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

Bo Cao

December 16, 2019

1 Revision History

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

Contents

2 Symbols, Abbreviations and Acronyms

Some symbols, abbreviations and acronyms are defined in the Software Requirement and Specification (SRS) document [1]. For simplicity and maintainability, they are not redefined here. Readers shall refer to the SRS 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/SRS.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 SRS 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 the division operation of CFS's contain solving of linear equations. We plan to use existing libraries for these tasks. 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 author of this library, Bo Cao, will do all the Unit VnV, including the design, coding and execution of the Automated Testing.

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's Fourier Transformation library.

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 unit test cases of this library will be based on the system test cases of this library, with the following additions and modifications.

5.1 Tests for Functional Requirements

Due to the nature of the library, the test of most functions are defined in the System VnV. We will not elaborate on these defined functions.

We should also notice that these three functions must be tested in order, and these functions must be tested before other functions: `Subtraction`, `Amplitude`, `ToleratedEquality`.

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

5.1.1 Module 1: Linear Solver

1. Test of `LinSolve`:

   Type: Automatic
   Initial State: Loaded library
   Input: $m = 2, M = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right], b = \left[\begin{smallmatrix} 3 \\ 1 \end{smallmatrix}\right]$ Output: a, whose oracle is$\left[\begin{smallmatrix} 2 \\ 1 \end{smallmatrix}\right]$ Test Case Derivation: output shall match oracle exactly. How the test will be performed: Unit Test will call the function with the input, and compare the output with the oracle.

5.1.2 Module 2: Integral

1. Test of `Integral`:

   Type: Automatic
   Initial State: Loaded library
   Input: $f = t \to t, a = 0, b = 1$ Output: *res*, whose oracle is 0.5. Test Case Derivation: difference between oracle and output shall be smaller than $\epsilon = 10^{-6}$. How the test will be performed: Unit Test will call the function with the input, and compare the output with the oracle (called Compare later.)

5.1.3 Module 3: Conversion

The unit test is the same as the system test, and we give a table describing which system test is for which function.

Test name	Function
Test of conversion from other data format	`ConvertFrom`
Test of conversion to other data format	`ConvertTo`

5.1.4 Module 4: Transformation

The unit test is the same as the system test, and we give a table describing which system test is for which function.

Test name	Function
Test of coefficient (even function)	`Transform`
Test of coefficient (odd function)	
Test of approximated function value	`FunctionValue`

5.1.5 Module 5: Basic Operation

The unit test is the same as the system test except for the `CFSMatch` function, and we give a table describing which system test is for which function.

Test name	Function
Test of addition	`Addition`
Test of subtraction	`Subtraction`
Test of multiplication	`Multiplication`
Test of division	`Division`
Test of amplitude	`Amplitude`

The test template for the `CFSMatch` function is

1. Test of `CFSMatch`:

   Type: Automatic
   Initial State: Loaded library
   Input: $CFSf = [n = 2, \omega = 1.0, A = \{0, 0, 0\}, B = \{0, 0\}]$, $CFSg = [n = a, \omega = b, A = \{...\}, B = \{...\}]$. The length of A and B in $CFSg$ shall be decided in accordance with n, and the values of elements in A and B shall all be 0.
   Output: *res*, whose oracle is c.
   Test Case Derivation: output shall fully match the oracle.
   How the test will be performed: Unit Test will call the function with the input, and compare the output with the oracle.

We generate test cases from this template by doing the following replacements.

Note: For the sake of clarity, we can also choose other sets of data for the test of multiplication. For each set of data described

Case No.	a	b	c
1	2	1.0	`true`
2	2	0.5	`false`
3	1	1.0	`false`
4	1	0.5	`false`

below, users can generate a test case by replacing the data in the original test case with the data in each set.

Case No.	A of CFSf	B of CFSf	A of CFSg	B of CFSg	A of CFSres	B of CFSres
1	$\{1, 0, 0\}$	$\{0, 0\}$	$\{1, 0, 0\}$	$\{0, 0\}$	$\{0.5, 0, 0.5\}$	$\{0, 0\}$
2	$\{1, 0, 0\}$	$\{0, 0\}$	$\{0, 0, 0\}$	$\{0, 1\}$	$\{0, 0, 0\}$	$\{0.5, 0\}$
3	$\{0, 1, 0\}$	$\{0, 0\}$	$\{0, 0, 0\}$	$\{1, 0\}$	$\{0, 0, 0\}$	$\{-0.5, 0\}$
4	$\{0, 0, 0\}$	$\{1, 0\}$	$\{0, 0, 0\}$	$\{1, 0\}$	$\{0.5, 0, -0.5\}$	$\{0, 0\}$

5.1.6 Module 6: Advanced Operations

The unit test is the same as the system test except for the `Power` function, and we give a table describing which system test is for which function.

Test name	Function
Test of tolerated equality function `TolEq` (`True` result)	`ToleratedEquality`
Test of tolerated equality function `TolEq` (`False` result)	
Test of function of CFS	`Function`

Note: e^{CFS} is `Function(CFS, expDeriv)`.
The test case template for the `Power` function is

1. Test of `Power`:
   Input: $CFSf = [n = 3, \omega = 1.0, A = a, B = b]$, $m = 3$
   Output: `Power(CFSf, m)`, whose oracle is $CFSstd = [n = 3, \omega = 1.0, A = a_o, B = b_o]$
   Test Case Derivation: `ToleratedEquality(Power(CFSf, m), CFSstd, err)`, which should be `true` when $err = 10^{-6}$.

We generate two test cases from this template with the following sets of values.

Case No.	A	B	A_o	B_o
1	$\{0, 1, 0, 0\}$	$\{0, 0, 0\}$	$\{0, 0.75, 0, 0.25\}$	$\{0, 0, 0\}$
2	$\{0, 0, 0, 0\}$	$\{1, 0, 0\}$	$\{0, 0, 0, 0\}$	$\{0, 75, 0, -0.25, 0\}$

5.2 Tests for Nonfunctional Requirements

Same as the System VnV.

5.3 Traceability Between Test Cases and Modules

Since the test cases are already grouped in accordance with their module, this section is not necessary.

References