System Verification and Validation Plan for Time_Freq_Analysis

Elizabeth Hofer

October 29, 2020

1 Revision History

Date	Version	Notes
10.22.2020	1.0	Initial release

Contents

List of Tables

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
STFT	Short-Time Fourier Transform
FR	Functional Requirement
NFR	Non-Functional Requirement
SRS	Software Requirements Specification

This document outlines the verification and validation plan for Time_Freq_Analysis to help ensure (but not prove) correctness and completeness of the program. It includes some background information on Time_Freq_Analysis, a plan for testing the functional and non-functional requirements, an outline of the system tests, and an outline of the unit tests (not yet complete as it is depended on the MIS).

3 General Information

3.1 Summary

This document reviews the validation and versification plan for Time_Freq_Analysis, a program for the time-frequency analysis of 1D signals. Time_Freq_Analysis takes a signal $x(n)$ and computes a 2D matrix $X(\omega, n)$ which contains the frequency content at frequency ω at time n of $x(n)$.

3.2 Objectives

The objective of this document is to build confidence in the software's correctness. This document will be used to verify and validate the final product, and should therefore aim to encompass all validation and verification elements required for testing.

3.3 Relevant Documentation

This document is related to the system requirements specification found here: https://github.com/liziscool/cas741_project/blob/master/docs/SRS/SRS.pdf

4 Plan

4.1 Verification and Validation Team

The verification and validation team includes the domain expert Naveen Ganesh Muralidharan, the verification and validation reviewer Leila Mousapour, and Dr. Spencer Smith.

4.2 SRS Verification Plan

The SRS verification plan is to address the 5 functional and 4 non-functional requirements addressed in the SRS. Some of the requirements are combined for an area of testing so that one area of testing serves the purpose of testing multiple requirements. Some areas of testing may have more tests than others, what is important is that each area of testing covers the whole domain and includes the boundary cases where applicable.

4.3 Design Verification Plan

The main user of Time_Freq_Analysis is also the author of this VnV document. As such, the verification of the design can be confirmed as they work through the test cases created from them SRS verification plan.

4.4 Implementation Verification Plan

Individual tests are outlined in sections 5.1 through 5.6. The tests will either be automated, performed manually (usually just for quick tests that involve the tester confirming that something happened), or with someone reviewing the code.

4.5 Automated Testing and Verification Tools

Since Time_Freq_Analysisis fairly simple in terms of inputs and outputs, a simple script is all that is needed to automate testing for those tests that need be automated.

4.6 Software Validation Plan

To test the output of the software, a pseudo-oracle will be used. As is outlined below, pseudo-oracle is needed compare to the output of Time_Freq_Analysis for very simple input functions. Since these input signals are trivial, their time-frequency transform is also trivial, so the tester will construct those herself.

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Area of Testing1 - Fun. Req. 1 and Fun. Req. 2 - Inputs

Functional Requirement 1 states: Program shall take the signal to be analysed as input. All other inputs (as specified in table 1) will have defaults, but program shall accept user inputs for those as well.

Functional Requirement 2 states: Program shall notify user if an input value is illegal or out of bounds.

This area of tests will address the inputs to the program. The following tests include tests with inputs that are within bounds to test Fun. Req. 1 and tests with inputs that are out of bounds to test Fun. Req. 2.

Tests for Fun. Req. 1 and 2

1. test - Normal Input using defaults

   Control: Automatic

   Initial State: Pending input

   Input: signal $x(n)$, using default inputs boundaries

   Output: Program should run with no error, (correctness of solution is addressed in other tests)

   Test Case Derivation: Fun. Req. 1

   How test will be performed: Automatically with a script that runs for 10 different functions $x(n)$ as specified in table 1.

2. test - Normal Input with user entered bounds

   Control: Automatic

   Initial State: Pending input

   Input: signal $x(n)$ with n_i, δ_N, ω_{min}, ω_{max} define as in table

   Output: Program should run with no error, (correctness of solution is addressed in other tests)

Test Case Derivation: Fun. Req. 1

How test will be performed: Automatically with a script that runs for 10 different functions $x(n)$ as specified in table 1 with different input bounds as in other table 2

3. test - Out of Bounds for N

   Control: Manual

   Initial State: Pending input

   Input: signal $x(n)$, for $n[0:N]$ analyse for $n_i + \delta_n > N$

   Output: Program should return error about analysis range

   Test Case Derivation: Fun. Req. 2

   How test will be performed: Since this is just a quick check for an input, it does not require automation

4. test- Out of Bounds for ω_{max}

   Control: Manual

   Initial State: Pending input

   Input: $x(n)$, with sampling period P analyse for $\omega_{max} < 1/P$, or is close to $1/P$

   Output: Program should return error about analysis range

   Test Case Derivation: Fun. Req. 2

   How test will be performed: Since this is just a quick check for an input, it does not require automation

5. test- Empty $x(n)$

   Control: Manual

   Initial State: Pending input

   Input: signal $x(n)$ that is empty

   Output: Program should return warning about empty signal

   Test Case Derivation: Fun. Req. 2

How test will be performed: Since this is just a quick check for an input, it does not require automation

6. test- Insufficient Δ_n

   Control: Manual

   Initial State: Pending input

   Input: signal $x(n)$ and δ_n that is very small, e.x. $\delta_n < 10$

   Output: Program should return warning about small time period

   Test Case Derivation: Fun. Req. 2

   How test will be performed: Since this is just a quick check for an input, it does not require automation

7. test - Insufficient difference between ω_{max} and ω_{min}

   Control: Manual

   Initial State: Pending input

   Input: signal $x(n)$ and ω_{min} that is too close to ω_{max}, e.x. $\omega_{max} < \omega_{min} + 10$

   Output: Program should return warning about small time period

   Test Case Derivation: Fun. Req. 2

   How test will be performed: Since this is just a quick check for an input, it does not require automation

5.1.2 Area of Testing 2 - Fun. Req. 3 and Fun. Req. 5

Functional requirement 3 states: The output shall be a time frequency representation of the signal in the specified time period and over the specified frequency range.

Also functional requirement 5 states: The time-frequency representations of simple input signals (such as sinusoids of a constant frequency or an impulse) should be comparable to existing time-frequency transforms of that signal.

Table 1: Area of testing 1 - Inputs for testing, sampling period is 0.1 s

signal input	ω_{signal}
$x(n) = \sin(\frac{2\pi}{100}n)$	10 Hz
$x(n) = \sin(\frac{2\pi}{200}n)$	20 Hz
$x(n) = \sin(\frac{2\pi}{300}n)$	30 Hz
$x(n) = \sin(\frac{2\pi}{500}n)$	50 Hz
$x(n) = \sin(\frac{2\pi}{800}n)$	80 Hz
$x(n) = \sin(\frac{2\pi}{1000}n)$	100 Hz
$x(n) = \sin(\frac{2\pi}{5000}n)$	500 Hz
$x(n) = \sin(\frac{2\pi}{10000}n)$	1000 Hz
$x(n) = \sin(\frac{2\pi}{20000}n)$	20000 Hz
$x(n) = \sin(\frac{2\pi}{30000}n)$	3000 Hz

This area of tests will address the outputs of the program. The following tests address assuring the output is what is required and test if the output is correct.

Note that due to the nature of time frequency representations, there are many ways a representation can be considered correct. The following tests try to address this by using consistent parameters for testing and taking into considerations that there are multiple ways to represent a time-frequency transform correctly. For some of these tests, the program is given a basic signal, and the output is compared the output from a pseudo-oracle, in this case the pseudo oracle is the tester, since the inputs are trivial the time-frequency representations are also trivial, and so a oracle-program will be written to produce the output expected from these trivial inputs. Of course, the output of the software Time_Freq_Analysis will not align exactly with the pseudo-oracle, but for simple signals they should be comparable. For this reason it is essential that the input signals for this test case remain simple.

Additionally, there are tests included that do not depend on a pseudo-

Table 2: Area of testing 1 - Boundaries for testing

n_i	Δ_n	ω_{min}	ω_{max}
0	1000	10 Hz	1000 Hz
0	1000	50 Hz	1000 Hz
0	1000	10 Hz	2000 Hz
0	1000	50 Hz	2000 Hz
0	1000	10 Hz	3000 Hz
0	1000	50 Hz	3000 Hz
0	1000	1000 Hz	3000 Hz
500	1000	10 Hz	1000 Hz
0	500	10 Hz	1000 Hz
0	50	50 Hz	1000 Hz
0	50	10 Hz	3000 Hz

oracle that are included for extra assurance. These test cases are derived from the mathematical properties of the theoretical models that govern the program, as outlined in the SRS.

Tests for Fun. Req. 3 and 5

1. test1- STFT

   Control: Automatic

   Initial State: Pending input for STFT

   Input: signal $x(n)$ according to table 3

   Output: The time frequency representation $X(\omega, n)$

   Case Derivation: Fun. Req. 5

   How test will be performed:

(a) Step 1: Compute $X(\omega, n)$ with Time_Freq_Analysis and $X_o(\omega, n)$ with the oracle for each signal $x(n)$ with the same input boundaries

(b) Step 2: Compute the difference element-wise be X and X_o using:

$$\frac{X(\omega_i, n_j) - X_o(\omega_i, n_j)}{X_o(\omega_i, n_j)}$$

. Record for elements in matrix.

(c) Step 3: In addition, take an average of the error for all elements in the matrix, and record.

2. test1 - Wavelet

Control: Automatic

Initial State: Pending input for Wavelet

Input: signal $x(n)$ according to table 3

Output: The time frequency representation $X(\omega, n)$

Case Derivation: Fun. Req. 5

How test will be performed:

(a) Step 1: Compute $X(\omega, n)$ with Time_Freq_Analysis and $X_o(\omega, n)$ with the oracle for each signal $x(n)$ with the same input boundaries

(b) Step 2: Compute the difference element-wise be X and X_o using:

$$\frac{X(\omega_i, n_j) - X_o(\omega_i, n_j)}{X_o(\omega_i, n_j)}$$

. Record for elements in matrix.

(c) Step 3: In addition, take an average of the error for all elements in the matrix, and record.

3. test2 - STFT

Control: Automatic

Initial State: Pending input for STFT

Input: signal $x(n)$ according to table 3

Output: $X(\omega, n)$ such that for all n, $X(\omega_{signal}, n) > X(\omega, n)$ for all $\omega \neq \omega_{signal}$

Test Case Derivation: Functional requriment 5 and theoretical models 1 and 2.

How test will be performed: Automatically with a script

4. test2 - Wavelet

   Control: Automatic

   Initial State: Pending input for Wavelet Transform

   Input: signal $x(n)$ according to table 3

   Output: $X(\omega, n)$ such that for all n, $X(\omega_{signal}, n) > X(\omega, n)$ for all $\omega \neq \omega_{signal}$

   Test Case Derivation: Functional requirement 5 and theoretical Model 3

   How test will be performed: Automatically with a script

5.1.3 Area of Testing 3 - Fun. Req. 4

Functional Requirement 4 states: The program should minimize spectral leakage.

Due to the nature of time-frequency transforms there will always be some spectral leakage. The following sets of test will try to quantify the amount of spectral leakage in a similar method used in area of testing 3. These tests will input simple known signals $x(n)$ into Time_Freq_Analysis and compare the output to a known time-frequency representation of that signal, essentially a pseudo-oracle. It will then compare the amount of spectral leakage of the output of Time_Freq_Analysis to the pseudo-oracle representation, which should basically have no spectral leakage.

Tests for Fun. Req. 4

1. test-id1

   Type: Automatic

Table 3: Input Signals for area of testing 2, assume a sampling period of 0.1 seconds.

signal input	**time boundary**	**frequency boundary in Hz**	**average difference**
$x(n) = \sin(\frac{2\pi}{100}n)$	$n[0, 1000]$	$\omega[0, 1000]$	
$x(n) = \sin(\frac{2\pi}{200}n)$	$n[0, 1000]$	$\omega[0, 1000]$	
$\vdots$	$\vdots$	$\vdots$	
$x(n) = \sin(\frac{2\pi}{100}n) + \sin(\frac{2\pi}{500}n)$	$n[0, 1000]$	$\omega[0, 1000]$	
$\vdots$	$\vdots$	$\vdots$	
$x(n) = \begin{cases} \sin(\frac{2\pi}{100}n) & \text{if } 0 < n < 500 \\ 0 & \text{else} \end{cases}$	$n[0, 1000]$	$\omega[0, 1000]$	
$\vdots$	$\vdots$	$\vdots$	
$x(n) = \begin{cases} \sin(\frac{2\pi}{100}n) & \text{if } 0 < n \leq 200 \\ \sin(\frac{2\pi}{220}n) & \text{if } 200 < n \leq 400 \\ \sin(\frac{2\pi}{500}n) & \text{if } 400 < n \leq 600 \\ \sin(\frac{2\pi}{1000}n) & \text{if } 600 < n \leq 1000 \end{cases}$	$n[0, 1000]$	$\omega[0, 1000]$	

Initial State: Pending Input for STFT

Input/Condition: $x(n)$ according to table 1

Output/Result: time frequency representation $X(\omega, n)$

Test Case Derivation: Functional Req. 4

How test will be performed:

(a) Step 1 Compute $X(\omega, n)$ with Time_Freq_Analysisand $X_o(\omega, n)$ with the oracle for each signal $x(n)$ with the same input boundaries

(b) Step 2 Compute the difference element-wise be X and X_o using:

$$\frac{X(\omega_i, n_j) - X_o(\omega_i, n_j)}{X_o(\omega_i, n_j)}$$

record for elements in matrix.

(c) Step 3 In addition, take an average of the spectral leakage all elements in the matrix.

2. test-id2

Type: Automatic

Initial State: Pending Input for Wavelet Transform

Input/Condition: $x(n)$ according to table 3

Output/Result: time frequency representation $X(\omega, n)$

Test Case Derivation: Functional Req. 4

(a) Step 1 Compute $X(\omega, n)$ with Time_Freq_Analysisand $X_o(\omega, n)$ with the oracle for each signal $x(n)$ with the same input boundaries

(b) Step 2 Compute the difference element-wise be X and X_o using:

$$\frac{X(\omega_i, n_j) - X_o(\omega_i, n_j)}{X_o(\omega_i, n_j)}$$

record for elements in matrix.

(c) Step 3 In addition, take an average of the spectral leakage all elements in the matrix.

6 Tests for Nonfunctional Requirements

6.1 Area of testing 6 - Nonfun. Req. 6

Plotting Output as a Heat Map Non-Functional Requirement 6 states: Program shall plot time-frequency representation as a heat map.

1. test - Heat Map

   Type: Manual

   Initial State: Output

   Input/Condition: Any signal $x(n)$ from table 3

   Output/Result: Heat-map type plot of the matrix

   How test will be performed: Manually as it is a simple test that just needs to be ran once, the tester will have to look at the matrix and the heat map to make sure they are generally communicating the same representation.

6.2 Area of testing 8 - Nonfun. Req. 8

Useableity Non-Functional Requirement 8 states: Program will not have a graphical user interface but should still be easy to use, the input parameters besides the signal shall all have default values, there should be at most 6 optional inputs.

1. test - Usability

   Type: Manual

   Initial State: Off

   Input/Condition: User input for any signal $x(n)$ from table 3 and any boundaries in table 2

   Output/Result: Time_Freq_Analysis should compute the time frequency representation

   How test will be performed: Manually, as it is a test of usability by the *user*, and thus requires the test to be ran by the user.

6.3 Area of testing 9 - Nonfun. Req. 9

Readability Non-Functional Requirement 9 states: The program code should be clear and readable.

1. test - Readability

   Type: Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result:N/A

   How test will be performed: The author will review the code and determine if it satisfies the qualities in table 4

Table 4: Code Readability Qualities

Quality	Y/N
Readable and logical variable names	
Functions are self explanatory or have comments to describe what they do	
Every function is responsible for one single thing	
Code is organized structurally	

6.4 Area of testing 10 - Nonfun. Req.10

Easily integrate with other systems Non-Functional Requirement 10 states: The program should easily integrate with other software programs.

1. test - Heat Map

   Type: Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: N/A

How test will be performed: The tester just needs to ensure that the program can be called like a function from inside another program, therefore the inputs and outputs must have consistent types.

6.5 Traceability Between Test Cases and Requirements

Table 5 demonstrates the traceability between the tests cases and the requirements.

	Test 5.1.1	Test 5.1.2	Test 5.1.3	Test6.1	Test 6.2	Test 6.3	Test 6.4
R1	X						
R2	X						
R3		X					
R4			X				
R5		X					
R6				X			
R8					X		
R9						X	
R10							X

Table 5: Traceability Matrix Showing the Connections Between Requirements and Instance Models

7 Unit Test Description

This section is intentionally left blank until the MIS is completed.

7.1 Unit Testing Scope

7.2 Tests for Functional Requirements

7.2.1 Module 1

1. test-id1

Type:

Initial State:

Input:

Output:

Test Case Derivation:

How test will be performed:

2. test-id2

   Type:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

3. ...

7.2.2 Module 2

...

7.3 Tests for Nonfunctional Requirements

7.3.1 Module ?

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

7.3.2 Module ?

...

7.4 Traceability Between Test Cases and Modules

8 Appendix

8.1 Symbolic Parameters

N/A at this time.