CFS: System Verification and Validation Plan

Malavika Srinivasan

October 27, 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

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

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 . . . 2

4 Plan **2**
- 4.1 Verification and Validation Team . . . 2
- 4.2 SRS Verification Plan . . . 2
- 4.3 Design Verification Plan . . . 3
- 4.4 Implementation Verification Plan . . . 3
- 4.5 Software Validation Plan . . . 3

5 System Test Description **3**
- 5.1 Tests for Functional Requirements . . . 3
  - 5.1.1 Input testing . . . 3
  - 5.1.2 Interpolation Testing . . . 4
  - 5.1.3 Regression . . . 7
- 5.2 Tests for Nonfunctional Requirements . . . 10
- 5.3 Traceability Between Test Cases and Requirements . . . 11

6 Static Verification Techniques **12**

7 Appendix **14**
- 7.1 Symbolic Parameters . . . 14

List of Tables

1 Requirements Traceability Matrix . . . 12

List of Figures

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 along with its goals and objectives, qualities which are important for CFS and test cases for the functional and non functional requirements mentioned in the CA document.

3 General Information

This section explains the summary of what is being tested in this document, the objectives of this document and references for 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 found at https://github.com/Malavika-Srinivasan/CAS741/tree/master/docs/SRS/CA.pdf 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 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, re-usability and portability. The definitions of the above mentioned qualities are explained below.

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 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.

4 Plan

4.1 Verification and Validation Team

1. Malavika srinivasan

4.2 SRS Verification Plan

The CA document for CFS will be reviewed by Dr.Spencer Smith and my classmate Mr.Robert White.

4.3 Design Verification Plan

My design will be verified with the help of my supervisor Dr.Spencer Smith and my classmates Ms.Jennifer Garner and Mr.Brooks MacLachlan.

4.4 Implementation Verification Plan

My implementation will be verified by the tests listed in this document and the unitVnVplan document. My classmate Ms.Vajiheh Motamer will help me verify the document.

4.5 Software Validation Plan

Not Applicable

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

5.1.1 Input testing

1. **T1: I^{st} test case for faulty inputs**

   Control: Automatic

   Initial State: NA

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

   Output: Error message ("Please enter atleast 2 points")

   How test will be performed: Pytest

2. **T2: II^{nd} test case for faulty inputs**

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")

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

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

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

   Control: Automatic

   Initial State: NA

   Input: Data points x = [-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 (1997)

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

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]

   How test will be performed: Pytest

   Test case reference: Page 314, example 7.1 of Heath (1997)

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 (1997)

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: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.PchipInterpolator.html#scipy.interpolate.PchipInterpolator

11. **T11: II^{nd} test case for hermite 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: https://stackoverflow.com/questions/43458414/python-scipy-how-to-get-cubic-spline-equations-from-cubicspline
i

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

    Control: Automatic

    Initial State: NA

    Input: x = [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]

    How test will be performed: Pytest

    Test case reference: https://stackoverflow.com/questions/45179024/scipy-bspline-fitting-in-python

5.1.3 Regression

13. **T13: 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], deg = 1

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

    How test will be performed: Pytest

14. **T14: 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], deg = 1

Output: -2.33333333333, 3 (Coefficients of t, starting from t^0.)

How test will be performed: Pytest

Test case reference: http://www4.ncsu.edu/eos/users/w/white/www/white/ma341/lslecture.PDF

15. **T15: 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], 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: http://www4.ncsu.edu/eos/users/w/white/www/white/ma341/lslecture.PDF

16. **T16: 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

17. **T17: 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: https://math.stackexchange.com/questions/710750/find-a-second-degree-polynomial-that-goes-through-3-points

18. **T18: II^{nd} 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

19. **T19: 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 http://www.maths.lse.ac.uk/personal/james/old_ma201/solns8.pdf

5.2 Tests for Nonfunctional Requirements

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

    Type: Nonfunctional, Manual

    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.

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

$$err < Admissible\_error$$

    Output/Result: Pass/Fail

    How test will be performed: Manual

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

    Type: Nonfunctional, Manual

    Initial State: NA

    Input: Module guide, Module Interface specification

    Steps:

    (a) Choose a task such as changing the output type of a module

    (b) Give the participants MG and MIS

    (c) Ask them to find, which module undergoes change.

    Output: Pass/ Fail

    How test will be performed: Manual

22. **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.

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.

Michael T. Heath. *Scientific computing.* Mcgraw-Hill, 1997. ISBN 0071153365 9780071153362.

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.

- $Admissible\_error = 1e - 1$

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.

Michael T. Heath. *Scientific computing.* Mcgraw-Hill, 1997. ISBN 0071153365 9780071153362.