Slope Stability Analysis: Unit Verification and Validation Plan for SSP

Brooks MacLachlan

November 27, 2018

1 Revision History

Date	Version	Notes
2018/11/26	1.0	Initial template fill-ins

Contents

List of Tables

[Do not include if not relevant —SS]

List of Figures

[Do not include if not relevant —SS]

2 Symbols, Abbreviations and Acronyms

The symbols, abbreviations, and acronyms used in this document include those defined in the table below, as well as any defined in the tables found in Section 2 of the Software Requirements Specification (SRS) document.

symbol	**description**
j	index representing a single coordinate
MIS	Module Interface Specification
MG	Module Guide
TC	Test Case
VnV	Verification and Validation

This document provides the unit Verification and Validation (VnV) plan for the software. General information related to the system under test is given in Section 3. Section 4 outlines at a high level the plan for verifying and validating the software. Section 5 gives more detail about the specific tests that will be used to verify each module.

3 General Information

3.1 Purpose

The software being tested is the Slope Stability analysis Program (SSP). Based on user-defined slope geometry and material properties, SSP determines the critical slip surface of the given slope, the corresponding factor of safety, and interslice normal and shear forces along the critical slip surface. The purpose of the unit verification and validation activities is to confirm that every module of SSP performs its expected actions correctly. The tests described in this document cannot definitively prove correctness, but they can build confidence by verifying that the software is correct for the cases covered by tests.

3.2 Scope

M1 will not be unit tested as it is implemented by the operating system of the hardware on which SSP is running, and is assumed to work correctly. M11, M12, and M13 will also not be unit tested as they are all implemented by MatLab. Verification of the non-functional requirements is not included in the unit verification plan because they are sufficiently covered by the system tests outlined in the System VnV Plan document.

4 Plan

4.1 Verification and Validation Team

Brooks MacLachlan is responsible for the unit verification and validation of SSP, though input from various students and the professor, Dr. Spencer Smith, of CAS 741 will also contribute.

4.2 Automated Testing and Verification Tools

MatLab's built-in unit testing framework will be used to automatically run the unit tests and display the results.

[Is it inherently better to use a unit testing framework if you already have a lot of tests written without one? —BM]

4.3 Non-Testing Based Verification

Not applicable for SSP.

5 Unit Test Description

Test cases have been selected to verify that each module conforms to the specification for the module described in the Module Interface Specification (MIS) document. Where the MIS included conditional rules, at least one test case covers each branch of the conditional rule. Test cases are minimal, meaning that each test case verifies only one value. If the MIS for a module includes several results, there is a test case for each result, even if they all cover the same branch of a conditional. Throughout this section, if a test is verifying equality between two numbers, a relative tolerance for difference between the actual and expected values will be allowed. The tolerance will be described after running the tests.

5.1 Tests for Functional Requirements

5.1.1 Control Module

[Should control module be out of scope? All it does is call other modules. Maybe I could unit test that the expected functions were called, but is that valuable? —BM]

5.1.2 Input Module

As described in the MIS, the Input module is expected to read in many user inputs from a file. For each value contained in the file, there is a corresponding test case verifying that the value was properly read into the data structure containing the input parameters. In cases where the user input may take different forms, such as the input for when a water table exists and the input for when a water table does not exists, each potential form of input is covered by at least one test case. The input module is also responsible for verifying the input, so for each possible violation of an input constraint, there is a corresponding test case verifying that the correct exception was thrown. The values in Table 1 will be used as input for many of the test cases described throughout this section. These values were taken from the User's Guide for this project by Karchewski (8 January 2012). Individual test cases will reference the table as input but specify new values for any input parameter that should have a different value than specified by the table.

Valid User Input

The test cases described in Table 2 verify that each user input is correctly read. These test cases are identical to each other with the exception of the expected output which they assert. The input for each is a file containing the inputs specified in Table **??**.[Should the input be more specific and give the actual name of a file, or is this okay? —BM] The type of these test cases is automatic. The initial state for each is a new session. The expected output for each is given in Table 2. The expected output is derived based on the given inputs. The tests will be performed as automated tests on a unit testing framework.

Test Case	**Test Name**	**Expected Output**
TC1	test-input_slope	$slope.strat$ = [(0, 25), (20, 25), (30, 20), (40, 25), (70, 25)]
TC2	test-input_phi	$slope.phi$ = 0.34906585
TC3	test-input_coh	$slope.coh$ = 5000
TC4	test-input_gam	$slope.gam$ = 15000
TC5	test-input_gams	$slope.gams$ = 15000
TC6	test-input_piez	$piez.piez$ = [(0, 22), (10.87, 21.28), (21.14, 19.68), (31.21, 17.17), (38.69, 14.56), (40, 14), (70, 14)]
TC7	test-input_gamw	$piez.gamw$ = 9800
TC8	test-input_xExtMin	$search.Xext[0]$ = 34

TC9	test-input_xExtMax	$search.Xext[1] = 53$
TC10	test-input_xEtrMin	$search.Xetr[0] = 10$
TC11	test-input_xEtrMax	$search.Xetr[1] = 24$
TC12	test-input_yLimMin	$search.Ylim[0] = 5$
TC13	test-input_yLimMax	$search.Ylim[1] = 26$
TC14	test-input_ltor	$soln.ltor = 1$
TC15	test-input_ftype	$soln.ftype = 0$
TC16	test-input_evenslc	$soln.evenslc = 1$
TC17	test-input_cncvu	$soln.cncvu = 1$
TC18	test-input_obtu	$soln.obtu = 1$

Table 2: Input Test Cases

[Can I use MIS variable names for expected output? —BM]

TC19: test-input_rtol

Type: Automatic

Initial State: New session

Input: As described in Table 1, except with slope coordinates $\{(x_{\text{us}}, y_{\text{us}})\}$ increasing as x increases, as follows: {(0, 15), (30, 15), (40, 20), (50, 25), (70, 25)}.

Output: $soln.ltor = 0$.

Test Case Derivation: Based on the given slope stratigraphy, SSP should detect that the slope elevation is increasing as x increases, and set $soln.ltor$ accordingly.

How test will be performed: Automated test on unit testing framework.

TC20: test-input_noWTpiez

Type: Automatic

Initial State: New session

Input: As described in Table 1, except with no water table.

Output: $piez.piez = []$.

Input	Unit	Value
$\{(x_{\text{us}}, y_{\text{us}})\}$	m	{(0, 25), (20, 25), (30, 20), (40, 15), (70, 15)}
$\{(x_{\text{wt}}, y_{\text{wt}})\}$	m	{(0, 22), (10.87, 21.28), (21.14, 19.68), (31.21, 17.17), (38.69, 14.56), (40, 14), (70, 14)}
$x_{\text{slip}}^{\text{minEtr}}$	m	10
$x_{\text{slip}}^{\text{maxEtr}}$	m	24
$x_{\text{slip}}^{\text{minExt}}$	m	34
$x_{\text{slip}}^{\text{maxExt}}$	m	53
$y_{\text{slip}}^{\text{min}}$	m	5
$y_{\text{slip}}^{\text{max}}$	m	26
c'	Pa	5000
φ'	$^\circ$	20
γ	$\text{N}\,\text{m}^{-3}$	15000
γ_{Sat}	$\text{N}\,\text{m}^{-3}$	15000
γ_{w}	$\text{N}\,\text{m}^{-3}$	9800
const_f	N/A	0

Table 1: Input to be used for test cases

Test Case Derivation: If the input includes no water table vertices, the *piez.piez* variable should be the empty sequence.

How test will be performed: Automated test on unit testing framework.

TC21: test-input_noWTgamw

Type: Automatic

Initial State: New session

Input: As described in Table 1, except with no water table.

Output: *piez.gamw* $= 0$.

Test Case Derivation: If the input includes no water table vertices, the *piez.gamw* variable should be 0.

How test will be performed: Automated test on unit testing framework.

Invalid User Input
See TC9 - TC36 in the System VnV Plan document.

[Is this okay? The tests described in the SystVnVPlan are really unit tests —BM]

5.1.3 Module 2

...

5.2 Tests for Nonfunctional Requirements

Not applicable for any of the modules of SSP.

5.3 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

6 References

Brandon Karchewski. Slope stability program (ssp) user's guide 1.0, 8 January 2012.

7 Appendix

[This is where you can place additional information, as appropriate —SS]

7.1 Symbolic Parameters

[The definition of the test cases may call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance. —SS]