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

Bo Cao

November 18, 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 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.

3 General Information

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.

3.1 Summary

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 Objectives

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.

3.3 Relevant Documentation

As we said before, this document relies on the CA document. This document is also the base of the Unit Test Plan document.

4 Plan

[There should usually be text between section headings. In this case, a "roadmap" of the subsections in this section would be appropriate. —SS]

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. [You should specifically list the class mates that are assigned to review your documents. You should also list the course instructor. —SS]

4.2 CA Verification Plan

The verification of the CA document mainly consists of the feedback from reviewers, and the author's experience in developing and verifying this library. [A task directed verification plan could be much more effective than this ad hoc approach. —SS]

4.3 Design Verification Plan

The design of this library will be verified by reviewing how the functions in this library relies on each other, and how they are integrated. [This is not a very detailed plan. Did you consider other options? Some of your colleagues, like Peter, have a detailed plan for design verification. —SS]

4.4 Implementation Verification Plan

The verification of the implementation of this library is mainly done by unit testing. The detail of unit testing can be found in the Unit Test Plan document. Mainly, the unit test will be done by first testing the basic functions in this library, and then testing the advanced functions. Please note that the test result of any function in this library is acceptable, if and only if the reliant functions of this function is tested to be right.

4.5 Software Validation Plan

The transformation part of this library will be validated by comparing its result with the MATLAB pseudo-oracle. [Where does the Matlab pseudo-oracle come from? This is a good idea, but you should take it a step further and explain the source of the pseudo-oracle. —SS]

5 System Test Description

5.1 Tests for Functional Requirements

All tests in this section will be done by unit testing, the detail of which will be covered in the Unit Test VnV Plan document.

Since these tests involve comparing floating point numbers, and there are intrinsic errors regarding floating point computations, we have to set

a tolerance, so that two floating numbers with difference smaller than this tolerance are considered equal. We use ϵ to represent this tolerance, and choose its value as 10^{-6} for all tests. This value is chosen because it is large enough to include all computation-related error, but small enough to detect any error in the result of the library. This value can be changed if we have good reasons to support this change. [Yes, using a symbolic constant for this is a good idea. I suggest that you put this constant in a table in the Appendix to this report, so that it is easy to maintain. I see that you are varying ϵ throughout this document. You should say this here, and not follow my advice to add a table of constants at the end, since the value is not constant. —SS]

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 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: Evaluated result of $CFSf - CFSg$, which should be a CFS object $[n = 2, \omega = 1.0, A = \{1.0, -2.0, -1.0\}, B = \{-1.0, 1.0\}]$

   Test Case Derivation: Manual computation [Why do this manually? This is a good test for automation. A unit testing framework can take care of the details. I see that many of the following tests are manual. Please consider making all of your tests automatic. —SS]

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: Evaluated result of $|Amp(CFSf) - 3.0| \leq \epsilon$, 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: $CFSf = [n = 2, \omega = 1.0, A = \{1.0, 0.0, 0.0\}, B = \{0.0, 1.0\}]$, $CFSg = [n = 2, \omega = 1.0, A = \{0.0, 2.0, 1.0\}, B = \{1.0, 0.0\}]$, and error $\epsilon = 10.0$

   Output: Comparison result which should be `True` [What is being compared? This is not clear. —SS]

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

   Output: Evaluated result of $TolEq(CFSf, CFSstd, \epsilon)$, which should be `True`. [Is this TolEq function defined somewhere? —SS]

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

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

Output: Evaluated result of $|App(CFSf, t) - (2 + \sqrt{2}/2)| \leq \epsilon$, 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: $CFSf = [n = 2, \omega = 1, A = \{0.0, 2.0, 0.0\}, B = \{-2.0, 0.0\}]$, $CFSg = [n = 2, \omega = 1, A = \{1.0, 0.0, 2.0\}, B = \{0.0, 1.0\}]$, $CFSstd = [n = 2, \omega = 1, A = \{1.0, 2.0, 1.0\}, B = \{-2.0, 1.0\}]$, $\epsilon = 10^{-6}$

   Output: Evaluated result of $TolEq(CFSf + CFSg, CFSstd, \epsilon)$, 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: $CFSf = [n = 2, \omega = 1, A = \{1.0, 0.0, 1.0\}, B = \{1.0, 0.0\}]$, $CFSg = [n = 2, \omega = 1, A = \{1.0, 1.0, 0.0\}, B = \{1.0, 0.0\}]$, $CFSstd = [n = 2, \omega = 1, A = \{2.0, 2.0, 0.0\}, B = \{2.0, 1.0\}]$, $\epsilon = 10^{-6}$

   Output: Evaluated result of $TolEq(CFSf * CFSg, CFSstd, \epsilon)$, 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: $CFSf = [n = 2, \omega = 1, A = \{1.0, 0.0, 1.0\}, B = \{1.0, 0.0\}]$, $g(t) = e^t$, and $\epsilon = 10^{-6}$.

   Output: Evaluated result of $TolEq(g(CFSf), 1 + CFSf + 0.5CFSf^2, \epsilon)$, 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 Test for input constraints

We design input test for detecting mismatch n and ω for functions that accept two CFS's as inputs. For each test in the following list, we derive tests for input constraints based on it.

- Test of addition
- Test of subtraction
- Test of multiplication
- Test of division
- Test of tolerated equality

For each test in this list, we derive two tests. One is to change the ω in the second CFS to the half of its original value, and the other is to change the n in the second CFS to 1, and remove A_2 and B_2 of the second CFS accordingly. Error message indicating a pair of mismatched CFS's shall appear.

Additionally, we do input constraint check on the tests of conversion from other data formats. We make n in the original test from 2 to 3, and error message indicating mismatched n, number of $A_i (i \neq 0)$'s, and number of B_i's.

5.3 Tests for Nonfunctional Requirements

The tests in this section will also be done by unit testing.

5.3.1 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, [be more specific on what is involved in generating the random CFS's. What is the range of values that are allowed? —SS] 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. [I think the output should be the plot of the time versus n. —SS]

   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. However, when the test fails, tester can examine the value of $|r|$, and choose to make it acceptable failure, if he/she has good reasons to do so.

[You should explain all of the tests, but you can use a table to summarize just the deltas between the base test and the individual tests. —SS]

[What about other nonfunctional tests? Usability could be a valuable thing to assess. Portability? (I think this would be a fairly easy one to assess.) Installability? —SS]

5.4 Traceability Between Test Cases and Modules

We show each instance module's covering tests. [You don't have modules. I think you mean requirements? —SS]

- 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*

[We discussed that your unit verification and validation plan will be identical to the system plan. I agree there is overlap, but I expect that you will have unique tests for the unit case. For instance, if you introduce a data structure for the CFSs, then you want to be able to check each of the methods for the CFSs. —SS]