Module Interface Specification for Slope Stability Analysis Program (SSP)

Henry Frankis and Brooks MacLachlan

December 5, 2018

1 Revision History

Date	Version	Notes
11/12/18	1.0	Initial updates based on template
11/21/18	1.1	Finished updating all of the modules
11/29/18	1.2	Added additional constants to the genetic algorithm module for the adding of vertices to slip surfaces

2 Symbols, Abbreviations and Acronyms

See Section 2 of the Software Requirements Specification (SRS) document, available in the GitHub repository for the project.

Contents

1 Revision History i

2 Symbols, Abbreviations and Acronyms ii

3 Introduction 1

4 Notation 1

5 Numerical Algorithms 2

6 Module Decomposition 2

7 MIS of the Control Module 4
- 7.1 Module . . . 4
- 7.2 Uses . . . 4
- 7.3 Syntax . . . 4
  - 7.3.1 Exported Constants . . . 4
  - 7.3.2 Exported Data Types . . . 4
  - 7.3.3 Exported Access Programs . . . 4
- 7.4 Semantics . . . 4
  - 7.4.1 State Variables . . . 4
  - 7.4.2 Environment Variables . . . 4
  - 7.4.3 Assumptions . . . 4
  - 7.4.4 Access Routine Semantics . . . 5
  - 7.4.5 Local Functions . . . 5

8 MIS of the Input Module 6
- 8.1 Module . . . 6
- 8.2 Uses . . . 6
- 8.3 Syntax . . . 6
  - 8.3.1 Exported Constants . . . 6
  - 8.3.2 Exported Data Types . . . 6
  - 8.3.3 Exported Access Programs . . . 7
- 8.4 Semantics . . . 8
  - 8.4.1 State Variables . . . 8
  - 8.4.2 Environment Variables . . . 8
  - 8.4.3 Assumptions . . . 8
  - 8.4.4 Access Routine Semantics . . . 8
  - 8.4.5 Local Functions . . . 12

9 MIS of the Genetic Algorithm Module 13
9.1 Module . . . 13
9.2 Uses . . . 13
9.2.1 Imported Access Programs . . . 13
9.3 Syntax . . . 13
9.3.1 Exported Constants . . . 13
9.3.2 Exported Data Types . . . 13
9.3.3 Exported Access Programs . . . 13
9.4 Semantics . . . 14
9.4.1 State Variables . . . 14
9.4.2 Environment Variables . . . 14
9.4.3 Assumptions . . . 14
9.4.4 Access Routine Semantics . . . 14
9.4.5 Local Functions . . . 15

10 MIS of the Kinematic Admissibility Module 16
10.1 Module . . . 16
10.2 Uses . . . 16
10.3 Syntax . . . 16
10.3.1 Exported Constants . . . 16
10.3.2 Exported Data Types . . . 16
10.3.3 Exported Access Programs . . . 16
10.4 Semantics . . . 16
10.4.1 State Variables . . . 16
10.4.2 Environment Variables . . . 16
10.4.3 Assumptions . . . 16
10.4.4 Access Routine Semantics . . . 17
10.4.5 Local Functions . . . 17

11 MIS of the Slip Weighting Module 19
11.1 Module . . . 19
11.2 Uses . . . 19
11.3 Syntax . . . 19
11.3.1 Exported Constants . . . 19
11.3.2 Exported Data Types . . . 19
11.3.3 Exported Access Programs . . . 19
11.4 Semantics . . . 19
11.4.1 State Variables . . . 19
11.4.2 Environment Variables . . . 19
11.4.3 Assumptions . . . 19
11.4.4 Access Routine Semantics . . . 19
11.4.5 Local Functions . . . 20

12 MIS of the Slip Slicing Module 21
12.1 Module 21
12.2 Uses 21
12.3 Syntax 21
12.3.1 Exported Constants 21
12.3.2 Exported Data Types 21
12.3.3 Exported Access Programs 21
12.4 Semantics 21
12.4.1 State Variables 21
12.4.2 Environment Variables 21
12.4.3 Assumption 21
12.4.4 Access Routine Semantics 22
12.4.5 Local Functions 22

13 MIS of the Morgenstern-Price Calculation Module 23
13.1 Module 23
13.2 Uses 23
13.3 Syntax 23
13.3.1 Exported Constants 23
13.3.2 Exported Data Types 23
13.3.3 Exported Access Programs 23
13.4 Semantics 23
13.4.1 State Variables 23
13.4.2 Environment Variables 23
13.4.3 Assumptions 23
13.4.4 Access Routine Semantics 24
13.4.5 Local Functions 25

14 MIS of the Slice Property Calculation Module 26
14.1 Module 26
14.2 Uses 26
14.3 Syntax 26
14.3.1 Exported Constants 26
14.3.2 Exported Data Types 26
14.3.3 Exported Access Programs 26
14.4 Semantics 27
14.4.1 State Variables 27
14.4.2 Environment Variables 27
14.4.3 Assumptions 27
14.4.4 Access Routine Semantics 27
14.4.5 Local Functions 29

15 MIS of the Output Module **30**
15.1 Module 30
15.2 Uses 30
15.3 Syntax 30
15.3.1 Exported Constants 30
15.3.2 Exported Data Types 30
15.3.3 Exported Access Programs 30
15.4 Semantics 30
15.4.1 State Variables 30
15.4.2 Environment Variables 30
15.4.3 Assumptions 31
15.4.4 Access Routine Semantics 31
15.4.5 Local Functions 31

16 MIS of the Sequence Data Structure Module **32**
16.1 Module 32
16.2 Uses 32
16.3 Syntax 32
16.3.1 Exported Constants 32
16.3.2 Exported Data Types 32
16.3.3 Exported Access Programs 32
16.4 Semantics 32
16.4.1 State Variables 32
16.4.2 Environment Variables 32
16.4.3 Assumptions 33
16.4.4 Access Routine Semantics 33
16.4.5 Local Functions 33
16.4.6 Considerations 33

17 MIS of the Random Number Generation Module **34**
17.1 Module 34
17.2 Uses 34
17.3 Syntax 34
17.3.1 Exported Constants 34
17.3.2 Exported Data Types 34
17.3.3 Exported Access Programs 34
17.4 Semantics 34
17.4.1 State Variables 34
17.4.2 Environment Variables 34
17.4.3 Assumptions 34
17.4.4 Access Routine Semantics 34
17.4.5 Local Functions 35
17.4.6 Considerations 35

18 MIS of the Plotting Module **36**
18.1 Module 36
18.2 Uses 36
18.3 Syntax 36
18.3.1 Exported Constants 36
18.3.2 Exported Data Types 36
18.3.3 Exported Access Programs 36
18.4 Semantics 36
18.4.1 State Variables 36
18.4.2 Environment Variables 36
18.4.3 Assumptions 36
18.4.4 Access Routine Semantics 37
18.4.5 Local Functions 37
18.4.6 Considerations 37

19 Appendix **39**
19.1 Parameter Tables 39
19.1.1 Layer Parameters 39
19.1.2 Piezometric Parameter 39
19.1.3 Search Range Parameters 39
19.1.4 Solution Parameters 40
19.1.5 Internal Force Parameters 41
19.1.6 Angle Parameters 41

3 Introduction

The following document details the Module Interface Specifications for SSP, a program for determining the critical slip surface and corresponding factor of safety for a given sloped mass of soil. The document is intended to ease understanding of the design of SSP and should be used as a resource for any maintenance of SSP.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at the GitHub repository for the project. [good to point to repo —SS]

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$. The notation for quantifiers is from Gries and Schneider (1993).

The following table summarizes the primitive data types used by SSP.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
boolean	$\mathbb{B}$	a value from the set {true, false}
real	$\mathbb{R}$	any number in $(-\infty, \infty)$
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$

The specification of SSP uses some derived data types: sequences, strings, and tuples. Sequences are ordered lists of elements of the same data type, denoted by brackets enclosing the type of the data elements. If a sequence has fixed dimensions, the notation of the type will include the dimensions in superscript. Strings are sequences of characters. Tuples contain a list of values, potentially of different types, each associated with a field identifier. When a tuple is referenced in this document, a link to an appendix section that specifies the fields of the tuple will be provided. In addition, SSP uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Numerical Algorithms

Morgenstern-Price (Section 13)

The non-linear nature of the systems of equations in the Morgenstern-Price solver algorithm requires that the equations for the factor of safety (IM1), the interslice normal-to-shear force ratio (IM2), and the interslice normal forces (IM3) are solved iteratively, with an initial guess for two of the values, typically the factor of safety and interslice normal-to-shear force ratio.

Genetic Algorithm (Section 9)

SSP uses a genetic algorithm to find the coordinates of the critical slip surface vertices that minimize the factor of safety, as described in IM4. The genetic algorithm generates a set of initial potential slip surfaces, and subsequent generations are created by merging and mutating slip surfaces with low factors of safety from the previous generation. The minimum factor of safety after several generations is assumed to correspond to the critical slip surface.

[This section is not on the template. I've left it in for now because the information does seem useful, but maybe this is not the right place for it? Maybe this should go to an appendix? —BM]

[I like this content too. I particularly like that IM1 to IM4 are invoked. I think it is fine to keep this content here. This information might belong to the "connection between the requirement and design" section of the MG, but that feels a bit premature. Let's leave it here for now. —SS]

6 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Control
	Input
	Genetic Algorithm
	Kinematic Admissibility
	Slip Weighting
	Slip Slicing
	Morgenstern-Price Calculation
	Slice Property Calculation
	Output
Software Decision	Sequence Data Structure
	Random Number Generation
	Plotting

Table 1: Module Hierarchy

7 MIS of the Control Module

7.1 Module

Control

7.2 Uses

Input (Section 8), Output (Section 15), GenAlg (Section 9), Sequence (Section 16)

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Data Types

N/A

7.3.3 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Control	string	-	-

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

The access program is called with a string parameter.

7.4.4 Access Routine Semantics

control(*fname*):

- transition:

  Modifies the state of the Input Module by calling it, calls the Genetic Algorithm Module, and calls the Output Module. The series of calls is shown in more detail below.

load_params(*fname*);

verify_params();

[*Fs*, *crit_slip*, *G*, *X*] = genetic_alg();

verify_output(*Fs*);

output(*Fs*, *crit_slip*, *G*, *X*, *fname*)

[This function doesn't give me enough information. What is fname? Is it a file name? What states are being transitioned? The state of the Input Module? Can you explain the control module by giving the series of calls to the other access programs? Something along the lines of what is done for SWHS? —SS]

[Added the series of calls and modified the text since it was giving the false impression of changing the state of more than just the Input Module. Is this better? —BM]

7.4.5 Local Functions

N/A

[Add a new page between all modules —SS] [Done —BM]

8 MIS of the Input Module

8.1 Module

Input

8.2 Uses

Sequence (Section 16)

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Data Types

coord	=	tuple of (x : $\mathbb{R}$, y : $\mathbb{R}$)
coords	=	[coord]

[The above types don't really have anything specifically to do with the input module. I suggest that you add a new module "Types" that exports your "global" types for this program. This is the same idea as a module for exporting constants that are used throughout a specification. —SS]

[I like that the appendix explains these types. The names are very esoteric. An explanation is helpful. —SS]

[I've been looking in the document for where these types are used and, at least for the ones that I've checked, they don't seem to be used anywhere other than here. If that is the case, why even export types like this? They could just be defined locally, or not defined at all. The main thing seems to be having a way to capture the input parameters and the access programs that provide output seem to do that fine. —SS]

[The motivation behind these types was to clean up the specifications. When I first wrote these, I had the tuples being passed around to all of the functions instead of being state variables, and so the specifications got really cluttered with these long tuples. Since I switched to state variables, maybe I could have removed these types, because you're right that the types aren't used in the rest of the document. —BM]

8.3.3 Exported Access Programs

Name	In	Out	Exceptions
load_params	string	-	fileNotExist, badFileExtension, unexpectedInput
verify_params	-	-	badSlopeGeometry, badEffAngleFriction, badCohesion, badDryUnitWeight, badSatUnitWeight, badPiezGeometry, badWatUnitWeight
strat	-	coords	-
slopeX	-	$[\mathbb{R}]$	-
slopeY	-	$[\mathbb{R}]$	-
phi	-	$\mathbb{R}$	-
coh	-	$\mathbb{R}$	-
gam	-	$\mathbb{R}$	-
gams	-	$\mathbb{R}$	-
piez	-	coords	-
piezX	-	$[\mathbb{R}]$	-
piezY	-	$[\mathbb{R}]$	-
gamw	-	$\mathbb{R}$	-
xExt	-	$[\mathbb{R}]^{1,2}$	-
xEtr	-	$[\mathbb{R}]^{1,2}$	-
yLim	-	$[\mathbb{R}]^{1,2}$	-
ltor	-	$\mathbb{B}$	-
ftype	-	$\mathbb{B}$	-
evnslc	-	$\mathbb{B}$	-
cncvu	-	$\mathbb{B}$	-
obtu	-	$\mathbb{B}$	-

[Is there any value to combining slopeX and slopeY into a list of coordinates? The same question applies for piezX and piezY. It depends on how they are used in the code, but I'm not sure how slopeX is used. I couldn't find it used anywhere else in the specification. —SS]

[What happened here is that I forgot I had access programs for getting these input parameters, so in the rest of the document I called the state variables directly by name instead of using these access programs. Which means that I treated the slope and piez as lists of coordinates. If I combined slopeX and slopeY into "slope", I will have calls like "slope[0].x" and "slope[0].y", whereas if I keep them the same and update my calls in the rest of the document the calls will be "slopeX[0]" and "slopeY[0]". This choice seems like a matter of preference to me, but I think I might go with the first option because it makes it more explicit that the x and y are coming from a common slope value. What do you think? —BM]

8.4 Semantics

8.4.1 State Variables

slope : tuple of (strat : coords, phi : $\mathbb{R}$, coh : $\mathbb{R}$, gam : $\mathbb{R}$, gams : $\mathbb{R}$) (Appendix 19.1.1)
piez : tuple of (piez : coords, gamw : $\mathbb{R}$) (Appendix 19.1.2)
search : tuple of (Xext, Xetr, Ylim : $[\mathbb{R}]^{1,2}$) (Appendix 19.1.3)
soln : tuple of (ltor, ftype, evnslc, cncvu, obtu : $\mathbb{B}$) (Appendix 19.1.4)

8.4.2 Environment Variables

in_file : String

- *in_file* represents a file stored in the file system of the hardware running SSP.

8.4.3 Assumptions

- load_params is called before any of the other access programs.

8.4.4 Access Routine Semantics

load_params(*fname*):

- transition:

  $slope, piez, search, soln := slope', piez', search', soln'$
  where $slope'$, $piez'$, $search'$, and $soln'$ are populated based on the contents of *in_file*.

- exceptions:

  $exc :=$ (*fname* does not exist in file system $\Rightarrow$ fileNotExist
  | $fname[(|fname| - 4)..(|fname| - 1)] =$ ".out" $\Rightarrow$ badFileExtension
  | *in_file* is not formatted correctly $\Rightarrow$ unexpectedInput)

verify_params():

- exceptions:

$$exc := (\neg(\forall(i : \mathbb{Z}|i \in [0..|slope.strat| - 2] : slope.strat[i].x - slope.strat[i+1].x \leq 0)) \Rightarrow \text{badSlopeGeometry}$$
$$|\ \neg(0 < slope.phi < 90) \Rightarrow \text{badEffAngleFriction}$$
$$|\ \neg(0 < slope.coh) \Rightarrow \text{badCohesion}$$
$$|\ \neg(0 < slope.gam) \Rightarrow \text{badDryUnitWeight}$$
$$|\ \neg(0 < slope.gams) \Rightarrow \text{badSatUnitWeight}$$
$$|\ \neg(\forall(i : \mathbb{Z}|i \in [0..|piez.piez| - 2] : piez.piez[i].x - piez.piez[i+1].x \leq 0))$$
$$\vee\ piez.piez[0].x \neq slope.strat[0].x$$
$$\vee\ piez.piez[|piez.piez| - 1].x \neq slope.strat[|slope.strat| - 1].x$$
$$\Rightarrow \text{badPiezGeometry}$$
$$|\ (slope.strat[|slope.strat| - 1].x > slope.strat[0].x \Rightarrow \forall(i : \mathbb{Z}|i \in [0..1] :$$
$$search.Xext[i] > slope.strat[|slope.strat| - 1].x$$
$$\vee\ search.Xext[i] < slope.strat[0].x$$
$$\vee\ search.Xetr[i] > slope.strat[|slope.strat| - 1].x$$
$$\vee\ search.Xetr[i] < slope.strat[0].x)$$
$$|\ slope.strat[|slope.strat| - 1].x < slope.strat[0].x \Rightarrow \forall(i : \mathbb{Z}|i \in [0..1] :$$
$$search.Xext[i] < slope.strat[|slope.strat| - 1].x$$
$$\vee\ search.Xext[i] > slope.strat[0].x$$
$$\vee\ search.Xetr[i] < slope.strat[|slope.strat| - 1].x$$
$$\vee\ search.Xetr[i] > slope.strat[0].x))$$
$$\Rightarrow \text{badSlipGuess})$$

strat():

- output:

$$out := slope.strat$$

slopeX():

- output:

$$out := slope.strat[0].x || slope.strat[1].x || \ldots || slope.strat[|slope.strat| - 1].x$$

slopeY():

- output:

$$out := slope.strat[0].y||slope.strat[1].y|| \ldots ||slope.strat[|slope.strat|-1].y$$

phi():

- output:

$$out := slope.phi$$

coh():

- output:

$$out := slope.coh$$

gam():

- output:

$$out := slope.gam$$

gams():

- output:

$$out := slope.gams$$

piez():

- output:

$$out := piez.piez$$

piezX():

- output:

$$out := piez.piez[0].x||piez.piez[1].x|| \ldots ||piez.piez[|piez.piez|-1].x$$

piezY():

- output:

$$out := piez.piez[0].y||piez.piez[1].y|| \\ \dots||piez.piez[|piez.piez|-1].y$$

gamw():

- output:

$$out := piez.gamw$$

xExt():

- output:

$$out := search.Xext$$

xEtr():

- output:

$$out := search.Xetr$$

yLim():

- output:

$$out := search.Ylim$$

ltor():

- output:

$$out := soln.ltor$$

ftype():

- output:

$$out := soln.ftype$$

evnslc():

- output:

$$out := soln.evnslc$$

cncvu():

- output:

$$out := soln.cncvu$$

obtu():

- output:

$$out := soln.obtu$$

8.4.5 Local Functions

N/A

[I'm not suggesting we change the design, but I wonder if there is a different design that is more elegant? I don't like that we are passing around so many parameters and lists. At some point in the future, not as part of 741, I'd like to discuss a more OO style design. What if we had an object for the soil mass and another for the slip surface? The constructor could populate the necessary information and then the object could be queried for the services that matter. The slip surface could be asked to return its own Fs, rather than use another module to calculate it. This redesign probably won't go anywhere, but we should discuss it at some point. We could just sketch out the access programs and state variables, to see if there is something viable. —SS]

[We can discuss that, certainly —BM]

9 MIS of the Genetic Algorithm Module

9.1 Module

GenAlg

9.2 Uses

9.2.1 Imported Access Programs

Input (Section 8), MorgPriceSolver (Section 13), Slicer (Section 12), KinAdm (Section 10), SlipWeighter (Section 11), Sequence (Section 16), Rand (Section 17)

9.3 Syntax

9.3.1 Exported Constants

MIN_GENS = 100
NUM_SLIPS = 20
REL_DIFF = 0.00005
INIT_NUM_VERTICES = 4
NUM_ADDS = 2

9.3.2 Exported Data Types

slip	=	tuple of (surf : coords, Fs : $\mathbb{R}$, G : coords, X : coords, wt : $\mathbb{R}$)
slips	=	[slip]

[If you do introduce a module whose sole purpose is to export types, these types could go in that module. —SS]

[As mentioned in an earlier comment, I will likely remove lots of the types I have defined. Types like slip, slips, coord, and coords, however, are used in many modules. Would it still make sense to write a separate module just for these types? I think yes. (Though I should mention that in the actual implementation, none of these types exist. Slips are simply tuples and coords are arrays, though I found it easier to create types for these for the purposes of specification) —BM]

9.3.3 Exported Access Programs

Name	In	Out	Exceptions
genetic_alg	-	$\mathbb{R}$, coords, coords, coords	-

9.4 Semantics

9.4.1 State Variables

N/A

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

N/A

9.4.4 Access Routine Semantics

genetic_alg():

- output:

  $out :=$ weighter($slip\_surfs$)[0].$surf$, weighter($slip\_surfs$)[0].Fs, weighter($slip\_surfs$)[0].G, and weighter($slip\_surfs$)[0].X, where $slip\_surfs$, of type slips, is developed by:

  * using rand to randomly generate coordinates for NUM_SLIPS potential slip surfaces, where the entry and exit x-coordinate for each slip surface are computed according to *generate_slips(xEtr)* and *generate_slips(xExt)*. Corresponding y-coordinates are determined by interpolating on the slope geometry. The total number of coordinates for each slip surface is INIT_NUM_VERTICES.
  * using kinAdm to verify that the geometry of each potential slip surface is physically realizable. If any are not, new slip surfaces are randomly generated until NUM_SLIPS valid slip surfaces have been generated,
  * using slicer to redefine each slip surface's coordinates based on the desired number of slices
  * using morg_price to determine the Fs, G, and X fields of each slip surface
  * using weighter to determine the wt field of each slip surface
  * using rand to generate a new pool of NUM_SLIPS slip surfaces by applying crossovers and mutations to the previous generation, with the more lowly-weighted members having a greater likelihood of contributing to the subsequent generations
  * applying kinAdm, slicer, morg_price, and weighter to the new generation

* repeating the above two steps, but every time $\frac{\text{MIN\_GENS}}{\text{NUM\_ADDS}+1}$ generations have occured, vertices are added to the pool of slip surfaces halfway between each of the existing vertices, so that the new slip surfaces each have (INIT_NUM_VERTICES$*$ 2) $-$ 1 vertices.
* repeating until at least MIN_GENS have occured and the relative difference between subsequent generations is less than REL_DIFF.

9.4.5 Local Functions

generate_slips($Xrange$) : $[\mathbb{R}] \rightarrow \mathbb{R}$
generate_slips($Xrange$) $= (Xrange[0] + rand() * (Xrange[1] - Xrange[0]))$

10 MIS of the Kinematic Admissibility Module

10.1 Module

KinAdm

10.2 Uses

Input (Section 8), Sequence (Section 16)

10.3 Syntax

10.3.1 Exported Constants

N/A

10.3.2 Exported Data Types

N/A

10.3.3 Exported Access Programs

Name	In	Out	Exceptions
kinAdm	slip	$\mathbb{B}$	-

10.4 Semantics

10.4.1 State Variables

N/A

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

- The *surf* field is populated for every member of the input sequence of slip data.

10.4.4 Access Routine Semantics

kinAdm(*slip_surf*):

- output:

$$out := (\ \neg(\forall(i:\mathbb{Z}|i \in [0..|slip\_surf.surf| - 2] : slip\_surf.surf[i].x - slip\_surf.surf[i+1].x \leq 0))$$
$$\vee \neg is\_on\_slope(slip\_surf.surf[0])$$
$$\vee \neg is\_on\_slope(slip\_surf.surf[|slip\_surf.surf - 1|])$$
$$\vee \neg is\_in\_slope(slip\_surf.surf)$$
$$\vee (cncvu() \wedge \neg(is\_concave\_up(slip\_surf.surf)))$$
$$\vee (obtu() \wedge \neg(has\_no\_sharp\_angles(slip\_surf.surf)))$$
$$\Rightarrow false$$
$$|true \Rightarrow true)$$

[Not sure if I'm allowed to use "else" here but don't know how else to express the "else" case succinctly —BM]

[With the conditional rule, you would just replace your "else" with "True." The H&S syntax works by following the first rule where the condition evaluates to true. A true at the end acts as a catch all. —SS]

[Makes sense, thanks. Fixed —BM]

10.4.5 Local Functions

linSlope(*point1*, *point2*) : coord $\times$ coord $\rightarrow \mathbb{R}$
linSlope(*point1*, *point2*) $= \frac{point2.y - point1.y}{point2.x - point1.x}$

is_on_slope(*point*) : coord $\rightarrow \mathbb{B}$
is_on_slope(*point*) $= (\exists(i:\mathbb{Z}|i \in [0..|slope.strat| - 1] : point = slope.strat[i]))$
$\vee (\exists(i:\mathbb{Z}|i \in [0..|slope.strat| - 2] : point.y = linSlope(slope.strat[i], slope.strat[i+1]) * point.x + \frac{slope.strat[i].y}{linSlope(slope.strat[i], slope.strat[i+1]) * slope.strat[i].x}))$

is_in_slope(*surf*) : coords $\rightarrow \mathbb{B}$
is_in_slope(*surf*) $= (\forall(i:\mathbb{Z}|i \in [1..|surf| - 2] : (\forall(j:\mathbb{Z}|j \in [0..|slope.strat| - 2] \wedge slope.strat[j].x \leq surf[i].x < slope.strat[j+1].x : surf[i].y < (slope.strat[j].y + (surf[i].x - slope.strat[j].x) * linSlope(slope.strat[j], slope.strat[j+1]))))))$

is_concave_up(*surf*) : coords $\rightarrow \mathbb{B}$
is_concave_up(*surf*) $= (\forall(i:\mathbb{Z}|i \in [0..|surf| - 3] : linSlope(surf[i+1], surf[i+2]) \geq linSlope(surf[i], surf[i+1]))$

distance(*point1*, *point2*) : coord $\times$ coord $\rightarrow \mathbb{R}$
distance(*point1*, *point2*) $= \sqrt{(point1.x - point2.x)^2 + (point1.y - point2.y)^2}$

$\text{has\_no\_sharp\_angles}(\mathit{surf}) : \text{coords} \rightarrow \mathbb{B}$
$\text{has\_no\_sharp\_angles}(\mathit{surf}) = (\forall (i : \mathbb{Z} | i \in [0..|\mathit{surf}| - 3] :$
$\arccos \frac{(distance(surf[i], surf[i+1]))^2 + (distance(surf[i+1], surf[i+2]))^2 - (distance(surf[i], surf[i+2]))^2}{2 * distance(surf[i], surf[i+1]) * distance(surf[i+1], surf[i+2])} \geq 1.9199))$

11 MIS of the Slip Weighting Module

11.1 Module

SlipWeighter

11.2 Uses

Sequence (Section 16)

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 Exported Data Types

N/A

11.3.3 Exported Access Programs

Name	In	Out	Exceptions
weighter	slips	slips	-

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

N/A

11.4.3 Assumptions

- The Fs field is populated for every member of the input sequence of slip data.

11.4.4 Access Routine Semantics

weighter($slip\_surfs$):

- output:

$$out := slip\_surfs' \text{ such that } slip\_surfs' = assign\_weights(sort\_Fs(slip\_surfs))$$

11.4.5 Local Functions

sort_Fs($unsorted$) : slips $\rightarrow$ slips sort_Fs($unsorted$) $=$ $sorted$ such that
$\forall(a : slip | a \in unsorted : \exists(b : slip | b \in sorted : b = a \wedge count(a, A) = count(b, B))) \wedge \forall(i : \mathbb{Z} | i \in [0..|unsorted| - 1] : sorted[i].Fs \leq sorted[i+1].Fs)$

count(a, A) : slip $\times$ slips $\rightarrow \mathbb{Z}$
count(a, A) $= +(x : slip | x \in A \wedge x = a : 1)$

#The weight assigned to a given slip is the difference between that slip's factor of safety and the largest factor of safety, divided by the sum of all of these differences and cumulatively added, so that the slip with the lowest factor of safety has the lowest weight and the slip with the highest factor of safety has a weight of 1.
assign_weights(s) : slips $\rightarrow$ slips
assign_weights(s) $= s'$ such that
$s'[0].wt = \frac{s[0].Fs - s[|s|-1].Fs}{+(j:\mathbb{Z}|j\in[0..|s|-1]:s[j].Fs - s[|s|-1].Fs)}$ and
$\forall(i : \mathbb{Z} | i \in [1..|s| - 1] : s'[i].wt = s'[i-1].wt + \frac{s[i].Fs - s[|s|-1].Fs}{+(j:\mathbb{Z}|j\in[0..|s|-1]:s[j].Fs - s[|s|-1].Fs)})$

[Could you say in a comment what is happening with assign_weights. It is not obvious from reading the definition. The equality at the top of the fraction looks like it evaluates to a Boolean, but then you are diving by an integer? Even if the formula is correct, you should explain it, since whatever is going on, it is complicated. —SS]

[The boolean divided by an integer was a typo. Fixed that, and also added a comment with some explanation. —BM]

12 MIS of the Slip Slicing Module

12.1 Module

Slicer

12.2 Uses

Sequence (Section 16)

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Data Types

N/A

12.3.3 Exported Access Programs

Name	In	Out	Exceptions
slicer	coords, $\mathbb{Z}$	coords	-

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumption

- The integer input to *slicer* is greater than the size of the slip input to *slicer*.

12.4.4 Access Routine Semantics

#Based on the value of evnslc, there are two potential slicing algorithms that could be used. If evnslc is true, the slip surface's largest segment will be sliced in half, and then the new largest segment will be sliced in half, and so on until the desired number of slices is reached. If evnslc is false, each slice will be divided into an equal number of subslices such that the resulting number of slices is as close as possible to the desired number without going over.

slicer($slip\_surf$, num_slices):

- output:

$$out := (soln.evnslc \Rightarrow slip\_surf' \text{ obtained by repeatedly applying } slip\_surf[\text{large\_segment}(slip\_surf$$
$$||\ \text{midpoint}(slip\_surf[\text{large\_segment}(slip\_surf)], slip\_surf[\text{large\_segment}(slip\_surf)+1])$$
$$||\ slip\_surf[\text{large\_segment}(slip\_surf)+1] \text{ until } |slip\_surf'| = num\_slices$$
$$|\neg soln.evnslc \Rightarrow slip\_surf' \text{ such that } \forall(i : \mathbb{Z} | i \in [0..|slip\_surf| - 2] : slip\_surf'[i * round\_down(\tfrac{num\_slices}{|slip\_surf|-1})..(i+1) * round\_down(\tfrac{num\_slices}{|slip\_surf|-1})] = subslice(round\_down(\tfrac{num\_slices}{|slip\_surf|-1}), slip\_surf[i], slip\_surf[i+1]))$$

[The specification of the slicer access program is confusing to me. As mentioned previously for another complex spec, could you please say in words, in a comment, what is going on here? —SS]

[Added a comment above. —BM]

12.4.5 Local Functions

large_segment($surf$) : coords $\rightarrow \mathbb{Z}$
large_segment($surf$) = $index$ such that
$\forall(i : \mathbb{Z} | i \in [0..|surf| - 2] : surf[index + 1] - surf[index] \geq surf[i+1] - surf[i])$

midpoint($point1$, $point2$) : coord $\times$ coord $\rightarrow$ coord
midpoint($point1$, $point2$) = $< \frac{point1.x + point2.x}{2}, \frac{point1.y + point2.y}{2} >$

round_down(num) : $\mathbb{R} \rightarrow \mathbb{Z}$
round_down(num) = $rounded\_num$ such that
$\forall(i : \mathbb{Z} | i \leq num < i + 1 : rounded\_num = i)$

subslice(n, $point1$, $point2$) : $\mathbb{Z} \times$ coord $\times$ coord $\rightarrow$ coords
subslice(n, $point1$, $point2$) = $subslices$ such that
$\forall(i : \mathbb{Z} | i \in [0..n] : subslices[i].x = point1.x + \frac{i}{n} * (point2.x - point1.x) \wedge subslices[i].y = point1.y + \frac{i}{n} * (point2.y - point1.y))$

13 MIS of the Morgenstern-Price Calculation Module

13.1 Module

MorgPriceSolver

13.2 Uses

Input (Section 8), PropertyCalc (Section 14), Sequence (Section 16)

13.3 Syntax

13.3.1 Exported Constants

MAX_DIFF = 0.000001
MAX_ITER = 20
MIN_FS = 0.5

13.3.2 Exported Data Types

N/A

13.3.3 Exported Access Programs

Name	In	Out	Exceptions
morg_price	slip	slip	-

13.4 Semantics

13.4.1 State Variables

N/A

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

N/A

13.4.4 Access Routine Semantics

morg_price(*slip_surf*):

- output:

  out := *slip_surf′* where *slip_surf′*.*Fs*, *slip_surf′*.*G*, and *slip_surf′*.*X* satisfy the following system of equations, taken from the SRS document. The equations are presented with the symbols from the SRS document for readability, though the symbols F_{S}, G, and X correspond to *slip_surf′*.*Fs*, *slip_surf′*.*G*, and *slip_surf′*.*X*, respectively.

$$\text{(IM1)}:\ F_{\text{S}} = \frac{\sum_{i=1}^{n-1}\left[R_i \prod_{c=i}^{n-1} \Psi_c\right] + R_n}{\sum_{i=1}^{n-1}\left[T_i \prod_{c=i}^{n-1} \Psi_c\right] + T_n}$$

$$\text{(IM2)}:\ C_{\text{num},i} = \begin{cases} b_1\left[G_1 + H_1\right]\tan\left(\alpha_1\right) & i = 1 \\ b_i\left[\left(G_i + G_{i-1}\right) + \left(H_i + H_{i-1}\right)\right]\tan\left(\alpha_i\right) \\ \quad + h_i\left(-2\,U_{\text{t},i}\sin\left(\beta_i\right)\right) & 2 \leq i \leq n-1 \\ b_n\left[G_{n-1} + H_{n-1}\right]\tan\left(\alpha_{n-1}\right) & i = n \end{cases}$$

$$C_{\text{den},i} = \begin{cases} b_1 G_1 f_1 & i = 1 \\ b_i\left(f_i G_i + f_{i-1} G_{i-1}\right) & 2 \leq i \leq n-1 \\ b_n G_{n-1} f_{n-1} & i = n \end{cases}$$

$$\lambda = \frac{\sum_{i=1}^{n} C_{\text{num},i}}{\sum_{i=1}^{n} C_{\text{den},i}}$$

$$\text{(IM3)}:\ G_i = \begin{cases} \frac{(F_{\text{S}})T_1 - R_1}{\Phi_i} & i = 1 \\ \frac{\Psi_{i-1}\cdot G_{i-1} + (F_{\text{S}})\cdot T_i - R_i}{\Phi_i} & 2 \leq i \leq n-1 \\ 0 & i = 0 \vee i = n \end{cases}$$

$$\text{(GD8)}:\ X = \lambda \cdot f \cdot G$$

  The solution method is to start with initial guesses $F_{\text{S}} = 1$ and $\lambda = 0$ and use them to compute F_{S} using IM1 and G using IM3, then use these values to compute a new guess for λ using IM2. This iteration continues until the absolute difference between F_{S} in the current iteration and in the previous iteration is less than MAX_DIFF, or until the absolute difference between λ in the current iteration and in the previous iteration is less than MAX_DIFF. When this occurs, X is computed using GD8, and $slip\_surf'.Fs, slip\_surf'.G, slip\_surf'.X := F_{\text{S}}, G, X$. If MAX_ITER iterations occur, the solution is considered to be non-converging. If the solution converges but $F_{\text{S}} <$ MIN_FS, the solution is considered to be spurious. In either case, $slip\_surf'.Fs, slip\_surf'.G, slip\_surf'.X := 1000, [], []$.

[Nice to see excerpts from the SRS. That is an encouraging sign. The mix of equations and description seems like a good spec to me. —SS]

13.4.5 Local Functions

N/A

14 MIS of the Slice Property Calculation Module

14.1 Module

PropertyCalc

14.2 Uses

Input (Section 8), Sequence (Section 16)

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Data Types

N/A

14.3.3 Exported Access Programs

Name	In	Out	Exceptions
prop_calc	slip	-	-
ub	-	$[\mathbb{R}]$	-
ut	-	$[\mathbb{R}]$	-
w	-	$[\mathbb{R}]$	-
h	-	$[\mathbb{R}]$	-
alpha	-	$[\mathbb{R}]$	-
beta	-	$[\mathbb{R}]$	-
hts	-	$[\mathbb{R}]$	-

14.4 Semantics

14.4.1 State Variables

force : tuple of (Ub, Ut, W, H : $[\mathbb{R}]$) (Appendix 19.1.5)
angles : tuple of (alpha, beta : $[\mathbb{R}]$) (Appendix 19.1.6)
heights : $[\mathbb{R}]$

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

- prop_calc is called before any of the other access programs.

14.4.4 Access Routine Semantics

prop_calc(*slip_surf*):

- transition:

  The equations used below contain symbols from the SRS document for this project for the sake of brevity. The SRS should be consulted for the definitions of these symbols.

  $$\mathit{force}, \mathit{angles}, \mathit{heights} := \mathit{force}', \mathit{angles}', \mathit{soil}', \mathit{heights}',$$

  where

  $$\begin{aligned}
  &\forall (i : \mathbb{Z} | i \in [1..|\mathit{slip_surf}| - 1] : \\
  &\mathit{force}.Ub[i] = 0.5(U_{\mathrm{b},i,1} + U_{\mathrm{b},i,2}) \\
  &\wedge \mathit{force}.Ut[i] = 0.5(U_{\mathrm{t},i,1} + U_{\mathrm{t},i,2}) \\
  &\wedge \mathit{force}.W[i] = 0.5(W_{i,1} + W_{i,2}) \\
  &\wedge \mathit{force}.H[i] = \begin{cases} \frac{\left[y_{\mathrm{slope},i} - y_{\mathrm{slip},i}\right]^2}{2}\gamma_{\mathrm{w}} + \left[y_{\mathrm{wt},i} - y_{\mathrm{slope},i}\right]^2 \gamma_{\mathrm{w}} & y_{\mathrm{wt},i} \geq y_{\mathrm{slope},i} \\ \frac{\left[y_{\mathrm{wt},i} - y_{\mathrm{slip},i}\right]^2}{2}\gamma_{\mathrm{w}} & y_{\mathrm{slope},i} > y_{\mathrm{wt},i} > y_{\mathrm{slip},i} \\ 0 & y_{\mathrm{wt},i} \leq y_{\mathrm{slip},i} \end{cases} \\
  &\wedge \mathit{angles}.\mathit{alpha}[i] = \arctan\left(\frac{y_{\mathrm{slip},i} - y_{\mathrm{slip},i-1}}{x_{\mathrm{slip},i} - x_{\mathrm{slip},i-1}}\right) \\
  &\wedge \mathit{angles}.\mathit{beta}[i] = \arctan\left(\frac{y_{\mathrm{slope},i} - y_{\mathrm{slope},i-1}}{x_{\mathrm{slope},i} - x_{\mathrm{slope},i-1}}\right) \\
  &\wedge \mathit{heights}[i] = 0.5 * ((y_{\mathrm{slope},i} - y_{\mathrm{slip},i}) + (y_{\mathrm{slope},i-1} - y_{\mathrm{slip},i-1}))),
  \end{aligned}$$

  where

  $$U_{\mathrm{b},i,1} = \ell_{\mathrm{b},i} \begin{cases} (y_{\mathrm{wt},i} - y_{\mathrm{slip},i})\,\gamma_{\mathrm{w}} & y_{\mathrm{wt},i} > y_{\mathrm{slip},i} \\ 0 & y_{\mathrm{wt},i} \leq y_{\mathrm{slip},i} \end{cases},$$

  $$U_{\mathrm{b},i,2} = \ell_{\mathrm{b},i} \begin{cases} (y_{\mathrm{wt},i-1} - y_{\mathrm{slip},i-1})\,\gamma_{\mathrm{w}} & y_{\mathrm{wt},i-1} > y_{\mathrm{slip},i-1} \\ 0 & y_{\mathrm{wt},i-1} \leq y_{\mathrm{slip},i-1} \end{cases},$$

  $$U_{\mathrm{t},i,1} = \ell_{\mathrm{s},i} \begin{cases} (y_{\mathrm{wt},i} - y_{\mathrm{slope},i})\,\gamma_{\mathrm{w}} & y_{\mathrm{wt},i} > y_{\mathrm{slope},i} \\ 0 & y_{\mathrm{wt},i} \leq y_{\mathrm{slope},i} \end{cases},$$

$$U_{\mathrm{t},i,2} = \ell_{\mathrm{s},i} \begin{cases} \left(y_{\mathrm{wt},i-1} - y_{\mathrm{slope},i-1}\right)\gamma_{\mathrm{w}} & y_{\mathrm{wt},i-1} > y_{\mathrm{slope},i-1} \\ 0 & y_{\mathrm{wt},i-1} \leq y_{\mathrm{slope},i-1} \end{cases},$$

$$W_{i,1} = b_i \begin{cases} \left(y_{\mathrm{slope},i} - y_{\mathrm{slip},i}\right)\gamma_{\mathrm{Sat}} & y_{\mathrm{wt},i} \geq y_{\mathrm{slope},i} \\ \left(y_{\mathrm{slope},i} - y_{\mathrm{wt},i}\right)\gamma + \left(y_{\mathrm{wt},i} - y_{\mathrm{slip},i}\right)\gamma_{\mathrm{Sat}} & y_{\mathrm{slope},i} > y_{\mathrm{wt},i} > y_{\mathrm{slip},i} \\ \left(y_{\mathrm{slope},i} - y_{\mathrm{slip},i}\right)\gamma & y_{\mathrm{wt},i} \leq y_{\mathrm{slip},i} \end{cases},$$

$$W_{i,2} = b_i \begin{cases} \left(y_{\mathrm{slope},i-1} - y_{\mathrm{slip},i-1}\right)\gamma_{\mathrm{Sat}} & y_{\mathrm{wt},i-1} \geq y_{\mathrm{slope},i-1} \\ \begin{pmatrix} \left(y_{\mathrm{slope},i-1} - y_{\mathrm{wt},i-1}\right)\gamma \\ + \left(y_{\mathrm{wt},i-1} - y_{\mathrm{slip},i-1}\right)\gamma_{\mathrm{Sat}} \end{pmatrix} & y_{\mathrm{slope},i-1} > y_{\mathrm{wt},i-1} > y_{\mathrm{slip},i-1} \\ \left(y_{\mathrm{slope},i-1} - y_{\mathrm{slip},i-1}\right)\gamma & y_{\mathrm{wt},i-1} \leq y_{\mathrm{slip},i-1} \end{cases}$$

ub():

- output:

$$out := force.Ub$$

ut():

- output:

$$out := force.Ut$$

w():

- output:

$$out := force.W$$

h():

- output:

$$out := force.H$$

alpha():

- output:

$$out := angles.alpha$$

beta():

- output:

$$out := angles.beta$$

hts():

- output:

$$out := heights$$

14.4.5 Local Functions

N/A

15 MIS of the Output Module

15.1 Module

Output

15.2 Uses

Sequence (Section 16), Plot (Section 18)

15.3 Syntax

15.3.1 Exported Constants

N/A

15.3.2 Exported Data Types

N/A

15.3.3 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
verify_output	$\mathbb{R}$	-	negativeFS
output	$\mathbb{R}$, coords, coords, coords, string	-	-

15.4 Semantics

15.4.1 State Variables

N/A

15.4.2 Environment Variables

out_file : String

- *out_file* represents a file stored in the file system of the hardware running SSP.

screen : $[\mathbb{Z}]$

- *screen* represents the colour values for each pixel on the screen of the hardware running SSP.

15.4.3 Assumptions

N/A

15.4.4 Access Routine Semantics

verify_output(Fs):

- exceptions:

$$exc := Fs < 0 \Rightarrow \text{negativeFS}$$

output(Fs, $crit\_slip$, G, X, $fname$):

- transition:

  out_file is created at path $fname$ || ".out". The outputs of xEtr(), xExt(), yLim(), ftype(), Fs, $crit\_slip$, G, and X are written to out_file. $screen$ is modified to display the outputs of plot($crit\_slip.x$, $crit\_slip.y$), plot($G.x$, $G.y$), and plot($X.x$, $X.y$).

15.4.5 Local Functions

N/A

16 MIS of the Sequence Data Structure Module

[Nice that you included the specification of this module. You could have simply said, “as implemented by Matlab,” but I like this. Matlab has other operations on sequences that aren’t specified (like assignment), but we shouldn’t invest too much time in this spec, so you don’t need to add this. —SS]

16.1 Module

Sequence

16.2 Uses

N/A

16.3 Syntax

16.3.1 Exported Constants

N/A

16.3.2 Exported Data Types

[T] = sequence of T, where T is any type

16.3.3 Exported Access Programs

Name	In	Out	Exceptions
[_]	Any number of values of type T	[T]	-
_(_)	[T], $\mathbb{Z}$	T	
_(_:_)	[T], $\mathbb{Z}$, $\mathbb{Z}$	[T]	-

16.4 Semantics

16.4.1 State Variables

N/A

16.4.2 Environment Variables

N/A

16.4.3 Assumptions

N/A

16.4.4 Access Routine Semantics

[_](Any number of values):

- output:

  $out :=$ A sequence containing the arguments passed to the function.

_(_)(*list*, *int*):

- output:

  $out := list[int]$

_(_:_)(*list*, *int1*, *int2*):

- output:

  $out := list[int1..int2]$

16.4.5 Local Functions

N/A

16.4.6 Considerations

This module is the sequence data type and operations on sequences implemented by Matlab.

17 MIS of the Random Number Generation Module

17.1 Module

Rand

17.2 Uses

N/A

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Data Types

N/A

17.3.3 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
rand	-	$\mathbb{R}$	-

17.4 Semantics

17.4.1 State Variables

N/A

17.4.2 Environment Variables

N/A

17.4.3 Assumptions

N/A

17.4.4 Access Routine Semantics

rand():

- output:

  $out :=$ A random number in the interval (0,1).

17.4.5 Local Functions

N/A

17.4.6 Considerations

This module is the rand function implemented by Matlab.

18 MIS of the Plotting Module

18.1 Module

Plot

18.2 Uses

N/A

18.3 Syntax

18.3.1 Exported Constants

N/A

18.3.2 Exported Data Types

N/A

18.3.3 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
plot	$[\mathbb{R}]$, $[\mathbb{R}]$	-	-

18.4 Semantics

18.4.1 State Variables

N/A

18.4.2 Environment Variables

$screen : [\mathbb{Z}]$

- *screen* represents the colour values for each pixel on the screen of the hardware running SSP.

18.4.3 Assumptions

N/A

18.4.4 Access Routine Semantics

plot(x, y):

- transition:

  Modifies *screen* to display a plot with x on the horizontal axis and y on the vertical axis.

18.4.5 Local Functions

N/A

18.4.6 Considerations

This module is the plot function implemented by Matlab.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

David Gries and Fred B. Schneider. *A logical approach to discrete math*. Springer-Verlag Inc., New York, 1993.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

19 Appendix

19.1 Parameter Tables

19.1.1 Layer Parameters

The elements of the structure which describes the mass of soil on which slope stability analysis is to be performed are explained in the table below.

Parameter	Description
strat : coords	Coordinates describing the vertices of the slope of soil.
phi : $\mathbb{R}$	The effective angle of friction of the soil.
coh : $\mathbb{R}$	The effective cohesion of the soil.
gam : $\mathbb{R}$	The dry unit weight of the soil.
gams : $\mathbb{R}$	The saturated unit weight of the soil.

19.1.2 Piezometric Parameter

The elements in the structure which describes the water table are explained in the table below.

Parameter	Description
piez : *coords*	Coordinates describing the vertices of the water table. If there is no water table than *piez* is an empty array.
gamw : $\mathbb{R}$	The unit weight of water.

19.1.3 Search Range Parameters

The elements in the structure which holds parameters relating to the range of coordinates between which the critical slip surface may exist are described in the table below.

Parameter	Description
Xext : $[\mathbb{R}]^{1,2}$	The range of x-ordinates between which the exit point of the critical slip surface may exist. Exit refers to the point of the slip at lower elevation toward which the mass of soil will move during failure.
Xetr : $[\mathbb{R}]^{1,2}$	The range of x-ordinates between which the entry point of the critical slip surface may exist. Entry refers to the point of the slip at higher elevation away from which the mass of soil will move during failure.
Ylim : $[\mathbb{R}]^{1,2}$	The range of y-ordinates between which the critical slip surface may exist. The larger value should be greater than the max y-ordinate of the slope. The smaller value is the lowest elevation to which the critical slip surface may descend.

19.1.4 Solution Parameters

The elements in the structure which holds parameters relating to the solution method are described in the table below.

Parameter	Description
ltor : $\mathbb{B}$	Direction the slope is expected to experience failure in. If true then the side of the slope with a greater x-ordinate value is at a lower elevation. If false then the side of the slope with a greater x-ordinate is at a higher elevation.
ftype : $\mathbb{B}$	Function to use for interslice normal/shear force ratio variation function. If true then the function is a constant (Spencer's method). If false then the function is a half-sine (standard Morgenstern-Price method).
evnslc : $\mathbb{B}$	Method for slicing a slip surface prior to analysis. If true then slice slip surface into equal x-ordinate widths. If false then slice distance between vertices into even number of slices.
cncvu : $\mathbb{B}$	Concave slip surface admissibility criterion. If true then an admissible slip surface must be concave upwards towards the surface. If false then an admissible slip surface does not need to pass this criterion.
obtu : $\mathbb{B}$	Angle slip surface admissibility criterion. If true then an admissible slip surface must have all interior angles greater than a set limit. If false then an admissible slip surface does not need to pass this criterion.

19.1.5 Internal Force Parameters

The elements in the structure which holds parameters relating to the forces acting on a slice, and water in the slope acting on itself, are described in the table below. n refers to the number of slices composing the slip surface under evaluation, and is defined by the Slicer module (section 12).

Parameter	Description
Ub : $[\mathbb{R}]^{1,n}$	Sequence of the force acting on the basal surface of a slice as a result of pore water pressure within the slice. From DD2 of the SRS.
Ut : $[\mathbb{R}]^{1,n}$	Sequence of the force acting on the upper surface of a slice as a result of pore water pressure from standing water on the surface. From DD3 of the SRS.
W : $[\mathbb{R}]^{1,n}$	Sequence of the downward force acting on the slice caused by the mass of the slice and the force of gravity. From DD1 of the SRS.
H : $[\mathbb{R}]^{1,n-1}$	Sequence of the force acting into the interslice surfaces as a result of pore water pressure within the adjacent slices. From DD4 of the SRS.

19.1.6 Angle Parameters

The elements in the structure which holds parameters relating to the angles of the slice surfaces are described in the table below. n refers to the number of slices composing the slip surface under evaluation, and is defined by the Slicer module (section 12).

Parameter	Description
alpha : $[\mathbb{R}]^{1,n}$	Sequence of the angle that the basal surface of the slice makes with the horizontal. From DD5 of the SRS.
beta : $[\mathbb{R}]^{1,n}$	Sequence of the angle that the upper surface of the slice makes with the horizontal. From DD6 of the SRS.