A Fourier Series Library: Unit Verification and Validation Plan for FSL

Bo Cao

October 28, 2019

1 Revision History

Date	**Version**	**Notes**
Oct. 28, 2019	1.0	First draft.
Date 2	1.1	Notes

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **iii**

3 General Information **1**
- 3.1 Purpose 1
- 3.2 Scope 1

4 Plan **1**
- 4.1 Verification and Validation Team 1
- 4.2 Automated Testing and Verification Tools 1
- 4.3 Non-Testing Based Verification 2

5 Unit Test Description **2**
- 5.1 Tests for Functional Requirements 2
  - 5.1.1 Module 1: Basic comparison function 2
  - 5.1.2 Module 2: Fourier transformation and approximation . 4
  - 5.1.3 Module 3: Operations and functions 5
  - 5.1.4 Test of conversion 6
- 5.2 Tests for Nonfunctional Requirements 7
  - 5.2.1 Speed evaluation 7
- 5.3 Traceability Between Test Cases and Modules 7

2 Symbols, Abbreviations and Acronyms

Some symbols, abbreviations and acronyms are defined in the Common Analysis (CA) document [1]. For simplicity and maintainability, they are not redefined here. Readers shall refer to the CA documents when a certain item is not defined here.

symbol	**description**
T	Test

[1]This document is available at https://github.com/caobo1994/FourierSeries/blob/master/docs/SRS/CA.pdf.

This document provides an overview of the Verification and Validation (VnV) plan for the Fourier Series Library (FSL). It lays out the purpose, methods, and test cases for the VnV procedure.

3 General Information

3.1 Purpose

The library to be tested is called the Fourier Series Library (FSL). This library performs a set of computations, transformations, and/or input/output at the request of the library user.

3.2 Scope

The intended objective of the VnV procedure is to verify that this library has generally met the requirements described in the CA document. These requirements include the functional requirements (FRs) and the non-functional requirements (NFRs).

Note that if a small part of the NFRs has not been met, the library is still acceptable when the not-met NFRs' impact has been analyzed and deemed non-essential.

The transformation from a function to its CFS involves numerical integration, and we plan to use existing libraries for it. Instead of test these libraries, we will trust them on the ground of their developers' and other users' through testing.

4 Plan

4.1 Verification and Validation Team

The major member of the team is the author himself. Other contributors might assist in the VnV procedure, but their contributions are not guaranteed.

4.2 Automated Testing and Verification Tools

We have decided that this library will be implemented in `C++`.

The unit test library we choose is `Catch2`[2]. This library is designed for simple unit testing in both `C` and `C++`. We choose this library because it is light-weight, thus easy to install and use. These two properties are especially helpful in our situation, since I expect that most of the other collaborators have little knowledge in C++ and unit testing.

The static analysis will be done by `cpplint`. This is a classic `C++` program static analysis tool. It is maintain by Google for checking a `C++` program's consistency with Google's `C++` code style guidance. This tool and guidance can be found at `https://github.com/google/styleguide`.

The code coverage of the designed unit test cases will be tested by `gcov`. This is the standard code coverage tool that is assisted by the GNU program/library family's `C++` compiler, `g++`. An detailed introduction can be found in `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`.

The (pseudo-)oracles for unit testing come from manual computation and/or the results computed by MATLAB.

4.3 Non-Testing Based Verification

We will have a code inspection and walkthrough of some functions in this library, if they are deemed too complex. This test will be assisted by the result of `cpplint`.

5 Unit Test Description

The general principle for selecting test cases is to reach maximum coverage with minimum number of test cases and minimum test time.

To achieve this goal, the length of CFS's shall be neither too small nor too large. For simplicity, we choose an uniform $n = 2$ for all test cases.

5.1 Tests for Functional Requirements

5.1.1 Module 1: Basic comparison function

The tests here are selected to cover the tolerated comparison function and its base, subtraction operation and amplitude function. The tolerated comparison function will be used in later tests to compare the resulted CFS of

[2]Available at `https://github.com/catchorg/Catch2`

the tested function with its (pseudo-)oracle counterpart, so we need to test this function first to ensure that the following test results are reliable.

NOTE: Do not proceed with other modules unless you have succeed in this module. For each round of test, the result of other modules is trustworthy if and only if the test results of this module are all successes.

1. Test of subtraction:

   Type: Automatic

   Initial State: The subtract function of the loaded FSL library.

   Input: $CFSf = [n = 2, \omega = 1.0, A = \{1.0, 0.0, 0.0\}, B = \{0.0, 1.0\}]$ and $CFSg = [n = 2, \omega = 1.0, A = \{0.0, 2.0, 1.0\}, B = \{1.0, 0.0\}]$

   Output: $CFSf - CFSg$, which should be $[n = 2, \omega = 1.0, A = \{1.0, -2.0, -1.0\}, B = \{-1.0, 1.0\}]$

   Test Case Derivation: Manual computation

   How test will be performed: Feed function with the aforementioned input, and compare the function output to the aforementioned standard output by comparing its n, ω, A_i's and B_i's variable-by-variable.

2. Test of amplitude function:

   Type: Automatic

   Initial State: None.

   Input: $CFSf = [n = 2, \omega = 1.0, A = \{1.0, 0.0, 2.0\}, B = \{2.0, 0.0\}]$ and $\epsilon = 10^{-6}$.

   Output: $|Amp(CFSf) - 3.0| \leq \epsilon$, which should be `True`.

   Test Case Derivation: Manual computation

   How test will be performed: Compute and compare.

3. Test of tolerated comparison (`True` result):

   Type: Automatic

   Initial State: Verified amplitude function.

Input: $\mathit{CFSf} = [n = 2, \omega = 1.0, A = \{1.0, 0.0, 0.0\}, B = \{0.0, 1.0\}]$, $\mathit{CFSg} = [n = 2, \omega = 1.0, A = \{0.0, 2.0, 1.0\}, B = \{1.0, 0.0\}]$, and error $\epsilon = 10.0$

Output: Comparison result which should be `True`

Test Case Derivation: Manual computation

How test will be performed: Compute and compare.

4. Test of tolerated comparison (`False` result):

   Same as *Test of tolerated comparison (`True` result)*, but with $\epsilon = 1.0$ and Output as `False`.

5.1.2 Module 2: Fourier transformation and approximation

This module tests the functions that compute Fourier transformation and approximated values of functions.

1. Test of coefficient (even function):

   Type: Automatic

   Initial State: Verified tolerated equality function.

   Input: $f(t) = t^2$, $\omega = 1$, $n = 2$, $\epsilon = 10^{-6}$, $\mathit{CFSstd} = [n = 2, \omega = 1, A = \{\pi^2/3, -4.0, 1.0\}, B = \{0.0, 0.0\}]$

   Output: $\mathit{TolEq}(\mathit{CFSf}, \mathit{CFSstd}, \epsilon)$, which should be `True`.

   Test Case Derivation: Manual Computation, verified by MATLAB computation.

   How test will be performed: Compute and compare.

2. Test of coefficient (odd function):

   Type: Automatic

   Initial State: Verified tolerated equality function.

   Input: $f(t) = t$, $\omega = 1$, $n = 2$, $\epsilon = 10^{-6}$, $\mathit{CFSstd} = [n = 2, \omega = 1, A = \{0.0, 0.0, 0.0\}, B = \{-2.0, 1.0\}]$

Output: $\mathit{TolEq}(\mathit{CFSf}, \mathit{CFSstd}, \epsilon)$, which should be `True`.

Test Case Derivation: Manual Computation, verified by MATLAB computation.

How test will be performed: Compute and compare.

3. Test of approximated function value:

   Type: Automatic

   Initial State: None.

   Input: $\mathit{CFSf} = [n = 2, \omega = 1, A = \{0.0, 2.0, 0.0\}, B = \{-2.0, 1.0\}]$, $t = \pi/4$, $\epsilon = 10^{-6}$.

   Output: $|\mathit{App}(\mathit{CFSf}, t) - (2 + \sqrt{2}/2)| \leq \epsilon$, which should be `True`.

   Test Case Derivation: Manual Computation, verified by MATLAB computation.

   How test will be performed: Compute and compare.

5.1.3 Module 3: Operations and functions

1. Test of addition:

   Type: Automatic

   Initial State: Verified tolerated equality function.

   Input: $\mathit{CFSf} = [n = 2, \omega = 1, A = \{0.0, 2.0, 0.0\}, B = \{-2.0, 0.0\}]$, $\mathit{CFSg} = [n = 2, \omega = 1, A = \{1.0, 0.0, 2.0\}, B = \{0.0, 1.0\}]$, $\mathit{CFSstd} = [n = 2, \omega = 1, A = \{1.0, 2.0, 1.0\}, B = \{-2.0, 1.0\}]$, $\epsilon = 10^{-6}$

   Output: $\mathit{TolEq}(\mathit{CFSf} + \mathit{CFSg}, \mathit{CFSstd}, \epsilon)$, which should be `True`.

   Test Case Derivation: Manual Computation.

   How test will be performed: Compute and compare.

2. Test of multiplication:

   Type: Automatic

   Initial State: Verified tolerated equality function.

Input: $\mathit{CFSf} = [n = 2, \omega = 1, A = \{1.0, 0.0, 1.0\}, B = \{1.0, 0.0\}]$, $\mathit{CFSg} = [n = 2, \omega = 1, A = \{1.0, 1.0, 0.0\}, B = \{1.0, 0.0\}]$, $\mathit{CFSstd} = [n = 2, \omega = 1, A = \{2.0, 2.0, 0.0\}, B = \{2.0, 1.0\}]$, $\epsilon = 10^{-6}$

Output: $\mathit{TolEq}(\mathit{CFSf} * \mathit{CFSg}, \mathit{CFSstd}, \epsilon)$, which should be `True`.

Test Case Derivation: Manual Computation.

How test will be performed: Compute and compare.

3. Test of division:

   Same as *Test of multiplication*, but swap *CFSf* and *CFSstd*, while change $*$ to $/$.

4. Test of function of CFS:

   Type: Automatic

   Initial State: Verified tolerated equality, addition and multiplication function.

   Input: $\mathit{CFSf} = [n = 2, \omega = 1, A = \{1.0, 0.0, 1.0\}, B = \{1.0, 0.0\}]$, $g(t) = e^t$, and $\epsilon = 10^{-6}$.

   Output: $\mathit{TolEq}(g(\mathit{CFSf}), 1 + \mathit{CFSf} + 0.5\mathit{CFSf}^2, \epsilon)$, which should be `True`.

   Test Case Derivation: Manual Computation.

   How test will be performed: Compute and compare.

5.1.4 Test of conversion

1. Test of conversion from other data format:

   Type: Automatic

   Initial State: Verified tolerated equality function

   Input: $n = 2$, $\omega = 1$, $A = 1.0, 0.0, 2.0$, $B = 2.0, 0.0$.

   Output: constructed CFS

   Test Case Derivation: None

How the test will be performed: convert and compare the result with a standard CFS using tolerated equality and $\epsilon = 10^{-6}$.

2. Test of conversion to other data format:

   Same as *Test of conversion from other data format*, while exchanging input and output, and compare each coefficients with tolerance as ϵ.

5.2 Tests for Nonfunctional Requirements

5.2.1 Speed evaluation

We test the speed of IM1, IM3, IM4, IM5, IM8 and IM9.

For each test, generate random CFS's with same ω and various n, clock the execution time with the generated CFS's as input, and check whether the relationship between n and the execution time follows the requirements in the NFR.

We will demonstrate how the test is performed by showing one example.

1. Test the speed of addition

   Type: Automatic

   Initial State: Addition

   Input/Condition: Various pair of CFS's, with the same ω, $n = 100 : 100 : 1000$ respectively, and A_i's and B_i's generated from a random number generator. In this test, this generator is chosen as the uniform random number generator on $[-1.0, 1.0]$.

   Output/Result: Whether the execution time of addition is linear in regards to n.

   How test will be performed: For each n, generate 10 pairs of CFS's, clock their execution time to take average value as the execution time for CFS's of size n, find the r value of a linear fitting between n and the execution time. This test succeeds when $|r| > 0.9$, otherwise fails.

5.3 Traceability Between Test Cases and Modules

We show each instance module's covering tests.

- IM1: *Test of transformation (even function)* and *Test of transformation (odd function)*.
- IM2: *Test of approximated function value*
- IM3: *Test of addition*
- IM4: *Test of subtraction*
- IM5: *Test of multiplication*
- IM6: *Test of division*
- IM7: *Test of function of CFS*
- IM8: *Test of amplitude*
- IM9: *Test of tolerated equality*
- IM10: *Test of conversion from other data format*
- IM11: *Test of conversion to other data format*

References