

[Thank you :) —Malavika]

CFS: System Verification and Validation Plan

[good to see the library name in the title —SS]

Malavika Srinivasan

December 6, 2018

1 Revision History

Date	Version	Notes
Oct 16, 2018	1.0	First draft by Malavika
Oct 23, 2018	2.0	Second draft by Malavika
Dec 6, 2018	3.0	Third draft based on suggestions from Dr.Smith.

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
MG	Module Guide
MIS	Module Interface Specification

Also see the table of symbols in CA at: <https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf>

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	General Information	1
3.1	Summary	1
3.2	Objectives	1
3.2.1	Functional Suitability	2
3.2.2	Maintainability	2
3.2.3	Portability	2
3.3	References	3
4	Plan	3
4.1	Verification and Validation Team	3
4.2	SRS Verification Plan	3
4.3	Design Verification Plan	3
4.4	Implementation Verification Plan	4
4.5	Software Validation Plan	4
5	System Test Description	4
5.1	Tests for Functional Requirements	4
5.1.1	Input Testing	4
5.1.2	Interpolation Testing	6
5.1.3	Regression Testing	12
5.2	Tests for Nonfunctional Requirements	14
5.3	Traceability Between Test Cases and Requirements	16
6	Static Verification Techniques	17
7	Appendix	18
7.1	Symbolic Parameters	18

List of Tables

1	Requirements Traceability Matrix	17
---	--	----

Table name	description
Table 1	Requirement Traceability matrix

[if you don't have any figures, you can comment out this heading —SS][I
commented out listoffigures —Malavika]

This document explains the verification and validation plan to improve the quality of CFS. It is organized into different sections which gives a detailed description of the CFS, in terms of its goals, objectives, essential qualities and test cases for the functional and non functional requirements as mentioned in the CA document.

3 General Information

This section explains the summary of what is being tested in this document, the objectives of verification and validation process and the references which are necessary to read this document.

3.1 Summary

This document will summarize the plan for verification and validation of CFS in compliance with the requirements specified in the CA document found at <https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf> [good to have a link to your repo. —SS][Thank you :) —Malavika]

The goal statement as found in the CA document is presented below.
“Given the set of data points, the choice of software from CFS and the variabilities of the software the CFS should:

1. Compute the parameters of the curve which is the best possible fit through the set of data points.”

3.2 Objectives

The goal of verification and validation is to improve the quality of the CFS [I recently modified the blank project template to put the (equivalent of) the famname command in a common file, which can be shared between all your files. You might want to do the same. —SS][Yes, using it. Thanks. —Malavika]and obtain confidence in the software implementation. There are several standards which define software quality. According to the quality model of ISO 9126, software quality is described as a structured set of characteristics namely - Functional suitability, Performance, Efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability (ISO).

The qualities which are important concerning CFS are correctness(functional suitability), maintainability, reusability and portability. The definitions of the above mentioned qualities are explained below.

[You still don't have lines of text that are 80 characters long (separated by hard returns) in the tex file. —SS][I am so sorry, but I changed the setting to hard return after you mentioned. It does go to the next line when my typing reached 80 column width. I am really not sure why this happens. I forgot to discuss this during today's meeting. Please let me know if it still happens, I will drop by sometime and see what is wrong. —Malavika]

3.2.1 Functional Suitability

This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. It can be further characterized into completeness, correctness and appropriateness. In this project, we focus only on correctness which is defined as shown below.

Correctness [do you mean for this to be a subsection (not a paragraph), like the other subsections? —SS][I want to be a sub characteristic under correctness. I did not want to alter the sec depth as it would be modifying the structure, so I used paragraph. But it was intended to be subsubsubsection. —Malavika] The degree to which a product or system provides the correct results with the needed degree of precision.

3.2.2 Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in the environment, and in requirements.

3.2.3 Portability

The degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.

3.3 References

Throughout this document, we refer to the terminologies that have been already explained in the CA document for Program CFS. The CA document can be found at <https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf> [Reference your SRS. —SS][Yes, done. —Malavika]

4 Plan

4.1 Verification and Validation Team

1. Malavika srinivasan
2. Dr. Spencer Smith
3. Hanane Zlitni

4.2 SRS Verification Plan

The CA document for CFS will be reviewed by Dr. Spencer Smith [L^AT_EX has a rule that it inserts two spaces at the end of a sentence. It detects a sentence as a period followed by a capital letter. This comes up, for instance, with Dr. Smith. Since the period after Dr. isn't actually the end of a sentence, you need to tell L^AT_EX to insert one space. You do this either by Dr. Smith (if you don't mind a line-break between Dr. and Smith), or Dr. Spencer Smith (to force L^AT_EX to not insert a line break). —SS] and my classmate Robert White.

4.3 Design Verification Plan

My design will be verified with the help of my supervisor Dr. Spencer Smith and my classmates. My MG (Module Guide) will be verified by Jennifer Garner and my MIS (Module Interface Specification) will be verified by Brooks MacLachlan. [Good to list the specific people. You should also split this up between the MG and the MIS. —SS][Thank you. Changes done. —Malavika]

4.4 Implementation Verification Plan

My implementation will be verified against the tests listed in this document and the Unit Verification and Validation plan document.

4.5 Software Validation Plan

Not Applicable [why not? —SS][I think my software can be validated against the content in chapter 3 and 7 of Health1997. Is that ok to write? —Malavika]

5 System Test Description

System testing is a process in which we test the overall working of the system. The instance models in CA document will be tested here. This does not test the individual units or modules of the system. It is a black box testing approach.

5.1 Tests for Functional Requirements

In this section, we present the test cases which are related to the functional requirements of CFS. The functional requirements can be found at <https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf> The test cases below are aimed at testing the overall working of the system which is represented by the functional requirements. [It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good. —SS][Added —Malavika]

5.1.1 Input Testing

In this section, we will present the test cases for the input to CFS. The test cases are selected based on the constraints for the input data which is recorded as assumptions in the CA document found at <https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf> The typical constraints on the input data for CFS is associated with the data type and length of the input data. Some of the inputs such as the degree of the polynomial have mathematical restrictions of being a natural number.

The test cases listed below are designed to ensure that the input data to CFS does not violate these constraints.

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good. —SS]

1. **T1: Test case for inputs with less than 2 points**

Control: Automatic

Initial State: NA

Input: $t=[1]$, $y=[2]$ (length 1)

Output: Error message (“Please enter at least [proof read —SS][Corrected —Malavika] 2 points”)

How test will be performed: Pytest

2. **T2: Test case for data type mismatch**

Control: Automatic

Initial State: NA

Input: $t=[1,2,3,4,t]$, $y=[2,2,3,4,5]$

Output: Error message (“Please enter only numbers”) or TypeError exception

How test will be performed: Pytest

3. **T3: Test case for unordered inputs**

Control: Automatic

Initial State: NA

Input: $t=[1,2,5,4]$, $y=[2,0,3,4]$

Output: Error message (“Wrong input: t_{i+1} must be greater than t_i ”)

How test will be performed: Pytest

5.1.2 Interpolation Testing

In this section, we will represent the test cases for each access program present in the interpolation module. This module has access programs to find the coefficients of the interpolated curve through a set of points (t_i, y_i) for $i = 0$ to n and find the value of the interpolating polynomial at a given ‘ t ’ value.

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good. —SS]

4. **T4: I^{st} test case for Monomial interpolation**

Control: Automatic

Initial State: NA

Input: Data points $t = [0,1,2]$, $y = [0,1,2]$

Output: $[0, 1]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pyunit

Test case derivation:

[How did you come up with the output? How does the reader verify this? A similar comment applies elsewhere in your documentation. —SS]

5. **T5: II^{nd} test case for Monomial interpolation**

Control: Automatic

Initial State: NA

Input: Data points $t = [-2,0,1]$, $y = [-27,-1,0]$

Output: $[-1,5,-4]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: Page 314, example 7.1 of Heath (2002)

6. **T6: I^{st} test case for Lagrange's interpolation**

[You list first and second test cases, but why were they selected? Why does the second test case cover something different from the first? —SS]

Control: Automatic

Initial State: NA

Input: Data points $t = [0,1,2]$, $y = [0,1,2]$

Output: $[0, 1]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pyunit [You repeat this same test case several times, but for different algorithms. This is a good idea, but you should record this in a more efficient way. You can list the test case once and then reference it in the subsequent cases and simply say that it will be repeated, but for a different algorithm. You could summarize a much higher number of test cases if you used tables. —SS]

7. **T7: II^{nd} test case for Lagrange's interpolation**

Control: Automatic

Initial State: NA

Input: Data points $t = [-2,0,1]$, $y = [-27,-1,0]$

Output: $[-1,5,-4]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: Page 314, example 7.1 of Heath (2002)

8. **T8: I^{st} test case1 for Newton's interpolation**

Control: Automatic

Initial State: NA

Input: Data points $t = [0,1,2]$, $y = [0,1,2]$

Output: $[0, 1]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pyunit

9. **T9: II^{nd} test case for Newton's interpolation**

Control: Automatic

Initial State: NA

Input: Data points $t = [-2, 0, 1]$, $y = [-27, -1, 0]$

Output: $[-1, 5, -4]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: Page 314, example 7.1 of Heath (2002)

10. **T10: I^{st} test case for Hermite cubic interpolation**

Control: Automatic

Initial State: NA

Input: $t = [1, 3]$, $y = [2, 1]$

Output: $[1, -5.75, 9.5, 2]$ (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: HermiteCubic (a)

11. **T11: II^{nd} test case for Hermite [Hermite —SS][Changed —Malavika] cubic interpolation**

Control: Automatic

Initial State: NA

Input: $t = [1, 2, 4, 5]$, $y = [2, 1, 4, 3]$

Output:

$[1.0, 1.0, 1.38888889, 2.0]$ in $[1, 2)$, (Coefficients of t , starting from t^0 .)

$[4.0, 4.0, 1.0, 1.0]$ in $[2, 4)$, (Coefficients of t , starting from t^0 .)

$[3.0, 3.61111111, 4.0, 4.0]$ in $[4, 5)$, (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: HermiteCubic (b)

12. **T12: I^{st} test case for BSpline interpolation**

Control: Automatic

Initial State: NA

Input: $t = [0.0, 1.2, 1.9, 3.2, 4.0, 6.5]$, $y = [0.0, 2.3, 3.0, 4.3, 2.9, 3.1]$,
 $s=0$, $k=4$

Output:

$[-5.62048630e-18, 2.98780300e+00, -5.74472095e-01, 1.46700914e+01,$
 $-1.03253068e+01, 3.10000000e+00, 0.00000000e+00, 0.00000000e+00,$
 $0.00000000e+00, 0.00000000e+00, 0.00000000e+00]$, (Coefficients of t ,
starting from t^0 .)

How test will be performed: Pytest

Test case reference: BSpline

13. **T13: I^{st} test case for evaluating Monomial interpolation**

Type: Automatic

Initial State: NA

Input: $[0,1]$, 2

Output: 2

Test Case Derivation: Please see inputs of T4 in section 5.1.1 in System
verification and validation plan.

How test will be performed: Pytest

14. **T14: II^{nd} test case for evaluating Monomial interpolation**

Type: Automatic

Initial State: NA

Input: $[-1, 5, 4]$, -2

Output: -27

Test Case Derivation: Page 314, example 7.1 of Heath (2002).

How test will be performed: Pytest

15. **T15: I^{st} test case for evaluating Lagrange's interpolation**

Type: Automatic

Initial State: NA

Input: $[0,1,2]$, 1

Output: 1

Test Case Derivation: Please see inputs of T4 in section 5.1.1 in System verification and validation plan.

How test will be performed: Pytest

16. **T16: II^{nd} test case for evaluating Lagrange's interpolation**

Type: Automatic

Initial State: NA

Input: $[-27,-1,0]$, -2

Output: -27

Test Case Derivation: Page 314, example 7.1 of Heath (2002).

How test will be performed: Pytest

17. **T17: I^{st} test case for evaluating Newton's interpolation**

Type: Automatic

Initial State: NA

Input: $[0,1]$, 2

Output: 2

Test Case Derivation: Please see inputs of T4 in section 5.1.1 in System verification and validation plan.

How test will be performed: Pytest

18. **T18: II^{nd} test case for evaluating Newton's interpolation**

Type: Automatic

Initial State: NA

Input: $[-1, 5, 4]$, -2

Output: -27

Test Case Derivation: Page 314, example 7.1 of Heath (2002).

How test will be performed: Pytest

19. **T19: I^{st} test case for evaluating Hermite cubic interpolation**

Type: Automatic

Initial State: NA

Input:

[1, -5.75, 9.5, 2], [1,3][Interval to be read as 1 to 3 —Malavika], [1,3][Input x values —Malavika]

Output: [2,1][Output for each x value —Malavika]

Test Case Derivation: HermiteCubic (b)

How test will be performed: Pytest

20. **T20: II^{nd} test case for evaluating Hermite cubic interpolation**

Type: Automatic

Initial State: NA

Input:

[[1.0, 1.0, 1.38888889, 2.0]
[4.0, 4.0, 1.0, 1.0]
[3.0, 3.61111111, 4.0, 4.0]], [1,2,4,5][Interval to be read as 1 to 2, 2 to 4 and 4 to 5 —Malavika], [1,2,4,5] [Input x values —Malavika]

Output: [2, 1, 4, 3][Output for each x value —Malavika]

Test Case Derivation: HermiteCubic (b)

How test will be performed: Pytest

21. **T21: Test case for evaluating BSplines**

Type: Automatic

Initial State: NA

Input: [[-5.62048630e-18, 2.98780300e+00, -5.74472095e-01,
1.46700914e+01, -1.03253068e+01, 3.10000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],
t = [0.0, 1.2, 1.9, 3.2, 4.0, 6.5]

Output: [0.0, 2.3, 3.0, 4.3, 2.9, 3.1]

Test Case Derivation: BSpline

How test will be performed: Pytest

5.1.3 Regression Testing

22. **T22:** I^{st} test case for regression using normal equations

Control: Automatic

Initial State: NA

Input: $t = [0,1,2,3,4,5,6,7,8,9,10]$, $y = [0,1,2,3,4,5,6,7,8,9,10]$, $\text{deg} = 1$

Output: 0,1 (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

23. **T23:** II^{nd} test case for regression using normal equations

Control: Automatic

Initial State: NA

Input: Data points $t = [1,2,3]$, $y = [1,3,7]$, $\text{deg} = 1$

Output: -2.333333333333, 3 (Coefficients of t , starting from t^0 .)

How test will be performed: Pytest

Test case reference: NormalEquations

24. **T24:** III^{rd} test case for regression using normal equations

Control: Automatic

Initial State: NA

Input: $t = [0,200,400,600,800]$, $y = [0.0010, 0.0015, 0.0021, 0.0051, 0.0094]$, $\text{degree} = 2$

Output: 0.00116857142857, -0.00000408571429, 0.00000001785714
(Coefficients of t, starting from t^0 .)

How test will be performed: Pytest

Test case reference: NormalEquations

25. **T25: I^{st} test case for regression using augmented systems**

Control: Automatic

Initial State: NA

Input: Data points $t = [0,1,2,3,4,5,6,7,8,9,10]$, $y = [0,1,2,3,4,5,6,7,8,9,10]$, degree = 1

Output: 0,1 (Coefficients of t, starting from t^0 .)

How test will be performed: Pytest

26. **T26: II^{nd} test case for regression using augmented systems**

Control: Automatic

Initial State: NA

Input: Data points $t = [1,-1, -2]$, $y = [3,-5,12]$

Output: -8, 4, 7 (Coefficients of t, starting from t^0 .)

How test will be performed: Pytest

Test case reference: AugmentedEquations

27. **T27: I^{st} test case for regression using orthogonal transformations**

Control: Automatic

Initial State: NA

Input: Data points $t = [0,1,2,3,4,5,6,7,8,9,10]$, $y = [0,1,2,3,4,5,6,7,8,9,10]$

Output: 0,1 (Coefficients of t, starting from t^0 .)

How test will be performed: Pytest

28. **T28: II^{nd} test case for regression using orthogonal transformations**

Control: Automatic

Initial State: NA

Input: Data points $t = [1,2,3,4]$, $y = [5,3,2,1]$, degree = 1

Output: 6,-1.3 (Coefficients of t, starting from t^0 .)

How test will be performed: Pytest

Test case reference: Page 4 and 5 of OrthogonalTransformations

29. **T29: Test case for evaluating Regression**

Control: Automatic

Initial State: NA

Input: [-8, 4, 7],-2

Output: 12

How test will be performed: Pytest

Test case reference: AugmentedEquations

5.2 Tests for Nonfunctional Requirements

30. **T20: Test case for correctness**

Type: Nonfunctional, Manual, Parallel testing

Initial State: NA

Input/Condition: Results from Matlab for T4, T6, T9, T10, T12, T14, T16 and T19 will be manually compared using relative error by the formula below.

$$err = \frac{val_{CFS} - val_{Matlab}}{val_{CFS}}$$

$$err < Admissible_error$$

[good test case —SS]

ADMISS_ERR is available in appendix section. [This isn't a good symbol. A constant like this should be in all caps and it won't space out properly. Use something like *ADMISS_ERR*. —SS]

Output/Result: Pass/Fail [The output should be the expected result. Since you are summarizing multiple test cases at once, you could think of your output as a table of all of the relative errors, or you could report the maximum relative error (infinity norm.) —SS]

How test will be performed: Manual

31. **T21: Test case for maintainability**

Type: Nonfunctional, Manual

Initial State: NA

Input: Module guide, Module Interface specification

Steps:

- (a) Choose participants
- (b) Assign a task such as changing the secret of a module
- (c) Give the participants MG and MIS
- (d) Ask them to find, which module undergoes change.

This will help in accessing the maintainability of the software by measuring the ability to undergo change.

Output: Pass/ Fail

How test will be performed: Manual [I like this test case experiment. —SS]

32. **T22: Test case for Portability**

Type: Nonfunctional, Manual

Initial State: NA

Input/Condition: Try and run CFS in Mac, Windows and Linux using virtual machines.

Output/Result: Pass/Fail

How test will be performed: Manual

5.3 Traceability Between Test Cases and Requirements

The following table shows the traceability mapping for test cases, instance models and the requirements.

Table 1: Requirements Traceability Matrix

Test Number	Instance Models	CA Requirements
T1		R1, R3
T2		R1, R2
T3		R1, R3
T4	IM1,IM3	R4, R5, R6, R9, R10
T5	IM1,IM3	R4, R5, R6, R9, R10
T6	IM1, IM4	R4, R5, R6, R9, R10
T7	IM1, IM4	R4, R5, R6, R9, R10
T8	IM1,IM5	R4, R5, R6, R9, R10
T9	IM1, IM5	R4, R5, R6, R9, R10
T10	IM2, IM6	R4, R5, R7, R9, R10
T11	IM2, IM6	R4, R5, R7, R9, R10
T12	IM2, IM7	R4, R5, R7, R9, R10
T13	IM8	R4, R8, R9, R10
T14	IM8	R4, R8, R9, R10
T15	IM8	R4, R8, R9, R10
T16	IM9	R4, R8, R9, R10
T17	IM9	R4, R8, R9, r10
T18	IM10	R4, R8, R9, R10
T19	IM10	R4, R8, R9, R10
T20		R12
T21		R13
T22		R11

6 Static Verification Techniques

Code walkthrough and inspection will be used to verify the implementation.

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

- $ADMISS_ERR = 1 \times 10^{-1}$

[You should write numbers properly 1×10^{-1} —SS][Changed —Malavika]

References

An overview of the iso 9126-1 software quality model definition, with an explanation of the major characteristics. URL https://en.wikipedia.org/wiki/ISO/IEC_9126#cite_note-2.

AugmentedEquations. Regression using augmented systems in python. URL <https://math.stackexchange.com/questions/710750/find-a-second-degree-polynomial-that-goes-through-3-points>.

BSpline. Bsplines in python. URL <https://stackoverflow.com/questions/45179024/scipy-bspline-fitting-in-python>.

Michael T. Heath. *Scientific computing an introductory survey*. Boston McGraw-Hill, 2nd ed edition, 2002. ISBN 0072399104. URL <http://openlibrary.org/books/OL3946055M>. Includes bibliographical references (p. 523-548) and index.

HermiteCubic. Hermite cubic interpolation in python, a. URL <https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.interpolate.pchip.html>.

HermiteCubic. Hermite cubic interpolation in python, b. URL <https://stackoverflow.com/questions/43458414/python-scipy-how-to-get-cubic-spline-equations-from-cubicspline>.

NormalEquations. Normal equations regression in python. URL <http://www4.ncsu.edu/eos/users/w/white/www/white/ma341/lslecture.PDF>.

OrthogonalTransformations. Regression using orthogonal transformations in python. URL http://www.maths.lse.ac.uk/personal/james/old_ma201/solns8.pdf.