

Slope Stability Analysis: Unit Verification and Validation Plan for SSP

Brooks MacLachlan

December 3, 2018

1 Revision History

Date	Version	Notes
11/26/2018	1.0	Initial template fill-ins
11/29/2018	1.1	Updates based on presentation feedback

Contents

1	Revision History	i
	List of Tables	iii
	List of Figures	iii
2	Symbols, Abbreviations and Acronyms	iv
3	General Information	1
3.1	Purpose	1
3.2	Scope	1
4	Plan	2
4.1	Verification and Validation Team	2
4.2	Automated Testing and Verification Tools	2
4.3	Non-Testing Based Verification	2
5	Unit Test Description	2
5.1	Tests for Functional Requirements	3
5.1.1	Input Module	3
5.1.2	Genetic Algorithm Module	6
5.1.3	Kinematic Admissibility Module	7
5.1.4	Slip Weighting Module	12
5.1.5	Slip Slicing Module	15
5.1.6	Morgenstern-Price Calculation Module	17
5.1.7	Slice Property Calculation Module	18
5.1.8	Output Module	18
5.2	Tests for Nonfunctional Requirements	18
5.3	Traceability Between Test Cases and Modules	18
6	References	18
7	Appendix	19
7.1	Symbolic Parameters	19

List of Tables

1	Input to be used for test cases	3
2	Input Test Cases	5
3	Genetic Algorithm Test Cases	7
[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

The Hardware-Hiding module, 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. The Sequence Data Structure, Random Number Generation, and Plotting modules (M11, M12, and M13) will also not be unit tested as they are all implemented by MatLab. The Control Module, M2, is out of scope for this document because it is considered to have been sufficiently tested by the system tests outlined in the [System VnV Plan document](#). Verification of the non-functional requirements is not included in the unit verification plan because they are also sufficiently covered by the System VnV Plan.

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.

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 error for difference between the actual and expected values will be allowed. The error will be described after running the tests.

Many of the tests refer to variables or constants as part of the initial state, input, or expected output. The specifications for these variables and constants can be found in the MIS document for this project.

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.

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'	°	20
γ	N m ⁻³	15000
γ_{Sat}	N m ⁻³	15000
γ_{w}	N m ⁻³	9800
$const_f$	N/A	0

Table 1: Input to be used for test cases

5.1 Tests for Functional Requirements

5.1.1 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.

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 on which they assert. The input for each is a file containing the inputs specified in Table 1. 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	<i>slope.strat</i> = [(0, 25), (20, 25), (30, 20), (40, 25), (70, 25)]
TC2	test-input_phi	<i>slope.phi</i> = 0.34906585
TC3	test-input_coh	<i>slope.coh</i> = 5000
TC4	test-input_gam	<i>slope.gam</i> = 15000
TC5	test-input_gams	<i>slope.gams</i> = 15000
TC6	test-input_piez	<i>piez.piez</i> = [(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	<i>piez.gamw</i> = 9800
TC8	test-input_xExtMin	<i>search.Xext</i> [0] = 34
TC9	test-input_xExtMax	<i>search.Xext</i> [1] = 53
TC10	test-input_xEtrMin	<i>search.Xetr</i> [0] = 10
TC11	test-input_xEtrMax	<i>search.Xetr</i> [1] = 24
TC12	test-input_yLimMin	<i>search.Ylim</i> [0] = 5
TC13	test-input_yLimMax	<i>search.Ylim</i> [1] = 26

TC14	test-input_ltor	<i>soln.ltor</i> = 1
TC15	test-input_ftype	<i>soln.ftype</i> = 0
TC16	test-input_evnslc	<i>soln.evnslc</i> = 1
TC17	test-input_cncvu	<i>soln.cncvu</i> = 1
TC18	test-input_obtu	<i>soln.obtu</i> = 1

Table 2: Input Test Cases

TC19: test-input_rtol

Type: Automatic

Initial State: New session

Input: As described in Table 1, except with slope coordinates $\{(x_{us}, y_{us})\}$ 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* = [].

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.

5.1.2 Genetic Algorithm Module

The test cases described in Table 3 verify that the critical slip surface returned by the genetic algorithm satisfies certain properties. Since the genetic algorithm includes randomness, it is impossible to verify the exact output. This is why test cases were selected to verify properties of the output instead. These test cases are identical to each other with the exception of the expected output on which they assert. The input for each is nothing; the `genetic_alg` method does not take any inputs, it uses state variables instead. The type of these test cases is automatic. As the initial state for each, $slope$, $piez$, $search$, and $soln$ state variables are loaded with the inputs from Table 1. For each test case, a predicate on the output that is expected to hold true is given for each in Table 3. These predicates include a variable called $cslip$, which represents the critical slip surface output by the genetic algorithm. The expected output is derived based on the specification for the genetic algorithm found in the MIS document. The tests will be performed as automated tests on a unit testing framework.

Test Case	Test Name	Expected Output
TC22	test-genAlg_xExtMin	$cslip[cslip - 1].x \geq search.Xext[0]$
TC23	test-genAlg_xExtMax	$cslip[cslip - 1].x \leq search.Xext[1]$
TC24	test-genAlg_xEtrMin	$cslip[0].x \geq search.Xetr[0]$
TC25	test-genAlg_xEtrMax	$cslip[0].x \leq search.Xetr[1]$

TC26	test-genAlg_yLimMin	$\forall(i : \mathbb{Z} i \in [0.. cslip - 1] : cslip[i].y \geq search.Ylim[0])$
TC27	test-genAlg_yLimMax	$\forall(i : \mathbb{Z} i \in [0.. cslip - 1] : cslip[i].y \leq search.Ylim[1])$
TC28	test-genAlg_vertices	$ cslip = (INIT_NUM_VERTICES * (2^{NUM_ADDS}) - (2^{NUM_ADDS} - 1))$

Table 3: Genetic Algorithm Test Cases

5.1.3 Kinematic Admissibility Module

Based on the specification for this module in the MIS, there are six conditions under which the module should return false. Thus, test cases have been selected to test each of these six conditions. For each condition, there is one test case verifying the behaviour when the condition is met and another test case verifying the behaviour when the slip surface is on the border of meeting the condition. For example, if the condition is that one value must be strictly less than another, there would be a test case where the two values are equal. Two of the conditions are turned on or off based on a boolean, and so an additional test case was selected for each of those to confirm that if the condition is turned off, the module should return true even if the condition is met.

TC29: test-kinAdm_xDec

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 15), (25, 15), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that the *x*-ordinates are not monotonically increasing, so the kinematic admissibility module should return false.

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

TC30: test-kinAdm_xEq

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 14), (30, 14), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the *x*-ordinates are equal for two adjacent points. This does not violate the condition, so the kinematic admissibility module should return true.

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

TC31: test-kinAdm_xEtrOnSlope

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(-10, 25), (30, 15), (35, 12), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that the *x*-ordinate of the slip surface entry is not on the slope, so the kinematic admissibility module should return false.

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

TC32: test-kinAdm_xEtrOnEdge

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(0, 25), (30, 15), (35, 14), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the x -ordinate of the slip surface entry is just on the edge of the slope, so the kinematic admissibility module should return true.

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

TC33: test-kinAdm_xExtOnSlope

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 15), (35, 12), (80, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that the x -ordinate of the slip surface exit is not on the slope, so the kinematic admissibility module should return false.

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

TC34: test-kinAdm_xExtOnEdge

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 15), (35, 12), (70, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the x -ordinate of the slip surface exit is just on the edge of the slope, so the kinematic admissibility module should return true.

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

TC35: test-kinAdm_inSlope

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 25), (35, 12), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that one x -ordinate of the slip surface is above the slope, so the kinematic admissibility module should return false.

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

TC36: test-kinAdm_onSlope

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1 and with *soln.cncvu*=0 and *soln.obtu*=0.

Input: (surf : [(20, 25), (30, 15), (34, 18), (38, 15), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that one interior x -ordinate of the slip surface is on the slope. This does not violate the condition, so the kinematic admissibility module should return true. Note that *soln.cncvu* and *soln.obtu* were set to 0 to relax the concave-up condition and obtuse-angles condition, as this test case would violate those conditions otherwise.

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

TC37: test-kinAdm_cncvUp

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1.

Input: (surf : [(20, 25), (30, 15), (32, 15), (35, 12), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that the slip surface is not fully concave-up, so the kinematic admissibility module should return false.

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

TC38: test-kinAdm_cncvUpOff

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1 and with *soln.cncvu*=0 and *soln.obtu* = 0.

Input: (surf : [(20, 25), (30, 15), (32, 15), (35, 12), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the slip surface is not fully concave-up, but *cncvu*=0, so the condition is not enforced, so the kinematic admissibility module should return true. Note that *soln.obtu* was set to 0 to relax the obtuse-angles condition, as this test case would violate that condition otherwise.

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

TC39: test-kinAdm_cncvUpEq

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1 and with *soln.cncvu*=0.

Input: (surf : [(20, 25), (30, 15), (35, 15), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the slip surface has adjacent slopes that are equal. This does not violate the condition, so the kinematic admissibility module should return true.

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

TC40: test-kinAdm_obtu

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1 and with *soln.cncvu*=0.

Input: (surf : [(20, 25), (30, 15), (31, 5), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: false.

Test Case Derivation: The input was designed so that the slip surface has an angle less than 110 degrees, so the kinematic admissibility module should return false. Note that *soln.cncvu* was set to 0 to relax the concave-up condition, as this test case would violate that condition otherwise.

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

TC41: test-kinAdm_obtuOff

Type: Automatic

Initial State: *slope*, *piez*, *search*, and *soln* state variables are loaded with the inputs from Table 1 and with *soln.cncvu*=0 and *soln.obtu*=0.

Input: (surf : [(20, 25), (30, 15), (31, 5), (40, 15)], Fs : 0, G : [], X : [], wt : 0).

Output: true.

Test Case Derivation: The input was designed so that the slip surface has an angle less than 110 degrees, but *soln.obtu*=0, so the condition is not enforced, so the kinematic admissibility module should return true. Note that *soln.cncvu* was set to 0 to relax the concave-up condition, as this test case would violate that condition otherwise.

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

5.1.4 Slip Weighting Module

As specified in the MIS for this module, it accepts a list of slip surfaces, sorts them based on their factor of safety, and assigns a weight to each slip surface. To cover this functionality, there are test cases verifying that the slip surfaces are correctly sorted, and test cases verifying that the weights are calculated correctly. A final test case verifies that the weights are correct in the special case where every slip surface has the same factor of safety.

TC42: test-weighter_sort

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 1.0, G : [], X : [], wt : 0), (surf : [], Fs : 0.8, G : [], X : [], wt : 0), (surf : [], Fs : 1.5, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:

$slips'[0].Fs = 0.8$

$\wedge slips'[0].Fs = 1.0$

$\wedge slips'[0].Fs = 1.5.$

Test Case Derivation: The input list of slip surfaces was manually sorted in ascending order of factor of safety to derive the expected output.

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

TC43: test-weighter_size

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 1.0, G : [], X : [], wt : 0), (surf : [], Fs : 1.5, G : [], X : [], wt : 0), (surf : [], Fs : 0.8, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:

$|slips'| = 3.$

Test Case Derivation: The weighter function should not change the size of the input list of slip surfaces, so the expected size of the output is equal to the size of the input.

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

TC44: test-weighter_first

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 0.8, G : [], X : [], wt : 0), (surf : [], Fs : 1.4, G : [], X : [], wt : 0), (surf : [], Fs : 1.8, G : [], X : [], wt : 0), (surf : [], Fs : 2.0, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:

$slips'[0].wt = 0.6.$

Test Case Derivation: The input list of slip surfaces was chosen to be already sorted so that this test case is independent of the sorting. The

expected output was derived by manually calculating the weight of the first slip surface according to the specification in the MIS.

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

TC45: test-weighter_mid

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 0.8, G : [], X : [], wt : 0), (surf : [], Fs : 1.4, G : [], X : [], wt : 0), (surf : [], Fs : 1.8, G : [], X : [], wt : 0), (surf : [], Fs : 2.0, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:
slips[1].wt = 0.9.

Test Case Derivation: The input list of slip surfaces was chosen to be already sorted so that this test case is independent of the sorting. The expected output was derived by manually calculating the weight of the second slip surface according to the specification in the MIS.

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

TC46: test-weighter_secondLast

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 0.8, G : [], X : [], wt : 0), (surf : [], Fs : 1.4, G : [], X : [], wt : 0), (surf : [], Fs : 1.8, G : [], X : [], wt : 0), (surf : [], Fs : 2.0, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:
slips[2].wt = 1.0.

Test Case Derivation: The input list of slip surfaces was chosen to be already sorted so that this test case is independent of the sorting. The expected output was derived by manually calculating the weight of the third slip surface according to the specification in the MIS.

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

TC47: test-weighter_last

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 0.8, G : [], X : [], wt : 0), (surf : [], Fs : 1.4, G : [], X : [], wt : 0), (surf : [], Fs : 1.8, G : [], X : [], wt : 0), (surf : [], Fs : 2.0, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:
slips[3].wt = 1.0.

Test Case Derivation: The input list of slip surfaces was chosen to be already sorted so that this test case is independent of the sorting. The expected output was derived by manually calculating the weight of the fourth slip surface according to the specification in the MIS.

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

TC48: test-weighter_last

Type: Automatic

Initial State: New session.

Input: [(surf : [], Fs : 1.0, G : [], X : [], wt : 0), (surf : [], Fs : 1.0, G : [], X : [], wt : 0), (surf : [], Fs : 1.0, G : [], X : [], wt : 0)].

Output: *slips*' such that the following predicate holds true:
 $\forall(i : \mathbb{Z} | i \in [0..2] : \text{slips}[i].wt = \text{NaN})$.

Test Case Derivation: All slip surfaces in the input list were chosen to have equal factors of safety. According to the specification in the MIS, this results in a division by zero in the calculation of the weights. In MatLab, the result of a division by zero is NaN, thus each weight in the output list is expected to be NaN.

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

5.1.5 Slip Slicing Module

The MIS for this module shows that it uses two different slicing algorithms, depending on the value of the boolean *soln.evnslc*. Thus, a test case was selected for each algorithm, to ensure that it slices the given slip surface properly.

TC49: test-slicer_evnslcSize

Type: Automatic

Initial State: $soln.evnslc = 1$.

Input: $[(0, 10), (8, 2), (10, 3), (16, 5)], 6$.

Output: $slip'$ such that the following predicate holds true:
 $|slip'| = 7$.

Test Case Derivation: The input 6 means that the output slip surface should have 6 slices. The number of slices is one less than the number of vertices, so the expected size of the output is 7.

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

TC50: test-slicer_evnslcVals

Type: Automatic

Initial State: $soln.evnslc = 1$.

Input: $[(0, 10), (8, 2), (10, 3), (16, 5)], 6$.

Output: $[(0, 10), (2, 8), (4, 6), (8, 2), (10, 3), (13, 4), (16, 5)]$.

Test Case Derivation: The specification in the MIS for the case where $soln.evnslc$ is true was applied manually to the input coordinates to derive the expected output coordinates.

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

TC51: test-slicer_notEvnslcSize

Type: Automatic

Initial State: $soln.evnslc = 0$.

Input: $[(0, 10), (8, 2), (10, 3), (16, 5)], 6$.

Output: $slip'$ such that the following predicate holds true:
 $|slip'| = 7$.

Test Case Derivation: The input 6 means that the output slip surface should have 6 slices. The number of slices is one less than the number of vertices, so the expected size of the output is 7.

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

TC52: test-slicer_notEvnslcVals

Type: Automatic

Initial State: $soln.evnslc = 0$.

Input: [(0, 10), (8, 2), (10, 3), (16, 5)], 6.

Output: [(0, 10), (4, 6), (8, 2), (9, 2.5), (10, 3), (13, 4), (16, 5)].

Test Case Derivation: The specification in the MIS for the case where *soln.evnslc* is false was applied manually to the input coordinates to derive the expected output coordinates.

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

TC53: test-slicer_notEvnslcVals

Type: Automatic

Initial State: *soln.evnslc* = 0.

Input: [(0, 10), (8, 2), (10, 3)], 5.

Output: *slip'* such that the following predicate holds true:
 $|slip'| = 5$.

Test Case Derivation: In the case where *soln.evnslc* is false, if the desired number of slices is not divisible by the starting number of slices, the desired number of slices should be adjusted to the greatest number divisible by the starting number of slices but less than the input desired number of slices. In this case, that number is 4, so the size of the output should be 5.

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

5.1.6 Morgenstern-Price Calculation Module

Correct calculation of the factor of safety is covered by TC37 and TC39 in the System VnV Plan document. Correct calculation of the interslice normal forces and interslice shear forces are covered by TC41 and TC42, respectively, in the System VnV Plan document. The additional test cases specified here cover special cases not covered by the System VnV Plan: non-converging and spurious solutions. The MIS states that SSP should return a factor of safety of 1000 and empty lists for the interslice forces in both of these cases. For each of these special cases, there are three test case. One covers the factor of safety, one covers the interslice normal force, and one covers the interslice shear force.

5.1.7 Slice Property Calculation Module

The MIS shows that this module is responsible for calculating seven properties for each slice of a slip surface. The calculation of some of these properties is itself dependent on conditions. Test cases have been selected such that each possible calculation is covered by at least one test case.

5.1.8 Output Module

The verification of the output is covered by TC48 in the System VnV Plan document. The delivery of each output is covered by TC43 to TC47 in the System VnV Plan document. This document therefore only adds test cases for items not covered by the System VnV Plan document. The test case here verifies that an output file with the correct name is created.

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]