Appendix A

Verification

The verification of a computer program used for engineering design is essential because the results generated by the program are used for the design of structures where the welfare of the public is at stake. This means that errors in programs that are used for engineering design are much more serious than errors in typical software (e.g. word processors). This is certainly true for rockfall programs. When a rockfall program is used, it is usually because there is a risk of rocks falling on vehicles, buildings, or people.

There are many potential sources of errors when creating a computer program for engineering design. The equations that are used in the program can be theoretically flawed. The computer can introduce error because of the way it handles numbers. The equations can be theoretically correct but errors were introduced as they were coded into the program. Each of these potential sources of error will be discussed in more detail below.

Errors can arise if there is a flaw in the equations that form the foundation of the program. These equations can either have small errors that were not noticed during the derivation of the equations (e.g. using sine when cosine should have been used), or they can be theoretically flawed (e.g. based on assumptions that are not valid). Since most of the equations that are used in the program have been published for more than a decade, it is reasonable to conclude that this is not a source of error in the program.

A less obvious source of error is the inaccuracy that can be introduced by the computer because of “machine error” (the rounding-off of fractional numbers when they are stored in the computer's memory). Although each individual round-off error is extremely small, if it occurs in a section of the program that is executed frequently, or if the equations that use the number are particularly sensitive to the value, the inaccuracy can become a significant source of error. The parabola-line intersection routine that is used in the projectile algorithm was particularly sensitive to this problem (because it contains a square root term, which magnifies the error). A good deal of effort was required to remedy this problem.

Another potential source of error occurs when the equations are theoretically correct, but as they were coded into the program, typographical errors were introduced (e.g. typing "9.91" instead of "9.81"). Errors produced this way are more difficult to discover than one might think. They are often unnoticeable until the program is running (i.e. the program will compile and link without errors) and even then the errors may remain undetected until the program is run with a certain set of parameters. While an intelligent person will notice that a certain value is unreasonable and would be able to quickly track down their mistake, a program will continue with its calculations, unaware that an error has taken place. While some errors might be obvious, others may be subtle. Since computer programs are often used when hand-calculation is impractical, it can be difficult to detect errors in complex situations. For example, after running a large simulation, it would be hard to tell, without *extensive* hand calculations, that the velocity of a rock should be 9 m/s, when the program reported 7 m/s. Verification serves as a good tool for finding and eradicating these kinds of errors.

Verification Method

The verification consisted of a comparison between the program results and a set of manual calculations for the same geometry and parameters. For each example a fictitious slope and set of initial conditions for the rock were constructed, and the trajectory of the rock was calculated by hand. The same slope geometry and parameters were entered into RocFall, a simulation was performed, and the results were compared to the manual calculations.

The results from the program could not be replicated by hand when the initial conditions were specified by a random distribution. For this reason, all but one of the verification cases that were conducted did not include a random element (i.e. all standard deviations were set to 0). This made it difficult to test the coupling of the random number generation with the algorithms that use the random numbers. A technique for testing the random number generation and its use was the reason for creating the third verification model.

The results from RocFall were compared to another program when possible. That is, when both programs were able to accept all of the geometry and parameters in the example. The program that was used for comparison was "rockfall", written by Salvador(1989). This program permitted sliding only on the first slope segment and the sliding was limited to the

downslope direction. Because of this limitation, the program could only be used for comparison with the projectile example. Since both programs use the same equations to calculate the projectile motion, it would be expected that the programs would give identical results. This was in fact the case.

Verification Contents

The verification consists of four examples, each meant to verify a different aspect of the program. The first two examples verify the projectile algorithm and the sliding algorithm. It is important to verify these algorithms because the program uses these two algorithms to calculate the velocity and position of the rock. The third example was designed to verify the random number generation and statistical features of the program. It is important to verify the statistical components of the program because most rockfall analyses rely heavily on statistics. The fourth example was designed to verify that the collection of data for the graphs and the production of the graphs were correct. It is important that the graphs present correct information because many conclusions are drawn directly from the graphical output (e.g. the location where remedial measures should be placed).

Together, these four examples verify all of the major components of the program. The verification covers the workings of the program from the start of a simulation (the projectile verification), to the end (the sliding verification), to the presentation of the results (the envelope verification), and ensure that the statistical functions used in the simulation were correct (the probability verification).

A list of the verification cases is presented in the following table:

Verification Case	Title	Component Verified
1	Projectile	Projectile algorithm
2	Sliding	Sliding algorithm
3	Probability	Random number generation, and use
4	Envelopes	Data-collectors, envelopes and graphical output

Table A.0.1 - Verification contents