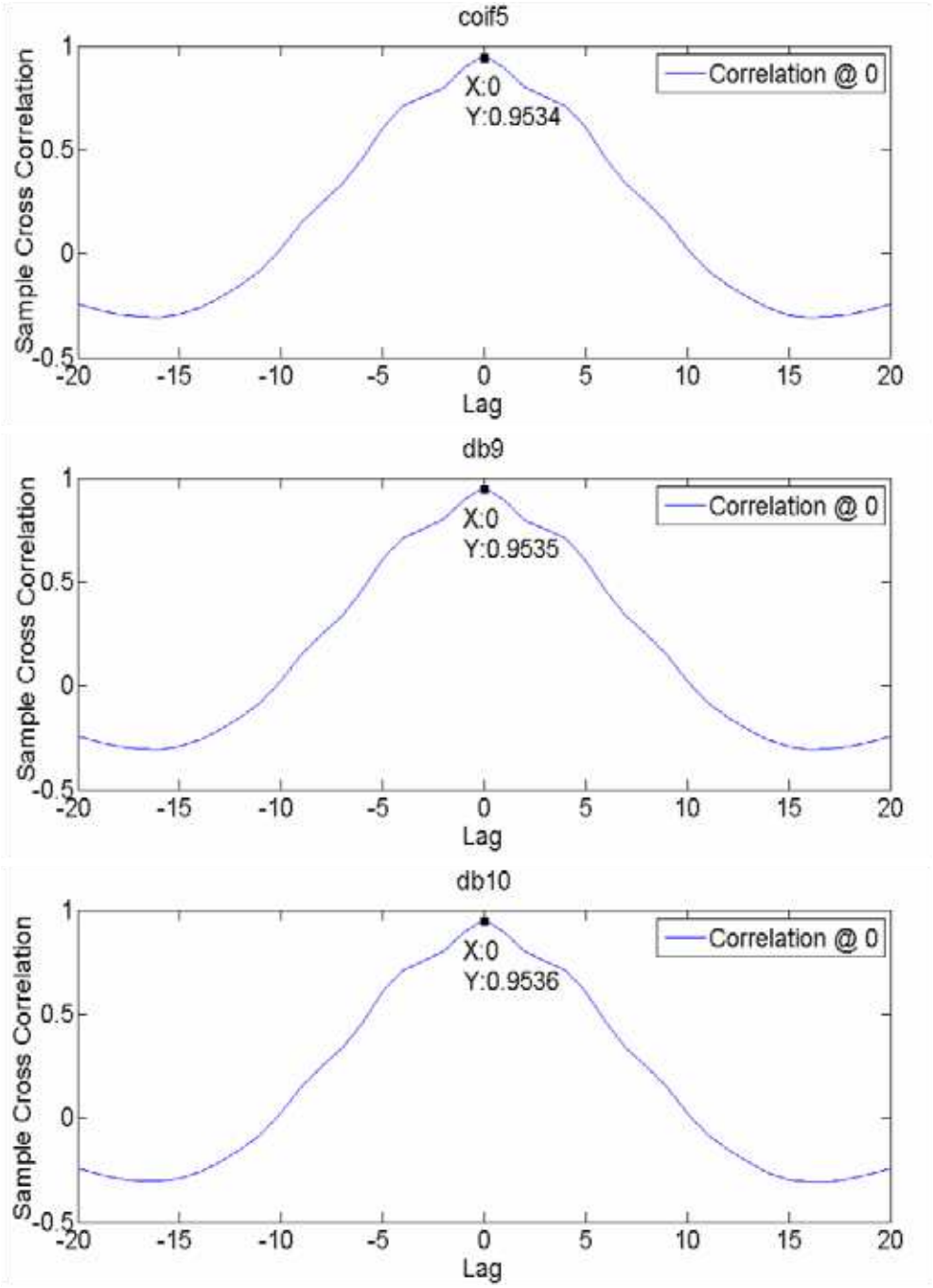


Fig. 15. Shows cross-correlation for determine the best result after noise reduction

```
wavwrite(y,Fs,'noisyvoice')
wname = 'coif5'; lev = 10;
tree = wpdec(y,lev,wname);
det1 = wpcoef(tree,2014);
 sigma = median(abs(det1))/0.6745;
alpha = 1.8;
thr = wpbmpen(tree,sigma,alpha)
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
D=crosscorr(x,y);
 z=-20:1:20;
 figure(1)
subplot(311)
plot(k,x)
xlabel('time')
ylabel('Amplitude')
title('original signal');
subplot(312)
plot(k,y,'k')
xlabel('time')
ylabel('Amplitude')
title('signal with noise');
subplot(313)
plot(k,xd,'g')
xlabel('time')
ylabel('Amplitude')
title('signal denoise');
figure(2)
plot(z,D)
title('coif5')
legend('Correlation @ 0');
wavwrite(xd,Fs,'voice5')
```

In signal processing, analogue-to-digital converters also suffer linearity errors, they add noise, distortion, and introduces quantization error due to the precision of their voltage sampling process. The result of all this is a computerized sequence of samples that may not be as closely related to the real-world sound as you might expect. Do not be surprised when high-precision analysis or measurements are unrepeatable due to noise, or if delicate changes made to a sampled audio signal are undetectable to the naked ear upon replay.

7. Graphical Interface using GUIDE

For better understanding of the content of this chapter, we have developed a graphical interface, only in the case of a sine wave. We used GUIDE of MATLAB to build this interface, you need pop-up menu, slider, edit text, four push button and four graphics (axes), after all that is placed in your figure you need to program each item. Figure 16 shows how all of this should be seen.

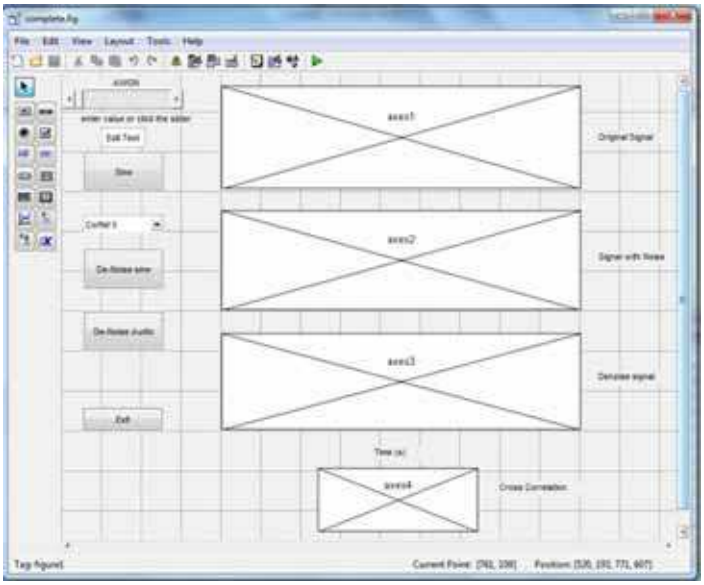


Fig. 16. Shows graphical interface using GUIDE

First you need to configure axes, you do this by double clicking on the graphic, this will show you an inspector on figure 17, you need to change *nextplot from replace to replacechildren* that means to remove all child objects when *HandleVisibility* property is set to an on function and then to reset figure *NextPlot* property to an add function. To set axes limits you need to change *Xlimit* from 0.20 to 0.24, and in our case, *Ylimit* from -1 to 1.

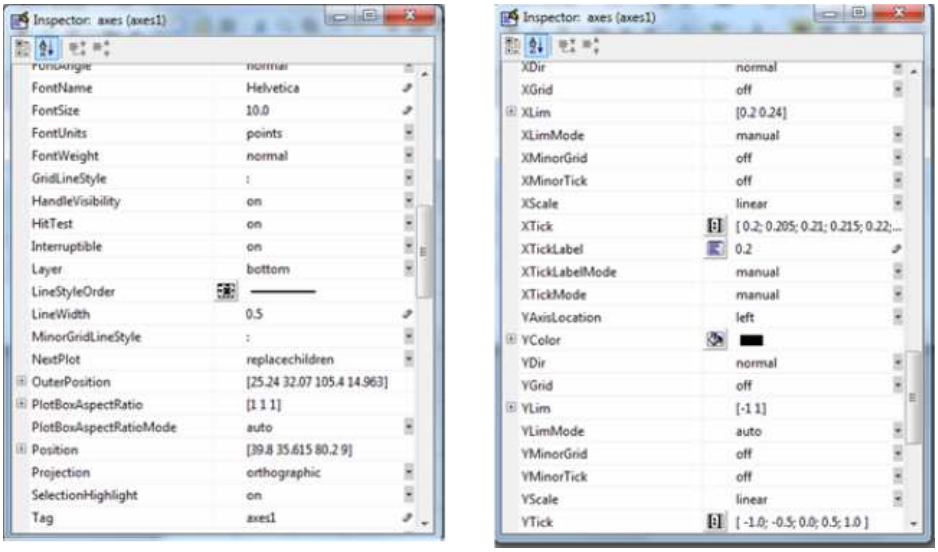


Fig. 17. Shows configurations of axes using inspector

In our case, a slider is coupled to an edit text component so that: The edit text displays the current value of the slider. The user can enter a value into the edit text box and cause the slider to update to that value. Both components update the appropriate model parameters when activated by the user. Our slider is called *AWGN* (average white generator noise) and it is from 0 to 10 *SNR*.

```
function complete_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for example5
handles.output = hObject;
 clc
% initalize error count and use edit1 object's userdata to store it.
data.number_errors = 0;
set(handles.edit1,'UserData',data)
% Update handles structure
guidata(hObject, handles);
% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
set(handles.edit1,'String',...
  num2str(get(hObject,'Value')));
% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
function edit1_Callback(hObject, eventdata, handles)
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 10.
if isnumeric(val) && length(val)==10 && ...
  val >= get(handles.slider1,'min') && ...
  val <= get(handles.slider1,'max')
  set(handles.slider1,'Value',val);
else
% Retrieve and increment the error count.
% Error count is in the edit text UserData,
% so we already have its handle.
  data = get(hObject,'UserData');
  data.number_errors = data.number_errors+1;
% Save the changes.
  set(hObject,'UserData',data);
% Display new total.
  set(hObject,'String',...
  ['You have entered an invalid entry ',...
   num2str(data.number_errors),' times.']);
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

In the case of the pushbutton 1 that is called sine, and if it only shows sine signal and signal with noise, it needs to be programmed as follow

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 D=crosscorr(x,y);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,z,D)
```

For programming, push button 2, called de-noise sine, this button performs all the processing method, which was described previously. It is necessary to write all this code

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
menu= get(handles.popupmenu1,'Value');
switch menu
   case 1  %case 1: coiflet 5
 % example3 code
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
wname = 'coif5'; lev = 10; %  wavelet that need to change!
tree = wpdec(y,lev,wname);
det1 = wpcoef(tree,2);
sigma = median(abs(det1))/0.6745;
alpha = 2;
thr =wpbmpen(tree,sigma,alpha);
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
D=crosscorr(x,xd);
z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
     set(gca,'XLim',[ 0.2,  0.24], ...
        'YLim',[-1 1]);
  plot(handles.axes4,z,D)
   case 2     %case2 Daubechies 10
```

```
·% example3 code
   case 3     %Case 3: Daubechies 9
·% example3 code
End
```

For programing push button 3, called de-noise audio, this button performs all processing method, which was described previously. It is necessary to write all this code

```
  % --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
menu= get(handles.popupmenu1,'Value');
switch menu
   case 1                                          %case 1: Coiflet 5
·% example4 code
 k = 0:9.0703e-005:5;
 w=500*pi;
 h=w.*k;
 [x,Fs,nbits]= wavread ('voice');
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 wavwrite(y,Fs,'noisyvoice')
wname = 'coif5'; lev = 10;
tree = wpdec(y,lev,wname);
det1 = wpcoef(tree,2);
sigma = median(abs(det1))/0.6745;
alpha = 2;
thr =wpbmpen(tree,sigma,alpha);
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
D=crosscorr(x,xd);
z=-20:1:20;
 plot(handles.axes1,k,x)
 plot(handles.axes2,k,y)
 plot(handles.axes3,k,xd)
 plot(handles.axes4,z,D)
case 2     %case2 Daubechies 10
·% example4 code
   case 3     %Case 3: Daubechies 9
·% example4 code
end
```

For programing push button 4 called exit

```
% --- Executes on button press in pushbutton4.
function pushbutton3_Callback(hObject, eventdata, handles)
close all
```

Figure 18 (left) shows *AWGN 1*, and it plots original signal, in this case sine, and signal with noise, cross correlation between them. Figure 18 (right) shows sine after de-noise and cross correlation between the original signal and the de-noised.

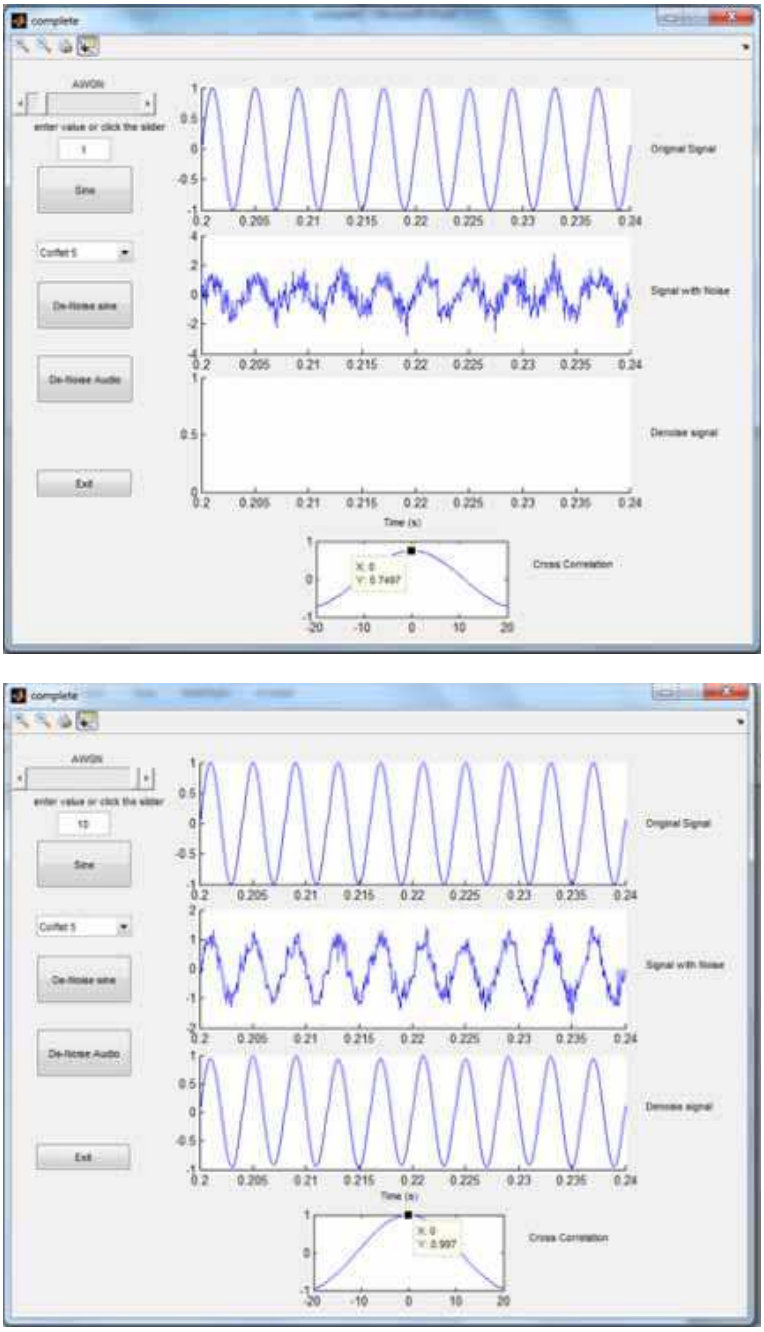


Fig. 18. (up) *AWGN* at 1 *SNR* with a correlation of 0.7497 and (down) sine after de-noise with *SNR* of 10 and correlation of 0.997

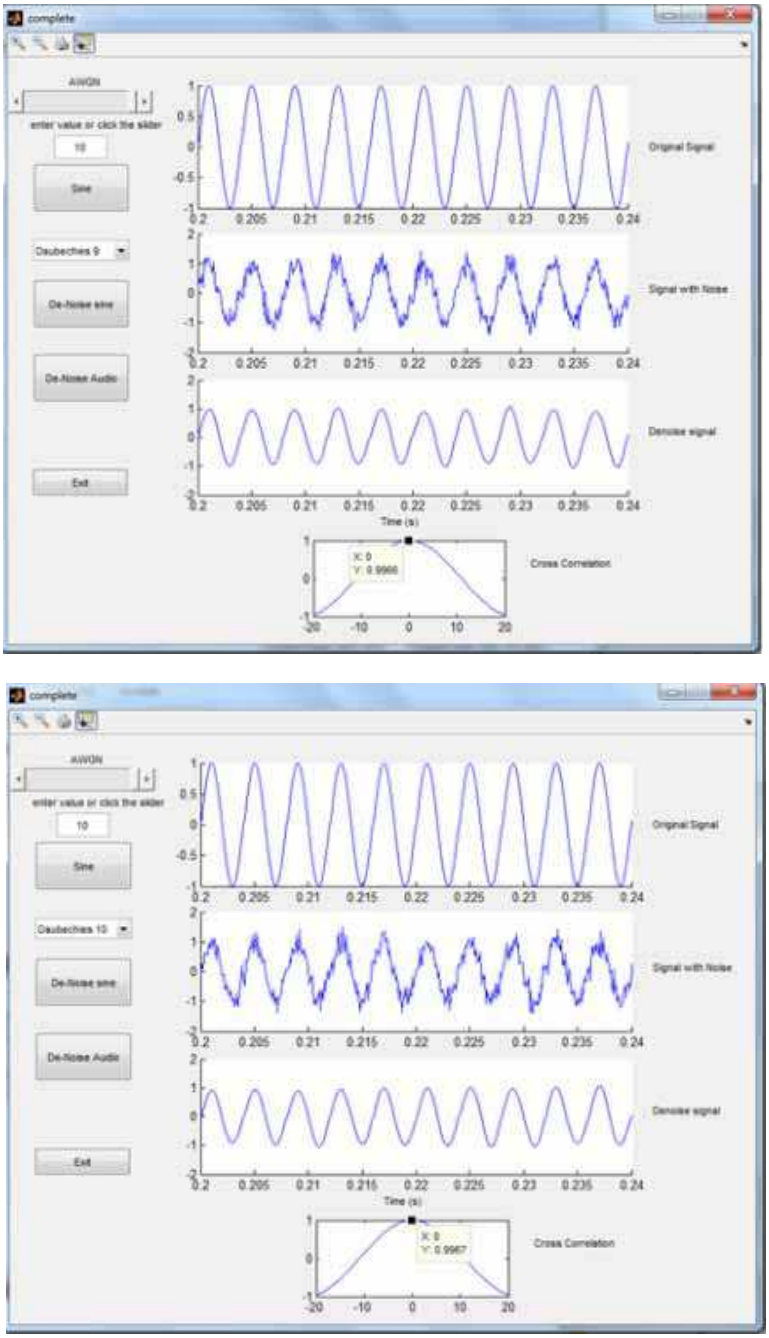


Fig. 19. (up). *AWGN* at 10 *SNR* with a correlation of 0.9966 after de-noise using Daubechies 9 and (down) *AWGN* at 10 *SNR* with a correlation of 0.9967 after de-noise using Daubechies 10

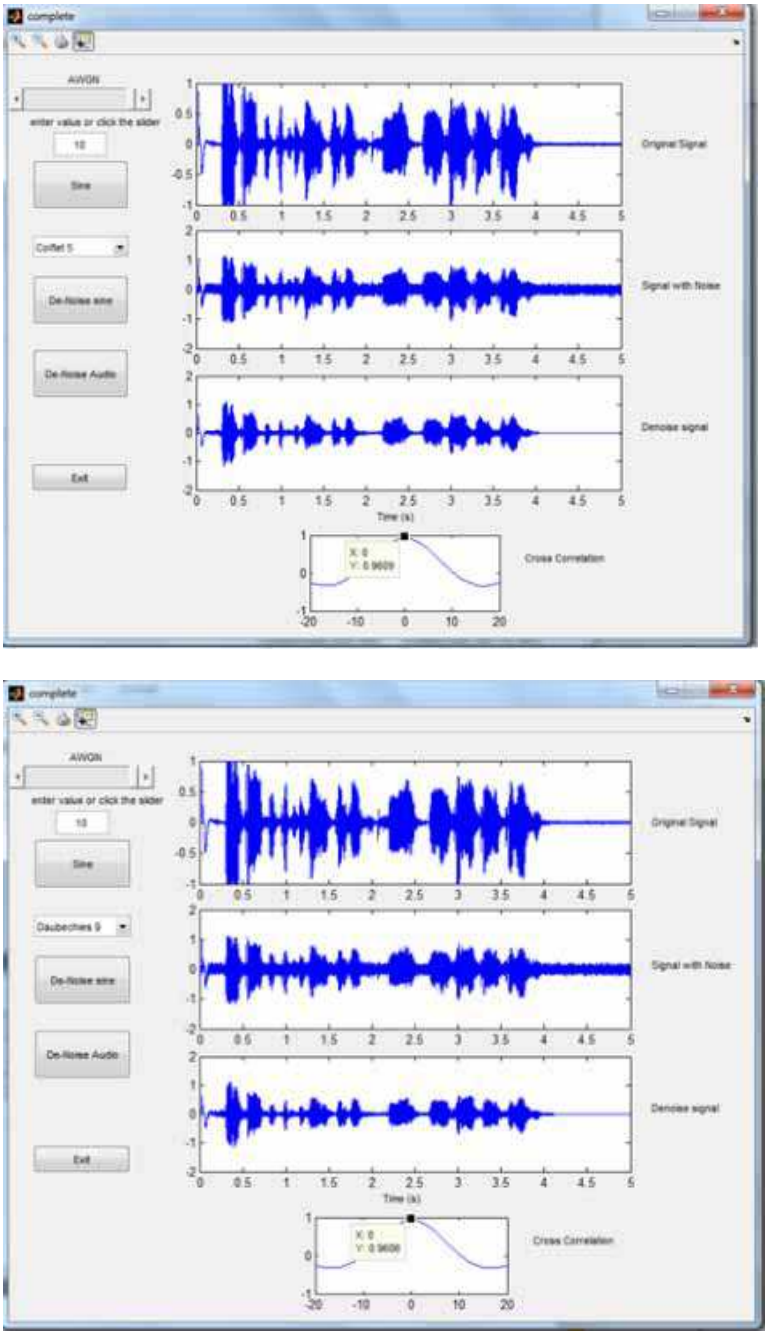


Fig. 20. (up). *AWGN* at 10 *SNR* with a correlation of 0.9609 after de-noise using Coiflet 5 and (down) *AWGN* at 10 *SNR* with a correlation of 0.9606 after de-noise using Daubechies 9

Figure 19 shows de-noise of sine with noise using $AWGN$ of 10, using Daubechies 9 and 10, this wavelets show a better performance that any others. Figure 20 shows behavior of Coiflet 5 and Daubechies 9 using $AWGN$ of 10 adding to audio signal.
In this section, it was described how to build a graphical interface using MATLAB code that was developed to de-noised sine and audio signals. This interface helps to be able to visualize the results obtained throughout this chapter, in addition to make it accessible to the user.

8. Conclusions

We provide a practical approach in how to put into practice wavelets in noisy audio data to improve clarity and signal retrieval. Since there are no books that show the code for a graphical interface with audio processing using wavelets, this chapter presents MATLAB code to reduce the Gaussian white noise in periodic signals (sine function) and in audio signals (composed of several frequencies) using wavelet analysis. We compared different wavelet families: Symlets, Daubechies and Coiflets, and we used cross-correlation to determine the best fit between an original signal and the processed one. By using Coiflet 5, Daubechies 9 and 10 we obtained the best result because they have a higher correlation at zero. Our signal processing technique recovers signal with a correlation higher than 99%. In analysis for audio signal with added Gaussian white noise, while using the technique we obtained a recovered signal with a correlation of 95%. This analysis is very useful to help the reader understand the know how in removing noise from a signal by using wavelets. Therefore, when a signal shows a periodic signal extraction from noise, it will be satisfactory. The graphical interface presented in the last section was performed while using GUIDE this one gives the readers a guideline to develop their own projects in MATLAB.

9. Acknowledges

The authors would like to thank Dr. Karen Esmonde-White for her helpful review and comments of this chapter. AEVL would like to thanks to Fundacion Pablo Garcia for help and support.

Apendix A: Five steps to a continuous Wavelet Transform

Here are the five steps of an easy recipe for creating a Continuous Wavelet Transform (CWT), see Figure A:

1. Take a wavelet and compare it to a section at the start of the original signal.
2. Calculate a number C, that represents how closely correlated the wavelet is with this section of the signal. The larger the number C is in absolute value, the more the similarity appears. If the signal energy and the wavelet energy are equal to one, C may be interpreted as a correlation coefficient. Note that, in general, the signal energy does not equal one and the CWT coefficients are not directly interpretable as correlation coefficients. Therefore, the CWT coefficients are different when you compute the CWT for the same signal by using different wavelets.
3. Shift the wavelet to the right and repeat steps 1 and 2 until you have covered the whole signal.
4. Scale (stretch) the wavelet and repeat steps 1 through 3.
5. Repeat steps 1 through 4 for all scales.

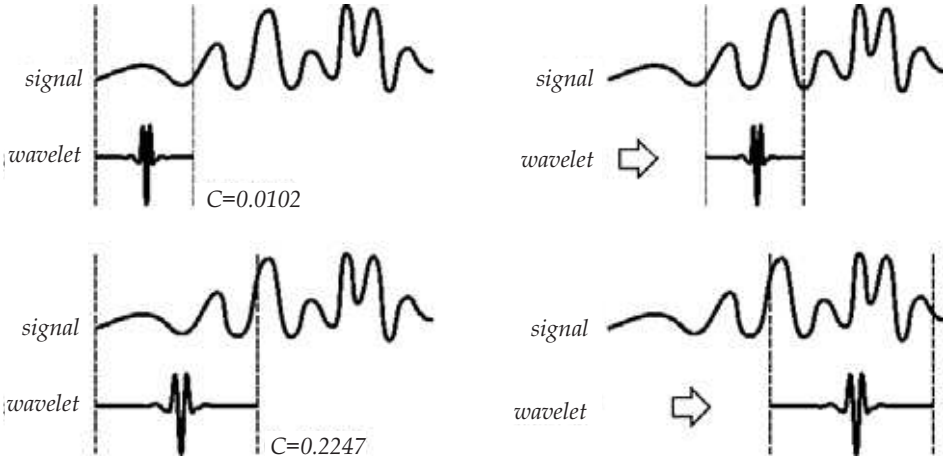


Fig. A. Shows recipe for creating a CWT

Appendix B: User Interface MATLAB code

In this appendix, it shows the complete code to develop the GUI using the GUIDE of MATLAB, and develops it as an example using a sine wave signal, which is added a certain level of noise and signal is shown graphically also recovered as the ratio correlation between the original signal and recovered signal.

```
function varargout = complete(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @complete_OpeningFcn, ...
                   'gui_OutputFcn',  @complete_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before complete is made visible.
function complete_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
clc
% Initalize error count and use edittext1 object's userdata to store it.
```

```
data.number_errors = 0;
set(handles.edit1,'UserData',data)
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = complete_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 D=crosscorr(x,y);
 z=-20:1:20;
 plot(handles.axes1,k,x)
 plot(handles.axes2,k,y)
 plot(handles.axes4,z,D)
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
menu= get(handles.popupmenu1,'Value');
switch menu
   case 1                                          %case 1: Coiflet 5
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 wname = 'coif5'; lev = 10;
 tree = wpdec(y,lev,wname);
 det1 = wpcoef(tree,2);
 sigma = median(abs(det1))/0.6745;
 alpha = 2;
 thr =wpbmpen(tree,sigma,alpha);
 keepapp = 1;
 xd = wpdencmp(tree,'s','nobest',thr,keepapp);
 D=crosscorr(x,xd);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
     set(gca,'XLim',[ 0.2,  0.24], ...
        'YLim',[-1 1]);
```

```
 plot(handles.axes4,z,D)
   case 2                                          %Case2 Daubechies 10
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 wname = 'db10'; lev = 10;
 tree = wpdec(y,lev,wname);
 det1 = wpcoef(tree,2);
 sigma = median(abs(det1))/0.6745;
 alpha = 2;
 thr =wpbmpen(tree,sigma,alpha);
 keepapp = 1;
 xd = wpdencmp(tree,'s','nobest',thr,keepapp);
 D=crosscorr(x,xd);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
      set(gca,'XLim',[ 0.2,  0.24], ...
       'YLim',[-1.1 1.1]);
  plot(handles.axes4,z,D)
   case 3                                          %Case 3: Daubechies 9
 k =(0:9.0703e-005:5);
 w=500*pi;
 h=w.*k;
 x = sin(h);
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 wname = 'db9'; lev = 10;
 tree = wpdec(y,lev,wname);
 det1 = wpcoef(tree,2);
 sigma = median(abs(det1))/0.6745;
 alpha = 2;
 thr =wpbmpen(tree,sigma,alpha);
 keepapp = 1;
 xd = wpdencmp(tree,'s','nobest',thr,keepapp);
 D=crosscorr(x,xd);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
  plot(handles.axes4,z,D)
end
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
menu= get(handles.popupmenu1,'Value');
switch menu
  case 1                                          %case 1: coiflet 5
 k = 0:9.0703e-005:5;
 w=500*pi;
 h=w.*k;
 [x,Fs,nbits]= wavread ('voice');
 f=gethandles.slider1,'Value');
 y = awgn(x,f,'measured');
 wavwrite(y,Fs,'noisyvoice')
 wname = 'coif5'; lev = 10;
 tree = wpdec(y,lev,wname);
 det1 = wpcoef(tree,2);
 sigma = median(abs(det1))/0.6745;
 alpha = 2;
 thr =wpbmpen(tree,sigma,alpha);
 keepapp = 1;
 xd = wpdencmp(tree,'s','nobest',thr,keepapp);
 D=crosscorr(x,xd);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
  plot(handles.axes4,z,D)
  case 2                                          %case 2 Daubechies 10
 k = 0:9.0703e-005:5;
 w=500*pi;
 h=w.*k;
 [x,Fs,nbits]= wavread ('voice');
 f=get(handles.slider1,'Value');
 y = awgn(x,f,'measured');
 wavwrite(y,Fs,'noisyvoice')
 wname = 'db10'; lev = 10;
 tree = wpdec(y,lev,wname);
 det1 = wpcoef(tree,2);
 sigma = median(abs(det1))/0.6745;
 alpha = 2;
 thr =wpbmpen(tree,sigma,alpha);
 keepapp = 1;
 xd = wpdencmp(tree,'s','nobest',thr,keepapp);
 D=crosscorr(x,xd);
 z=-20:1:20;
  plot(handles.axes1,k,x)
  plot(handles.axes2,k,y)
  plot(handles.axes3,k,xd)
```

```
 plot(handles.axes4,z,D)
  case 3                                          %Case 3: Daubechies 9
k = 0:9.0703e-005:5;
w=500*pi;
h=w.*k;
[x,Fs,nbits]= wavread ('voice');
f=get(handles.slider1,'Value');
y = awgn(x,f,'measured');
wavwrite(y,Fs,'noisyvoice')
wname = 'db9'; lev = 10;
tree = wpdec(y,lev,wname);
det1 = wpcoef(tree,2);
sigma = median(abs(det1))/0.6745;
alpha = 2;
thr =wpbmpen(tree,sigma,alpha);
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);
D=crosscorr(x,xd);
z=-20:1:20;
 plot(handles.axes1,k,x)
 plot(handles.axes2,k,y)
 plot(handles.axes3,k,xd)
 plot(handles.axes4,z,D)
end
% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
set(handles.edit1,'String',...
  num2str(get(hObject,'Value')));
% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function edit1_Callback(hObject, eventdata, handles)
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==10 && ...
  val >= get(handles.slider1,'min') && ...
  val <= get(handles.slider1,'max')
  set(handles.slider1,'Value',val);
else
% Retrieve and increment the error count.
% Error count is in the edit text UserData,
% so we already have its handle.
  data = get(hObject,'UserData');
  data.number_errors = data.number_errors+10;
```

```
% Save the changes.
  set(hObject,'UserData',data);
% Display new total.
  set(hObject,'String',...
  ['You have entered an invalid entry ',...
   num2str(data.number_errors),' times.']);
  % Restore focus to the edit text box after error
  uicontrol(hObject)
end
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
close all
```

10. References

Abbate A, Decusatis C. M, Das P. K. (2002). Wavelets and subbands: fundamentals and applications, ISBN 0-8176-4136-X, Birkhauser, Boston, USA.

Bahoura M & Rouat J, (2006). Wavelet speech enhancement based on time–scale adaptation, Speech Communication, Vol. 48, No. 12, pp: 1620-1637. ISSN: 0167-6393.

Davis, G, M, (2002). 'Noise Reduction in Speech Applications, CRC Press LLC, ISBN 0-8493-0949-2, USA.

Dong E & Pu X. (2008). Speech denoising based on perceptual weighting filter, Proceedings of 9th IEE International Conference on Signal Processing, pp: 705-708, October 26-29, Beijing. Print ISBN: 978-1-4244-2178-7.

Gold, B. & Morgan, N. (1999) Speech and audio signal processing: processing and perception of apeech, and music, John Wiley & Sons, INC., ISBN: 0-471-35154-7, New York, USA.

Johnson M. T, Yuan X and Ren Y, (2007). Speech Signal Enhancement through Adaptive Wavelet Thresholding, Speech Communications, Vol. 49, No. 2, pp: 123-133, ISSN: 0167-6393.

Képesia M & Weruaga L. (2006). Adaptive chirp-based time–frequency analysis of speech signals, Speech Communication, Vol. 48, No. 5, pp: 474-492. ISSN: 0167-6393.

Li N & Zhou M. (2008). Audio Denoising Algorithm Based on Adaptive Wavelet Soft-Threshold of Gain Factor and Teager Energy Operator, Proceedings of IEEE International Conference on Computer Science and Software Engineering, Vol. 1, pp: 787-790. Print ISBN: 978-0-7695-3336-0.

Mcloughlin I (2009). Applied Speech and Audio Processing With MATLAB Examples, Cambridge University Press, ISBN-13 978-0-521-51954-0, UK.

Minkoff, J. (2002). Signal Processing Fundamentals and Applications for Communications and Sensing Systems, ARTECH HOUSE, INC., ISBN 1-58053-360-4, USA.

Shankar B. J & Duraiswamy K. (2010). Wavelet-Based Block Matching Process: An Efficient Audio Denoising Technique,European Journal of Scientific Research, Vol.48 No.1, pp.16-28. ISSN 1450-216X.

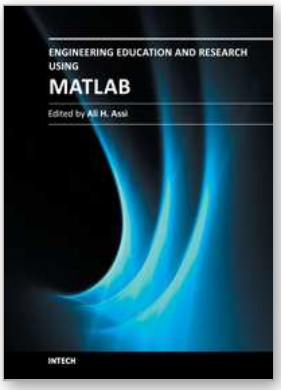
Tuzlukov, V. P. (2002). Signal processing noise, CRC Press LLC, ISBN 0-8493-1025-3,USA.

Vaseghi, S. V. (2008), Advanced Digital Signal Processing and Noise Reduction, fourth edition, John Wiley & Sons Ltd., ISBN 978-0-470-75406-1 (H/B), United Kingdom.

Visser E, Otsuka M & Lee T W. (2003). A spatio-temporal speech enhancement scheme for robust speech recognition in noisy environments, Speech Communication, Vol. 41, No. 2-3, pp: 393-407. ISSN: 0167-6393.

Wang C T& Wang H. G, (2003). Enhancement of single channel speech based on masking property and wavelet transform, Speech Communication, Vol. 41, No 2-3, pp: 409-427. ISSN: 0167-6393.

Wang C T& Wang H. G (2007). Speech enhancement using hybrid gain factor in critical-band-wavelet-packet transform, Digital Signal Processing, Vol. 17, No. 1, pp: 172-188. ISSN: 1051-2004.



Engineering Education and Research Using MATLAB

Edited by Dr. Ali Assi

ISBN 978-953-307-656-0
Hard cover, 480 pages
Publisher InTech
Published online 10, October, 2011
Published in print edition October, 2011

MATLAB is a software package used primarily in the field of engineering for signal processing, numerical data analysis, modeling, programming, simulation, and computer graphic visualization. In the last few years, it has become widely accepted as an efficient tool, and, therefore, its use has significantly increased in scientific communities and academic institutions. This book consists of 20 chapters presenting research works using MATLAB tools. Chapters include techniques for programming and developing Graphical User Interfaces (GUIs), dynamic systems, electric machines, signal and image processing, power electronics, mixed signal circuits, genetic programming, digital watermarking, control systems, time-series regression modeling, and artificial neural networks.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Adrian E. Villanueva- Luna, Alberto Jaramillo-Nuñez, Daniel Sanchez-Lucero, Carlos M. Ortiz-Lima, J. Gabriel Aguilar-Soto, Aaron Flores-Gil and Manuel May-Alarcon (2011). De-Noising Audio Signals Using MATLAB Wavelets Toolbox, Engineering Education and Research Using MATLAB, Dr. Ali Assi (Ed.), ISBN: 978-953-307-656-0, InTech, Available from: http://www.intechopen.com/books/engineering-education-and-research-using-matlab/de-noising-audio-signals-using-matlab-wavelets-toolbox

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821