Mechatronic Systems Identification

Project Report: Implementation of STFT and CWT in Matlab.

Witold Surdej - 407100

02.06.2024

1. Introduction

This report details the implementation of the Short-Time Fourier Transform (STFT) and Continuous Wavelet Transform (CWT) using the convolution theorem. It delves into the theoretical foundation of STFT and CWT and how they are implemented within the MATLAB environment, providing a step-by-step explanation of the process. The main part of the report covers the mathematical formulations, practical applications, and provides code snippets to illustrate the implementation and purpose of these two procedures in signal identification.

2. Background Theory

This section presents the relevant background information necessary to understand both procedures. It offers insight into the theory behind convolution, STFT and CTW.

2.1 Convolution Theorem

Convolution is a mathematical procedure used to combine two signals to form a third one. By many it is considered the most important operation in Digital Signal Processing. It allows us to describe a system by its impulse response. Convolution allows to relate input, output and response signals with each other. Therefore understanding the process of convolution is the most fundamental aspect of Digital Signal Processing.

Convolution is also known as impulse decomposition. It can be applied to both discrete and continuous signals, but the mathematics associated with computing continuous convolution are significantly more difficult, therefore much more prevalent is discrete convolution.

The aforementioned relationship which characterizes convolution can be represented mathematically in the following way for a linear system:

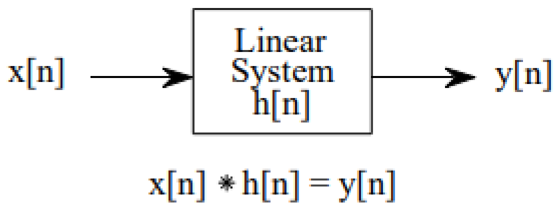


fig. 1 Convolution in DSP [1]

The equation presented on fig.1 showcases that in DSP the output of a linear system is a result of convolution between the input signal and the system's input response.

There is also an alternative way of understanding convolution. When dealing with Fourier transform related applications (such as STFT and CTW) we can consider the following definition stating that **convolution in time domain equals to pointwise multiplication in the frequency domain** and can be expressed by the following equation:

$r(x) = \{u * v\}(x) = F^{-1}\{U \cdot V\}$ [2]

where $\cdot$ denotes pointwise multiplication, F^{-1} is the inverse Fourier transform and * denotes convolution.

Finally the formal mathematical definition of the convolution is:

$(f * g)(t) = \int_{-\inf}^{+\inf} f(\tau)g(t - \tau)d\tau$ [3]

This equation can be interpreted as follows: To convolve a kernel $\{f(\tau)\}$ with an input signal $\{g\}$ flip the signal $\{g(-\tau)\}$, move it to the desired time $\{g(t - \tau)\}$ and accumulate every interaction with the kernel $\{d\tau\}$.

As mentioned before, convolution is an essential technique in Signal Processing found in different applications across the field. In our case we will focus on its uses in two particular techniques: Short-Time Fourier Transform (STFT) and Continuous Wavelet Transform (CWT).

2.2 Short-Time Fourier Transform (STFT)

Another fundamental method upon which modern day signal processing is built is Discrete Fourier Transform (FT). The DFT is an integral transform that takes an input function and transforms it into another function which describes the extent to which various frequencies are present in the original function. As such DFT is often called frequency domain representation of the original function. Formally DFT is described by the following mathematical formula:

$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$ [4]

Although the insight into frequency components of a signal is very valuable, it comes with a drawback, namely: TF is **insensitive** to the time at which the frequencies occur and disappear from the signal, meaning that a signal which is composed of a sum of simple cosine waves occurring at the same time will have the same spectrum as a signal composed of two cosine waves occurring at different times (see fig.2).

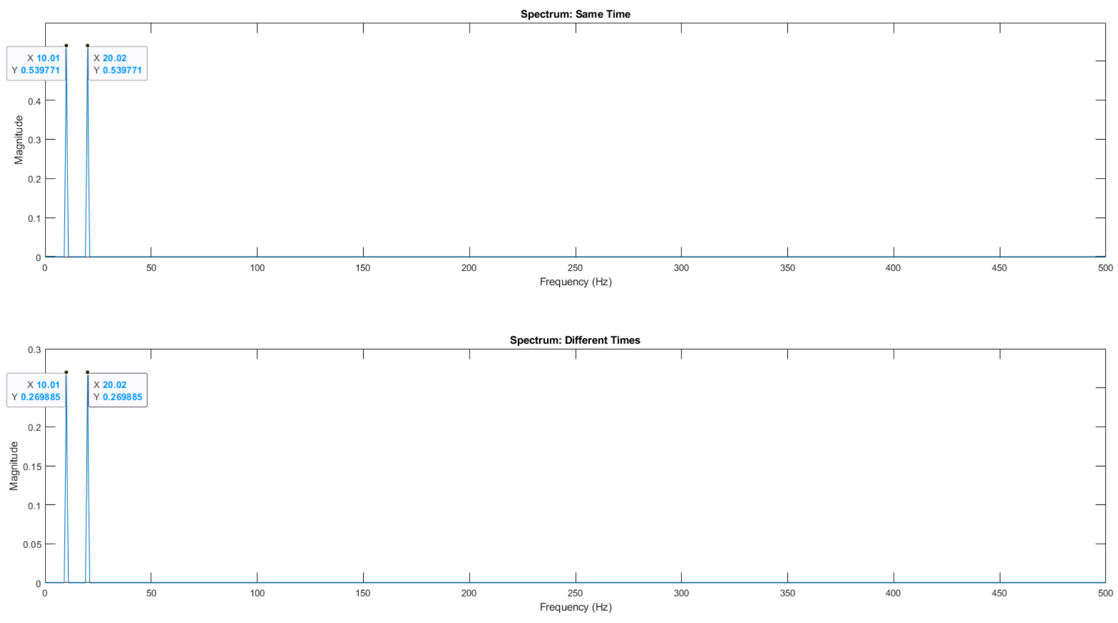


fig.2 proof of time insensitivity of a DFT

The STFT addresses the shortcomings of the DFT by introducing a windowing mechanism that enables localized analysis of signals over both time and frequency domains. Unlike the Fourier Transform, which considers the entire signal at once, the STFT breaks down the signal into smaller segments using a window function. This allows for a more detailed analysis of signals that vary over time.

The STFT works by segmenting the signal into overlapping windows of fixed length. Next a window function, such as Hamming or Hanning, is applied to each segment to reduce spectral leakage. Finally a spectra is computed for each segment by applying the DFT algorithm. This yields a series of frequency spectra corresponding to each time window. Next the spectra are arranged over time to construct a 2D time-frequency representation, where time is depicted along the horizontal axis and frequency along the vertical axis (the procedure is presented graphically on the fig.4).

From the mathematical perspective the STFT can be described by the following equation:

$$STFT\{x[n]\}(m, \omega) \equiv X(m, \omega) = \sum_{n--\text{inf}}^{+\text{inf}} x[n]w[n - m]e^{-i\omega n} \quad [5]$$

where $w[n]$ is the window function.

Commonly the results of the STFT are displayed as a spectrogram (fig.3), where the z-axis is the squared magnitude of the STFT.

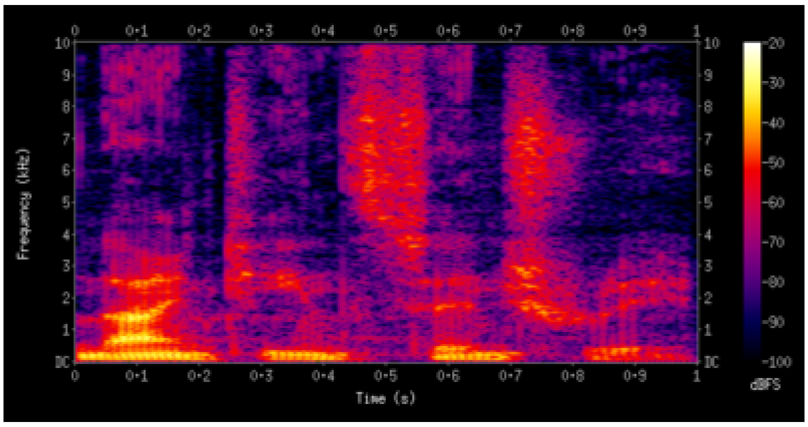


fig.3 Spectrogram [5]

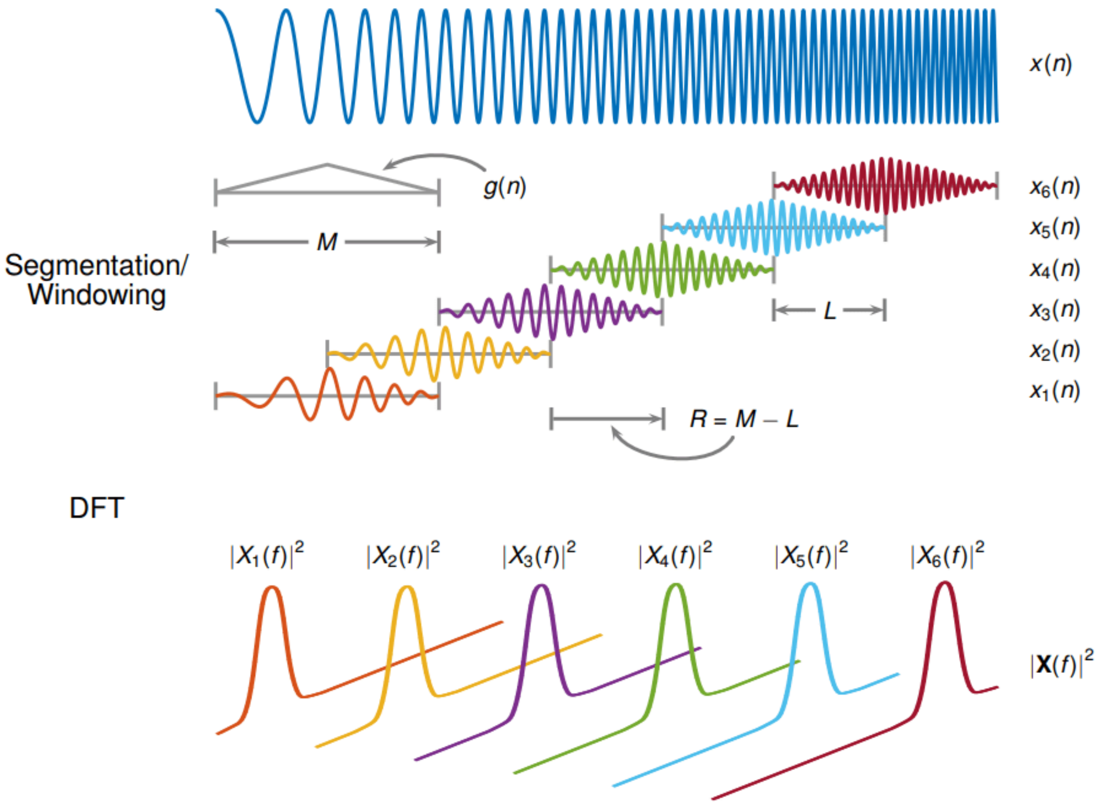


fig.4 Graphical description of an implementation of STFT [6]

Although STFT offers valuable insights into the time-varying frequency content of signals, it comes with inherent limitations due to the trade-off between time and frequency resolution. The windowing technique applied introduces a fundamental compromise: shorter windows improve temporal resolution but sacrifice frequency resolution, while longer windows enhance frequency resolution at the expense of temporal precision. This poses challenges when analyzing signals with rapidly changing frequency components or transient events, where achieving both high temporal and frequency resolutions simultaneously becomes unreachable. Due to this the selection of the parameters of the procedure is vital to its accuracy and usefulness.

Despite this limitation STFT remains the go-to method of analyzing the time-varying frequency content of signals.

2.3 Continuous Wavelet Transform (CWT)

The CWT is a pivotal tool in signal processing, offering an alternative approach to analyzing signals within the time-frequency domain. Unlike the Fourier Transform, which decomposes signals into sinusoids of various frequencies, the CWT employs wavelets—functions that are localized in both time and frequency domains. This characteristic allows for a more advanced analysis of transient features and non-stationary signals, making the CWT especially useful for signals that exhibit time-varying spectral content.

At the core of the CWT lies the wavelet function, denoted as Ψ(t). The transform involves comparing the signal to various shifted and scaled versions of this wavelet. The result is a function of two variables: scale (a) and position (b) (fig.5). The scale parameter controls the wavelet's width, analogous to frequency in Fourier analysis, while the position parameter determines the wavelet's location along the time axis.

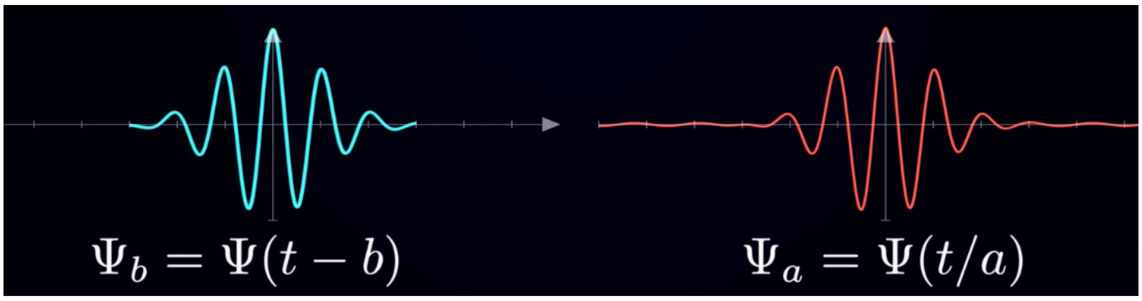


fig.5 influence of the parameters {a} and {b} on the mother wavelet [11]

Mathematically, the CWT of a signal f(t) with respect to the wavelet Ψ(t) is defined as:

$$W(a,b) = \frac{1}{\sqrt{a}} \int_{-\inf}^{+\inf} x(t)\ \bar{\Psi}\left(\frac{t-b}{a}\right)dt \quad [12]$$

where Ψ(t) is the wavelet function.

The primary wavelet Ψ(t) used in the CWT is known as the mother wavelet. This wavelet acts as the prototype from which other wavelets are generated through scaling and shifting. The choice of mother wavelet is crucial as it influences the CWT's sensitivity to different signal features.

In particular I have focused on two mother wavelets: Morlet and Morse, they are a popular choice when it comes to general purpose use of CTW due to their versatile properties. Morlet wavelet is popular due to its time-frequency localization properties. It is a plane wave modulated by a Gaussian window, which makes it particularly adept at identifying oscillatory behavior in signals. On the other hand the Morse wavelet is another versatile wavelet, known for its flexibility and ability to adapt to different signal characteristics. It is defined by a set of parameters that control its shape, allowing it to be tuned to specific features of the signal being analyzed. (fig.6)

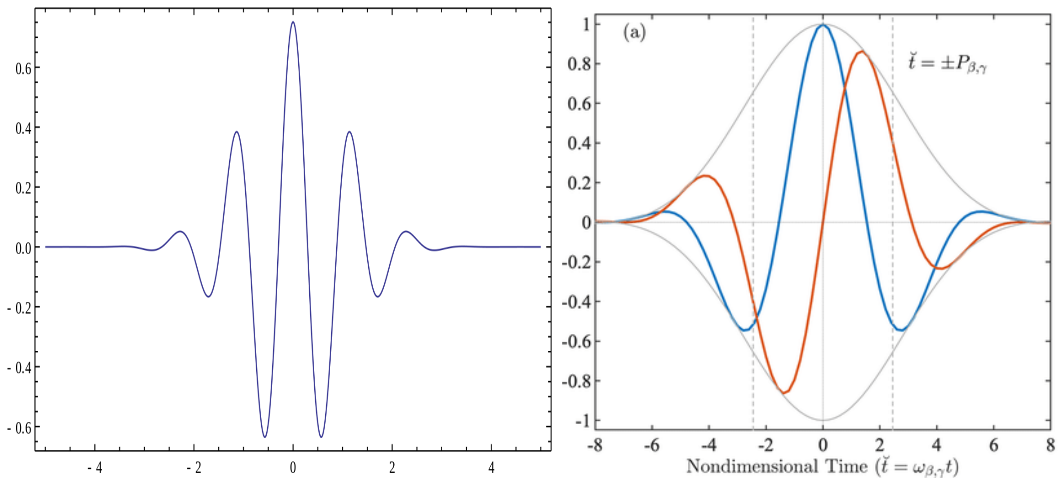


fig.6 Exemplary Morlet (left) and Morse (right) wavelets [13]

Another key element of the CWT is the relationship between scale and frequency. Smaller scales correspond to compressed wavelets that capture high-frequency, rapidly changing details, while larger scales correspond to stretched wavelets that capture low-frequency, slowly varying components. This relationship allows the CWT to provide a detailed time-frequency representation of a signal, adapting its resolution to the specific characteristics of the signal.

The concept of the Wavelet transform is a relatively new one (~40 years), but through the development and refinement of algorithms for CWT continue to expand its capabilities, making it an indispensable asset in both theoretical and applied signal processing since it provides a flexible and powerful means to analyze transient features in various applications.

3. Implementation Details

This section details the implementation of the STFT and CWT in MATLAB, as well as their integration into a graphical user interface (GUI). The primary objective was to ensure that my custom functions replicate MATLAB’s built-in functions. By maintaining identical input parameters, I have aimed to achieve identical results, facilitating direct comparison between my functions and MATLAB's native alternatives.

3.1 Parameters and Assumptions

Firstly let's discuss a couple general concepts and their impact on the STFT and CWT functions.

1) Window Length (Frequency Bandwidth):

Window Length (WL) refers to the size of the segment of the signal that is analyzed at one time. This is particularly relevant in the STFT since the WL determines the trade off between the time and frequency resolution (As mentioned in the section 2.2). When it comes to CWT the concept of WL is replaced by the "Scale" parameter of the wavelet.

2) Frequency Bins (Nfft):

Frequency Bins refer to the discrete divisions of the frequency spectrum in the output of a transform. In the context of STFT and CWT, Nfft is the number of points used in the computation of FFT. he number of frequency bins determines the frequency resolution of the transform. Higher Nfft results in finer frequency resolution, allowing for more detailed analysis of the frequency content of the signal at the cost of increased computational load. The choice of the Nfft value depends on the requirements of the analysis.

3.2 Implementation of STFT

This section highlights key elements of the code implementing STFT in MATLAB. Full code can be found in the attachments.

```
   % [S, F, T] = MYstft(x, fs, window, NoOverlap, NFFT)
   %
   % Inputs:
   %   x: Input signal (1D row vector)
   %   fs: Sampling frequency of the input signal (in Hz)
   %   window: Analysis window used for STFT (1D column vector, e.g.
Hamming window)
   %   NoOverlap: Percentage of overlap between consecutive windows (0
to 100)
   %   NFFT: Length of the FFT used for computing the STFT (integer,
preferably power of 2)
   %
   % Outputs:
   %   S: Short-Time Fourier Transform (STFT) matrix, where each column
represents
   %      the FFT of a windowed segment of the input signal (2D array of
size NFFT-by-N)
   %   F: Frequency vector corresponding to the STFT matrix S (1D array
of size NFFT)
   %   T: Time vector corresponding to the STFT matrix S (1D array of
size N)
```

The above snippet provides the information about the input and outputs of the function. As mentioned in the introduction I have utilized the same inputs as the MATLAB function.

```
% Compute hop size based on the overlap percentage
hop = round(wl * (1 - NoOverlap/100));
% Total number of time frames
N = floor((L - wl) / hop) + 1;
```

Firstly I have preallocated the space for the main computational loop and I have calculated hop size and number of frames based on the input data.

```
% Main computing loop for STFT
   for i = 0:N-1
       % Starting index of current frame
       start_idx = 1 + i * hop;
       % Extract current frame from the input signal
       x_frame = x(start_idx : start_idx + wl - 1).'; % Transpose to
column vector
       % Apply window to the frame
       x_windowed = x_frame .* window;
       % Compute FFT of the windowed frame
       fft_result = fft(x_windowed, NFFT);
       % Assign FFT result to the STFT matrix (X)
       S(:, i+1) = fft_result;
   end
```

The main computational loop is presented above, the exact steps are mentioned below on the fig.7.

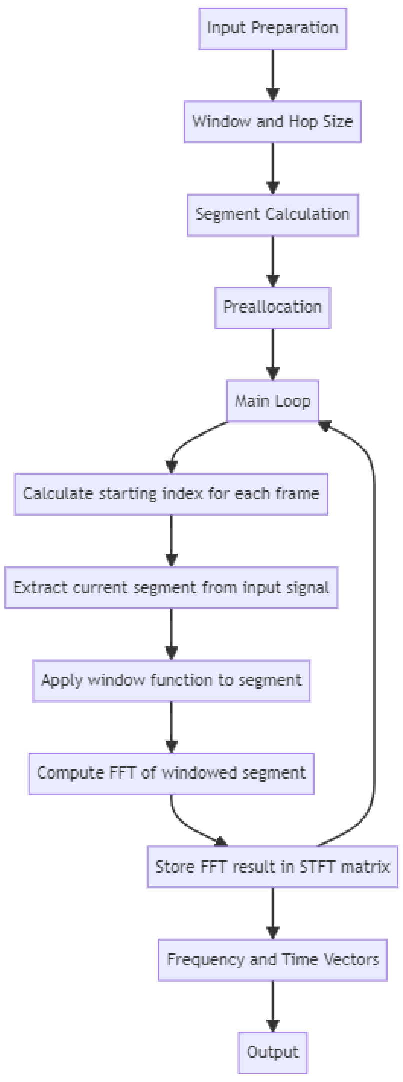

Input Preparation
Window and Hop Size
Segment Calculation
Preallocation
Main Loop
Calculate starting index for each frame
Extract current segment from input signal
Apply window function to segment
Compute FFT of windowed segment
Store FFT result in STFT matrix
Frequency and Time Vectors
Output

fig.7 Algorithm of the STFT

The computed STFT is plotted as a spectrogram using the Mesh function:

```
mesh(T2,F2,abs(S2).^2)
title("spectrogram based on my application of STFT")
view(2), axis tight
```

3.3 Implementation of CWT

In the preface of this section I would like to credit parts of the code for computing the CTW to prof. Colin M McCrimmon [10]. In particular I have utilized his implementation of the mother wavelets (both Morse and Morlet) as well as the plotting procedure for properly computing the scalogram. Remaining parts of the code are self derived from various sources on the internet with special mentions to the book: Multidimensional Signal, Image, and Video Processing and Coding (Second Edition) 2012.

The CWT algorithm is implemented with the following function:

```
function [timeFreqSpectrum, freqBins] = MYcwtOrg(signal, samplingFreq,
waveletType)
% Input:
% signal: input signal (real vector)
% samplingFreq: sampling frequency in Hz
% Output:
% timeFreqSpectrum: time-frequency spectrum (complex matrix)
% freqBins: frequency bins (in Hz) of the spectrum
```

Once again we start off by initiating the parameters and preallocating vectors. In addition due to the complexity of the code I have also added a section ensuring that the input signal is properly formatted. The code offers a simplified implementation of several features present in the actual Matlab function. Two wavelets were implemented (Morse and Morlet). For the Morse wavelet I have set the gamma and beta parameters as constant. Additionally by utilizing the functions from the internet [10] I have decided to implement automatic calculation of Scales and octaves (voices are constant), as well as automatically padding the signal.

The wavelet transform is computed based on the chosen wavelet type, which can be either Morse or Morlet. Each wavelet type has specific parameters:

- Morse Wavelet:

Parameters: Center frequency, gamma, and beta parameters.

Computation: The wavelet transform is computed using these parameters, and the frequency bins are determined accordingly.

- Morlet Wavelet:

Parameters: Central frequency (typically set to 6).

Computation: The wavelet transform is computed for the given scales and angular frequencies, and the frequency bins are derived based on the central frequency.

After defining the wavelet transform, the process involves performing the Fast Fourier Transform (FFT) on the input signal, applying the wavelet transform, and then computing the inverse FFT (IFFT) to obtain the time-frequency spectrum.

```
% FFT of the input

signalFFT = fft(signal);

timeFreqSpectrum = ifft((signalFFT * ones(1, size(waveletTransform, 2)))
.* waveletTransform);
```

The fft function computes the Fast Fourier Transform of the input signal, transforming it from the time domain to the frequency domain. The FFT of the signal is multiplied by the wavelet transform. This multiplication in the frequency domain corresponds to convolution in the time domain. replicates the FFT of the signal across the number of scales. The resulting matrix is then element-wise multiplied by the wavelet transform matrix. The ifft function computes the inverse FFT, transforming the product back to the time domain.

```
timeFreqSpectrum =
timeFreqSpectrum(1+paddingLength:originalLength+paddingLength, :).';
```

The result is trimmed to remove the padding added initially, retaining the original signal length.

Finally, the Cone of Influence (COI) is computed to identify regions in the time-frequency plane where edge effects become significant. The results are then plotted as a scalogram.

At the end, using the computed data a scalogram is plotted to graphically present the obtained results.

The Algorithm of the CTW looks as follows:

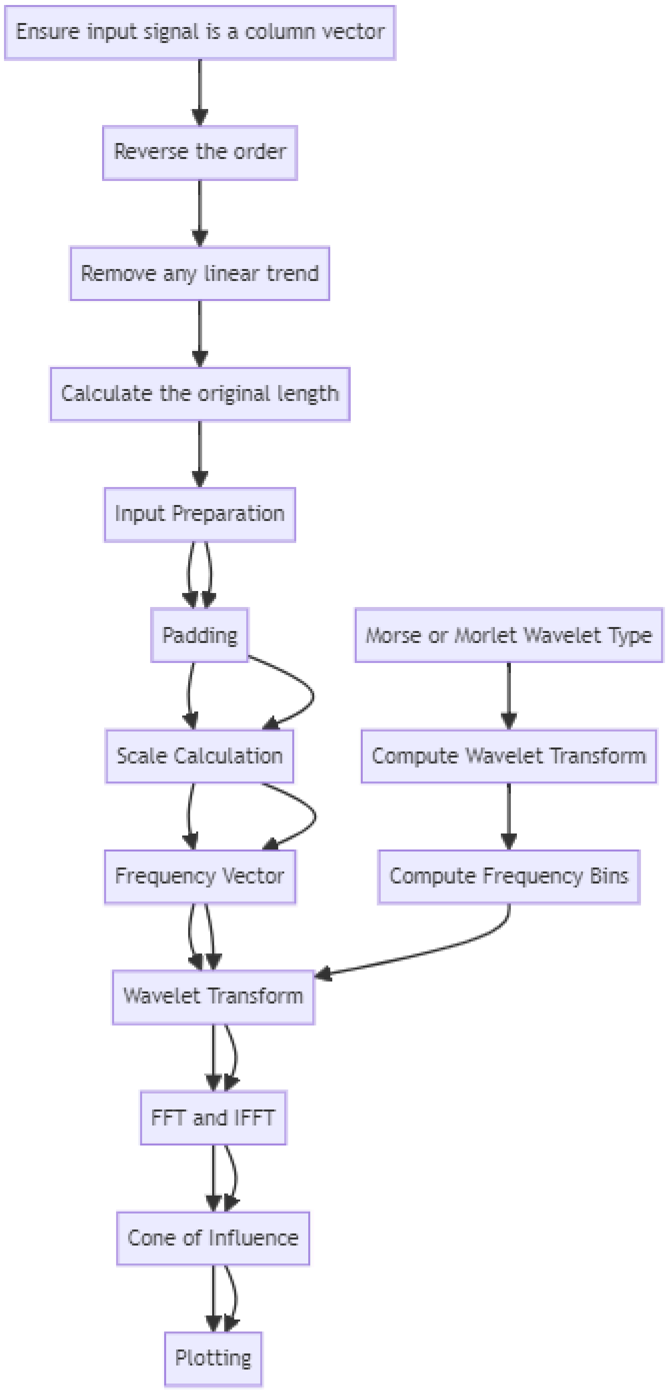


fig.8 Algorithm of the CWT

4. Results

The written functions were compared against their MATLAB counterparts to evaluate their performance.

Firstly the CTW was compared using the following parameters:

```
fs = 1400;
x = chirp(0:1/fs:2,600,2,100);
M = 128;
g = hann(M,"periodic");
L = 0.75*M;
Ndft = 128;
```

After running both my code and MATLAB's CTW function here are the results:

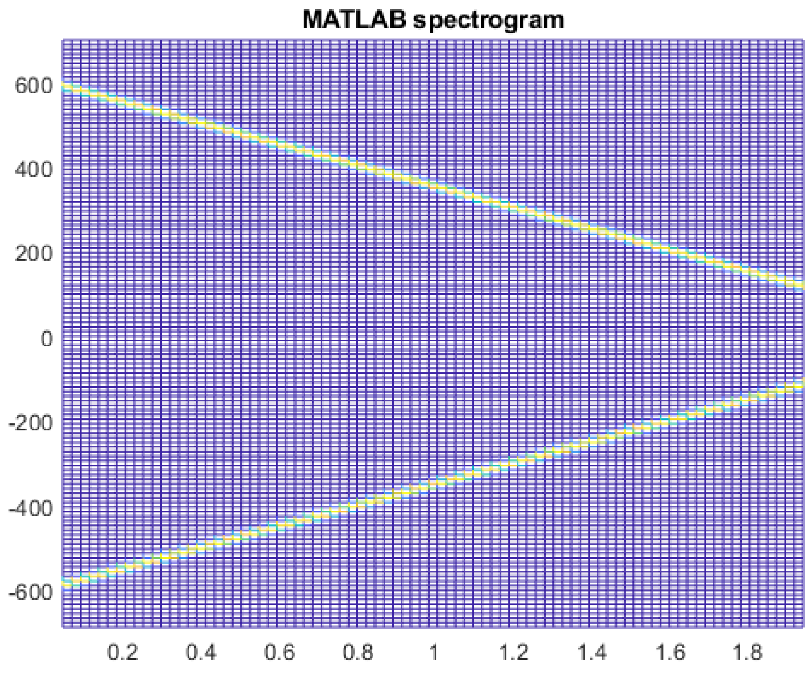


fig.9 Spectrogram of the chirp signal obtained using MATLAB Spectrogram

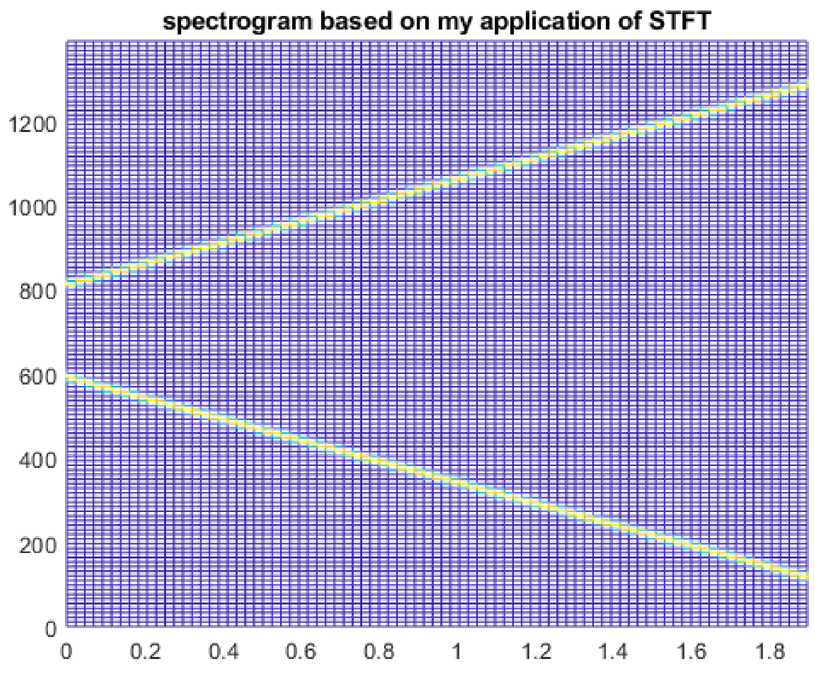


fig.10 Spectrogram of the chirp signal obtained using my STFT

As shown on the plots (fig.9 and 10) the results provided by my functions are correct. The frequencies present in both plots are plotted correctly, but the MATLAB function is more advanced and centers the frequencies at 0 making them easier to read. This feature is missing from my function but the overall performance was good.

When it comes to CTW the following results were obtained:

```
Fs = 1e3;
t = 0:1/Fs:1;
z = exp(1i*2*pi*32*t).*(t>=0.1 & t<0.3)+2*exp(-1i*2*pi*64*t).*(t>0.7);
wgnNoise = 0.05/sqrt(2)*randn(size(t))+1i*0.05/sqrt(2)*randn(size(t));
z = z+wgnNoise;
```

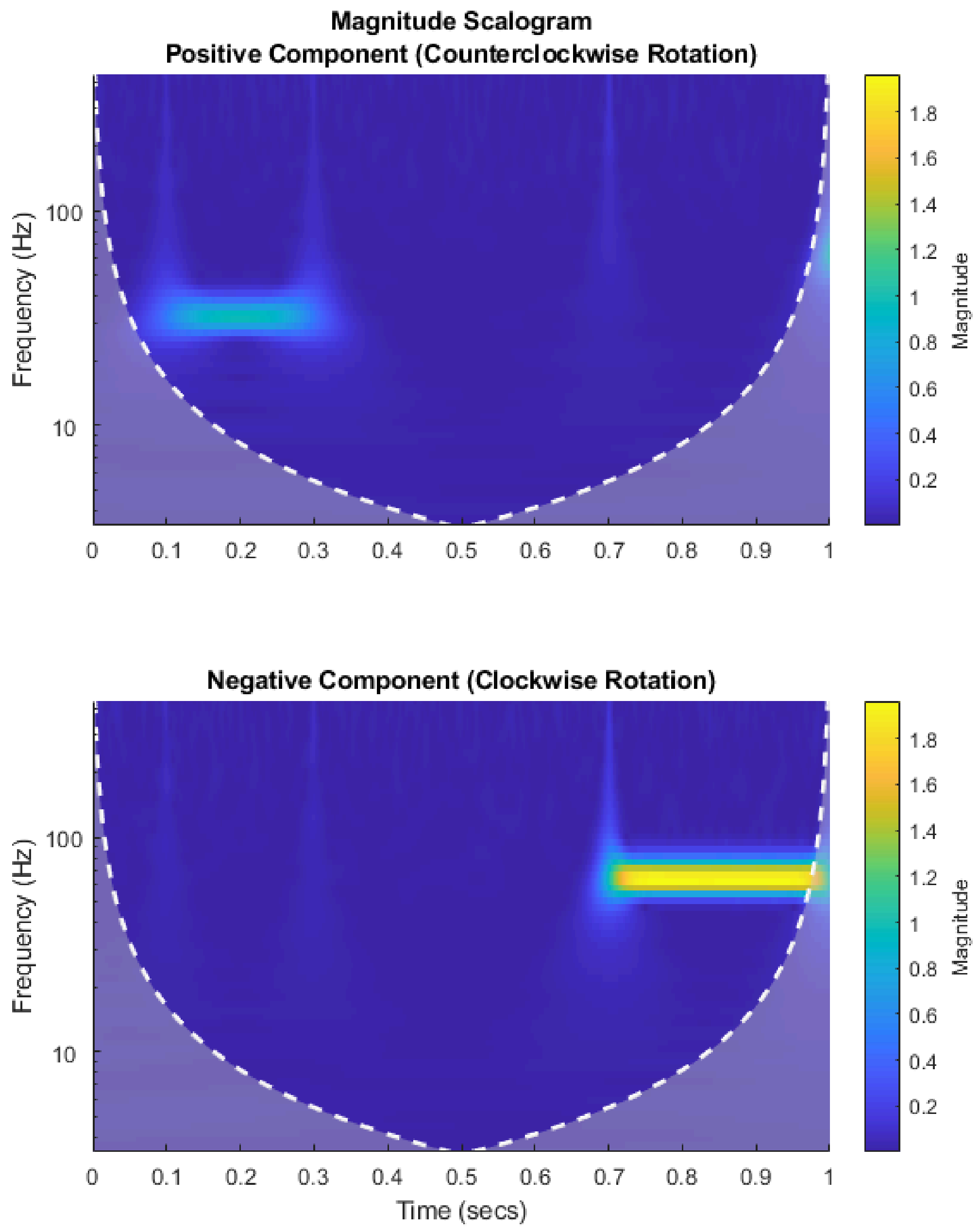


fig.11 Scalogram of the chirp signal obtained using MATLAB CWT

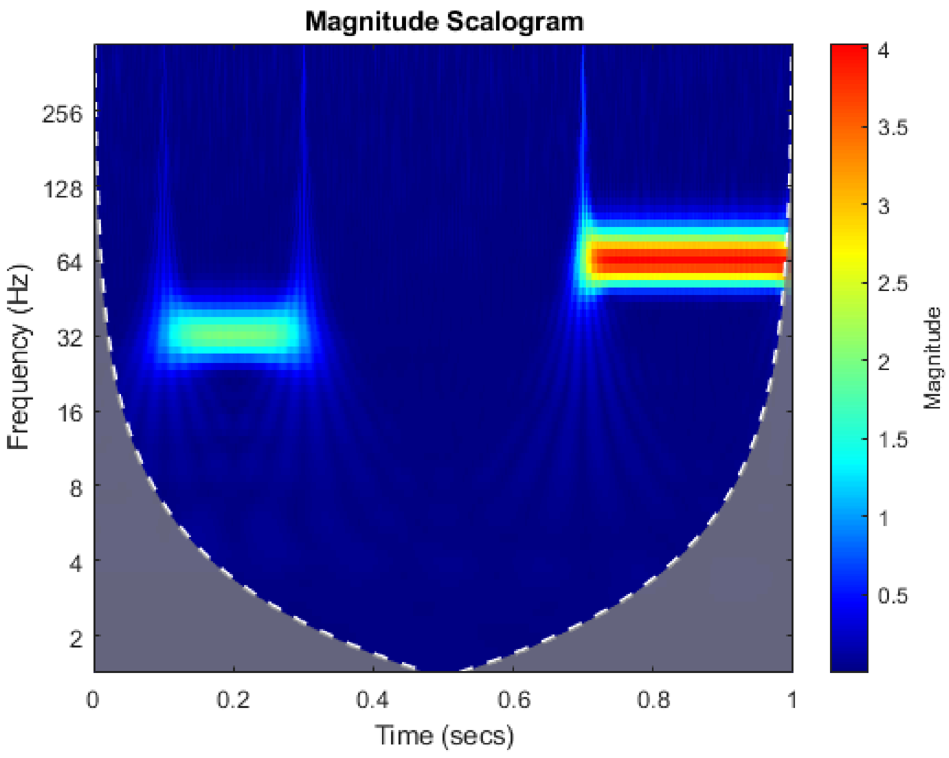
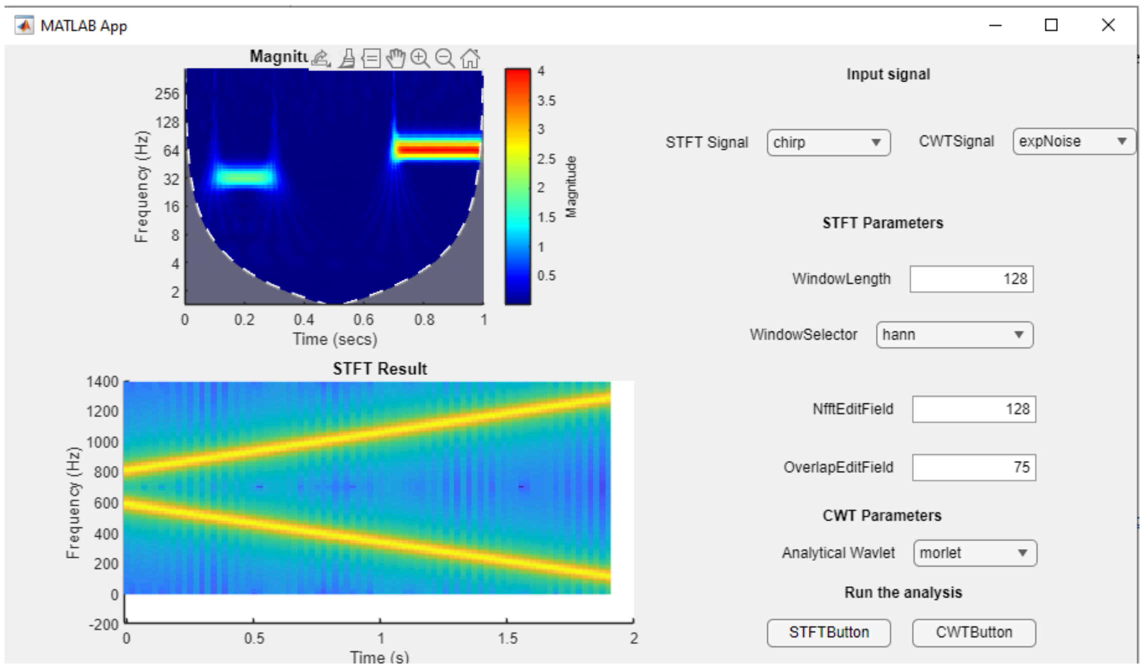


fig.12 Scalogram of the chirp signal obtained using my CWT

My implementation of the CWT offers the ability to implement a simplified version of the MATLAB CWT. It is able to accurately plot the scalogram using two wavelet types, but it does not allow for separate plotting of positive and negative components of the spectrum.

Finally both functions were implemented into a GUI which allows the user an easy way to perform signal analysis using my functions. The GUI looks as follows:


MATLAB App

Input signal

STFT Signal: chirp
CWTSignal: expNoise

STFT Parameters

WindowLength: 128
WindowSelector: hann
NfftEditField: 128
OverlapEditField: 75

CWT Parameters

Analytical Wavlet: morlet

Run the analysis

STFTButton
CWTButton

fig.13 GUI interface

The GUI allows for an easy modification of window length, Nfft, Overlap, selection of the analytical wavelet and the input signals. After selecting appropriate analysis parameters, the user can run the analysis to obtain the results.

5. Conclusion

The project on implementing the STFT and CWT in MATLAB gave me an insight regarding the theory behind those procedures, their importance in the field of signal processing and the difficulties associated with their implementation. By replicating MATLAB's built-in functions, we ensured our custom implementations could produce accurate and comparable results. The STFT allowed for localized time-frequency analysis using a windowing mechanism. On the other hand, the CWT, implemented with both Morse and Morlet wavelets, offered analysis of different signal characteristics and was particularly adept at capturing transient features. Using the convolution theorem, we have computed the wavelet transform in the frequency domain. The addition of a graphical user interface (GUI) provided a simple platform for signal analysis. Despite the success, there remains potential for further optimization and feature enhancement, such as incorporating more wavelet types and advanced visualization options. This project underscored the versatility and depth of time-frequency analysis.

6. References

[1] https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_ch6.pdf
[2] https://en.wikipedia.org/wiki/Convolution_theorem
[3] https://betterexplained.com/articles/intuitive-convolution/#Part_2_The_Calculus_Definition
[4] https://en.wikipedia.org/wiki/Fourier_transform
[5] https://en.wikipedia.org/wiki/Short-time_Fourier_transform
[6] https://uk.mathworks.com/help/signal/ref/stft.html
[7] https://qiml.radiology.wisc.edu/wp-content/uploads/sites/760/2020/10/notes_016_stft.pdf
[8]https://uk.mathworks.com/help/wavelet/gs/continuous-wavelet-transform-and-scale-based-analysis.html
[9] https://secwww.jhuapl.edu/techdigest/Content/techdigest/pdf/V15-N04/15-04-Sadowsky.pdf
[10] https://uk.mathworks.com/matlabcentral/fileexchange/65104-cmccrimm-continuous-wavelet-transform
[11] https://www.youtube.com/watch?v=jnxqHcObNK4&ab_channel=ArtemKirsanov
[12] https://en.wikipedia.org/wiki/Continuous_wavelet_transform
[13] https://en.wikipedia.org/wiki/Morlet_wavelet

Appendices

All the matlab codes are provided in the attached .zip file.