Appendix A

Numerical aspects of diffusion erosion of topographic profiles

&

User's guide to DIFFUSE

A. 1. Introduction

Efficient scientific progress may be achieved by ratcheting detailed field observations with careful theoretical analysis. We cannot solve geologic problems by burying ourselves in a profusion of data, nor by immersing ourselves in the minutiae of analysis and computation. Observations are incomplete perceptions of physical reality, but they are the test of the conceptual models we develop. The models are based upon simple assumptions about the behavior of physical reality. It is the interplay between the testing and refinement of models and the focusing of our repeated observations that builds the scientific intuition needed to solve geologic problems (*Margenau* [1950] and *D. D. Pollard, personal and written communications*, 1989-1994).

This appendix illustrates the numerical solution of a simple deterministic problem of topographic profile development under specific initial and boundary conditions, and a slope dependent constitutive material transport law. Elevation change with time is determined by the solution of the continuity equation for material transport [*Carson and Kirkby,* 1972; *Kirkby,* 1971]. Because the main governing equation is analogous to the heat conduction or chemical diffusion equation, this type of analysis has been called "diffusion erosion."

A variety of solution techniques are developed and investigated. A FORTRAN computer program, DIFFUSE, is provided and documented. Numerical experimentation demonstrates that a consistent set of assumptions will provide a possible explanation for observed forms and distributions [*Kirkby et al.,* 1993]. While many slope profiles can be simulated, it is up to the user to go out and make the observations to ratchet this investigation and his or her intuition further. Hopefully, a careful reading of this document and some numerical experiments with DIFFUSE will indicate a profitable path for field investigation.

A.1.1. About this document

This document is designed to discuss the essential aspects of the numerical determination of elevation change along profiles under "diffusion erosion" conditions. First, the numerical method of finite differences is described (Section A.2). The conditions of "diffusion erosion" are defined next and two finite difference techniques presented (Section A.3). A more precise solution is presented in Section A.4 in which the actual slope length over which material transport changes is explicitly considered, instead of its approximation as horizontal length. Section A.5 is a user's guide for DIFFUSE that discusses how to get the program, compile it, and run it; and input and output.

A.2. Numerical method--finite difference basics

A.2.1. Numerical solutions

While many analytic solutions are relatively easy to calculate, they depend on regular initial shapes and are not as flexible as numerical solutions. We use the technique of finite differences to approximate various equations in our analysis.

A.2.2. Finite differences background

Most of the following discussion is based upon *Chapra and Canale* [1988]; and *Ferziger* [1981]. Finite difference approximations for functions are based upon the Taylor Series approximation of a function at a point:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + f''\left(\frac{x_i}{2!}\right)(x_{i+1} - x_i)^2 + f'''\left(\frac{x_i}{3!}\right)(x_{i+1} - x_i)^3 + \ldots + f^{(n)}\left(\frac{x_i}{n!}\right)(x_{i+1} - x_i)^n + R_n \quad \text{(A.2.1)}$$

where R_n is a remainder term for terms (n+1) to infinity, and $f'(x_i)$ is the first derivative at x_i. Truncated after the first derivative, (A.2.1) becomes

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) \quad \text{(A.2.2)}$$

Solving for $f'(x_i)$ produces

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{(x_{i+1} - x_i)} - \frac{R_1}{(x_{i+1} - x_i)} \quad \text{(A.2.3)}$$

(1st order approximation - truncation error)

Then let $h = (x_{i+1} - x_i)$ [step size], and $\Delta f = f(x_{i+1}) - f(x_i)$ [first forward difference], so that the finite divided difference is

$$f'(x_i) = \frac{\Delta f}{h} - 0(h) \quad \text{(A.2.4)}$$

The above is the forward difference, but we may also have backward and central differences (*Figure A.1*); Centered difference approximation for 1st derivative:

Subtract $$f(x_{i-1}) = f(x_i) - f'(x_i)h + f''\frac{h^2}{2!} - \ldots \quad \text{(A.2.5)}$$

(backward difference)

from $$f(x_{i+1}) = f(x_i) + f'(x_i)h + f''\frac{h^2}{2!} + \ldots \quad \text{(A.2.6)}$$

(forward difference)

to yield $$f(x_{i+1}) = f(x_{i-1}) + 2f'(x_i)h + f'''\frac{h^3}{3!} + \ldots \quad \text{(A.2.7)}$$

and solve for $$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{f'''(x_i)}{6}h^2 \quad \text{(A.2.8a)}$$

or
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - 0h^2 \qquad \text{(A.2.8b)}$$

Equation A.2.8b is the centered or central difference representation of the first derivative. Notice that it is second order accurate.

A.2.2.1. Finite difference approximations of higher order derivatives

First, write a forward Taylor Series approximation of $f(x_{i+2})$ in terms of $f(x_i)$:

$$f(x_{i+2}) = f(x_i) + f'(x_i)(2h) + f''\frac{(x_i)}{2!}\left(2h^2\right) + \ldots \qquad \text{(A.2.9)}$$

Multiply (A.2.6) by 2 and subtract from (A.2.9) to yield

$$f(x_{i+2}) - 2f(x_{i+1}) = -f(x_i) + f''\frac{(x_i)}{2!}\left(2h^2\right) + \ldots \qquad \text{(A.2.10)}$$

Which is solved for

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} - 0h \qquad \text{(A.2.11)}$$

This relationship is called the *second forward finite divided difference.* Similar manipulations can be employed to derive a backward version

$$f''(x_i) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2} - 0h \qquad \text{(A.2.12)}$$

and a centered version

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} - 0h^2 \qquad \text{(A.2.13)}$$

As was the case with the 1st order approximations, the centered version is more accurate. Also notice that the centered version can alternatively be expressed as

$$f''(x_i) = \frac{\dfrac{f(x_{i+1}) - f(x_i)}{h} - \dfrac{f(x_i) - f(x_{i-1})}{h}}{h} - 0h^2 \qquad \text{(A.2.14)}$$

Thus, just as the second derivative is the derivative of a derivative, the second divided difference approximation is a difference of two first divided differences.

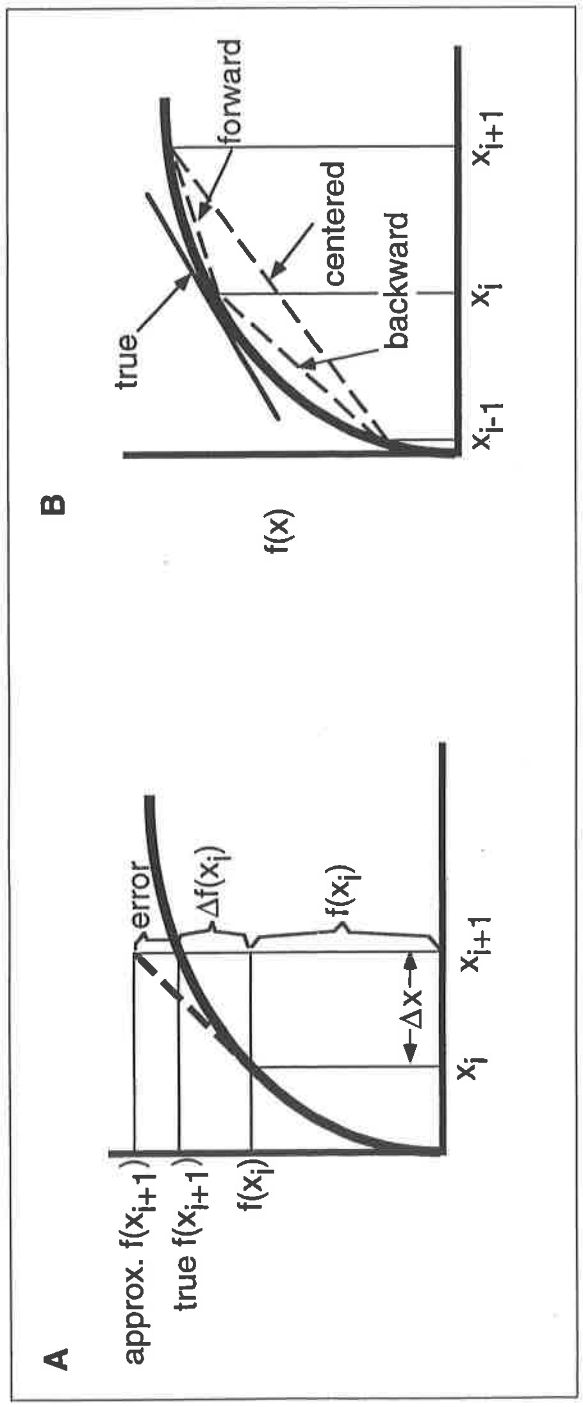


Figure A.1. A) Graphical description of forward difference and error; B) graphical representation of various difference approximations of the slope (1st derivative) of a function and their relation to the true slope at that point. Note the increased accuracy of the centered difference.

A.3. Diffusion erosion

Many approaches to the modeling of hillslope degradation governed by so-called diffusive processes are based upon the following derivation. The determination of the vertical component geomorphic displacement (erosion or deposition) is based upon the assumption of continuity of transportation of debris along the profile and is commonly referred to as the "continuity equation" (for example, [*Carson and Kirkby,* 1972; *Kirkby,* 1971]). In other words, in a given time increment Δt, the change in material per unit width contained along a slope segment (ΔQ) is the difference between that which is supplied from above (Q_A) and that which is removed from below (Q_R)--note that Q, the material transport rate per unit width, has dimensions of [L^2T^{-1}]:

$$Q_R - Q_A = \Delta Q \qquad (A.3.1)$$

Taking the limit in (A.3.1) and assuming transport-limited conditions and no tectonic vertical displacements leads to the relation that the change in elevation with time ($\frac{\partial H}{\partial t}$) is determined by the change in Q over the slope length:

$$\frac{\partial H}{\partial t} = -\frac{\partial Q}{\partial s} \qquad (A.3.2)$$

where *ds* is the change in slope length. *H* is the elevation of the topographic surface, and is not an independent variable in this analysis. It is dependent upon *t*, time, and *x*, the distance along the slope profile. The negative sign reflects the fact that a positive change in material transport rate corresponds to a positive change in elevation. See *Figure A.2* for an illustration of the profile geometry. (A.3.2) is simplified by approximating *ds* with *dx*, the horizontal distance:

$$\frac{\partial H}{\partial t} = -\frac{\partial Q}{\partial x} \quad \text{Continuity Equation} \qquad (A.3.3)$$

The assumption of $ds \sim dx$ will result in an overprediction of geomorphic displacement rate (because $dx \leq ds$ --in the denominator--for all but zero slope). However, the difference in *ds* and *dx* is small for the low slopes (<45°) and therefore, the approximation is justified in order to simplify our analysis (see Section A.4). Most of the derivatives in this analysis are of functions of two more variables (e. g., A.3.2); therefore, the equations must include the partial derivative form, and we assume that the other, undifferentiated variables are

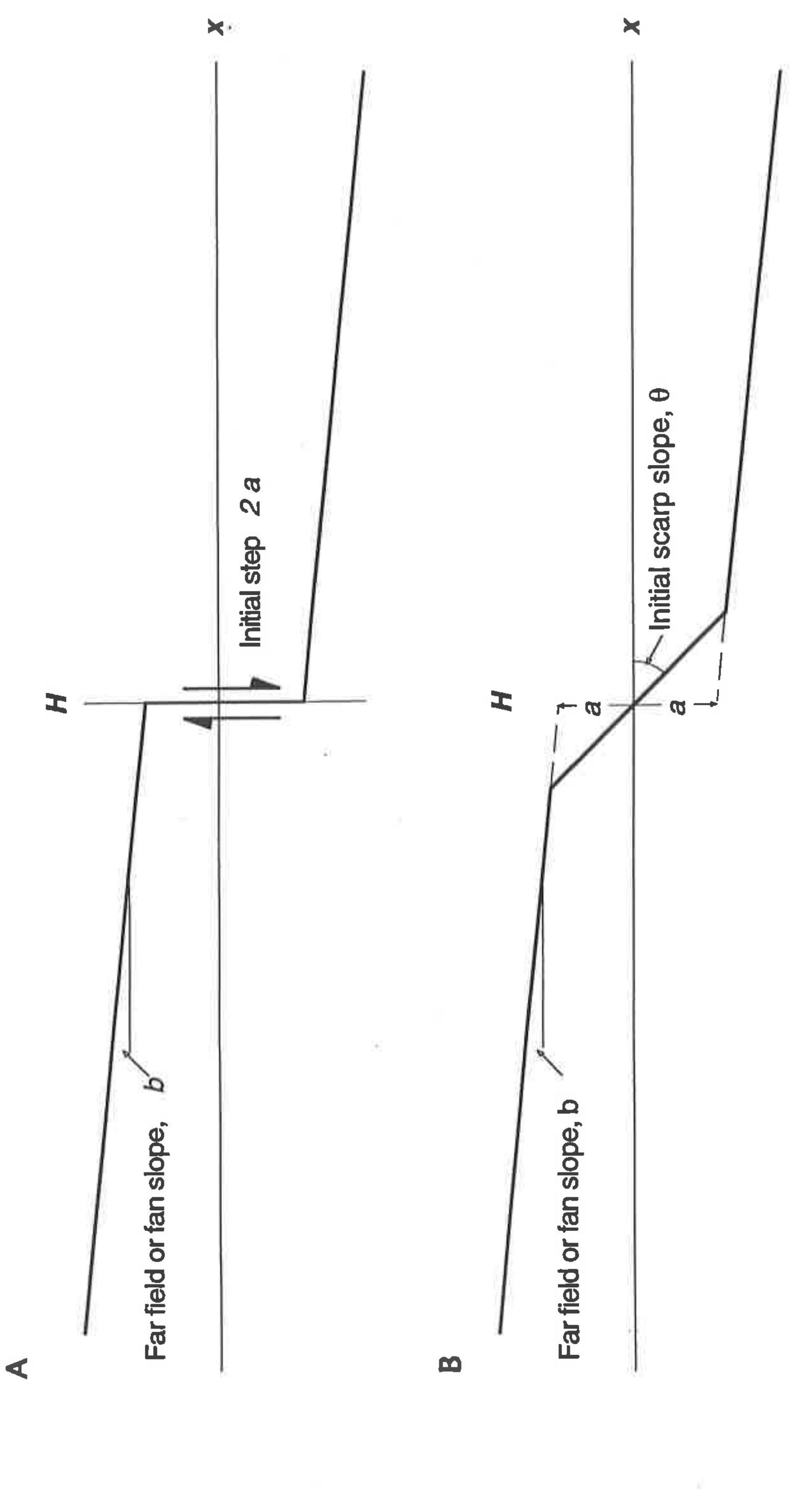


Figure A.2. Scarp geometry. A) Vertical scarp; B) finite initial slope conditions.

constant: this is appropriate in that rates of change are relative to particular dimensions and might depend upon other variables.

A.3.1. Constitutive Equation

The continuity equation presented above includes the consideration of no processes--only the fact that a change in material transport rate over a given slope segment will result in a change in elevation. We specify the rule for material transport as the following:

$$Q = -\alpha \, {\partial H}/{\partial x} \qquad \text{(A.3.4)}$$

where α is a constant of proportionality, and the negative sign is necessary for positive transport rate down a negative slope. This slope dependent law results from rainsplash, creep, thermal expansion, and animal induced disturbances [*Carson and Kirkby,* 1972; *Culling,* 1963; *Selby,* 1985]. It does not allow for slope-length dependent processes (where *Q* is proportional to *x*), such as overland flow [*Carson and Kirkby,* 1972]. Rainsplash impact can be as much as 256 times more energetic than overland flow, and therefore must dominate the surficial processes [*Carson and Kirkby,* 1972; *Selby,* 1982].

In the case of creep, *Mitchell* [1976] (as cited in *Nash* [1980]) showed that the shear strain rate is proportional to shear stress for the low stresses acting on hillslope materials. The shear stress in a slope is proportional to the sine of the slope angle:

$$Q = -\alpha \sin\theta \qquad \text{(A.3.5)}$$

where θ is the slope angle measured down form the horizontal. Because typical slope angles are less that 30°, $\sin\theta \approx \tan\theta = dH/dx$. Therefore, for small values of θ, (A.3.5) and (A.3.4) are approximately equivalent.

Combining (A.3.2) and (A.3.4) results in the following differential equation for the change in elevation with time:

$$\frac{\partial H}{\partial t} = k\frac{\partial}{\partial s}\left({\partial H}/{\partial x}\right) \quad \text{(true slope length case)} \qquad \text{(A.3.6)}$$

Where $k = \alpha$. If $ds \approx dx$ (i.e., using (A.3.3) and $\beta = 1$) then (A.3.6) is simplified as

$$\frac{\partial H}{\partial t} = k\frac{\partial^2 H}{\partial x^2} \quad \text{(linear diffusion case)} \qquad \text{(A.3.7)}$$

Where the vertical geomorphic displacement rate is proportional to the curvature of the landscape. This analysis has been used in several geomorphic and tectonic applications (e.g., [*Avouac,* 1993; *Colman,* 1987; *Culling,* 1963; *Hanks and Andrews,* 1989; *Hanks et al.,* 1984; *Hirano,* 1968; *Nash,* 1980; *Nash,* 1981]). A.3.7 is analogous to the heat conduction equation (homogeneous linear diffusion equation) for one dimensional heat transport in which k would be the coefficient of proportionality called the thermal diffusivity and H would be temperature (e.g., *Avouac,* 1993; *Hanks et al.,* 1984; *Nash,* 1980).

A.3.2. Vertical scarp initial conditions

First, we will illustrate the analytic solutions to the simple, linear diffusion case (an approach presented in *Hanks et al.* [1984]. A solution of the homogeneous diffusion equation (A.3.7) for a step of topography of $2a$ at $t = 0$ and $x = 0$ (e.g., a newly formed fault scarp, and note that the origin for x in this example is in the mid scarp and not at the upper boundary; see *Figure A.2* for the geometry of these model scarps) on a pre-existing topographic slope of b is

$$H(x,t) = a \cdot erf\left[\frac{x}{2\sqrt{(kt)}}\right] + b \cdot x \qquad \text{(A.3.8)}$$

in which H is elevation, x is the horizontal dimension, t is time, and k is the erosive diffusivity. The resulting profiles are not uniquely dependent on t or k, but rather their product. For example, the same profile will result for $k = 1$ m^2/ka and $t = 10$ ka or $k = 10$ m^2/ka and $t = 1$ka (*T. C. Hanks personal communication* and [*Hanks et al.,* 1984]). For this presentation, we have separated these values to emphasize the fact that a known value for k or t may be used to constrain the other. *Figure A.3* shows the geometry of a simple fault scarp and a graphical solution to (A.3.8) for a step of 2 m at time zero and a far field or fan slope of 5°, and its subsequent degradation with time.

A.3.3. Ramp (finite) slope initial conditions

Equation (A.3.8) provided a solution of the homogeneous diffusion equation (A.3.7) for a step of topography of $2a$ at $t = 0$ and $x = 0$ (e.g., a newly formed fault scarp) on a pre-existing topographic slope of b and an infinite slope (vertical) initial scarp slope (*Figures A.2 and A.4*). Clearly, a vertical scarp will not last long, nor will "diffusive" processes operate on such steep slopes. Therefore, it is important to develop the analysis for finite initial slope scarps. *Hanks and Andrews* [1989] provided the following equation for the same conditions that give rise to (A.3.7), then, except that the fault "occurs" on a slope of θ instead of ∞, the solution to (A.3.7) is:

$$\begin{aligned} H(x,t) &= (\theta - b)\left(\frac{kt}{\pi}\right)^{1/2} \\ &\cdot\left\{exp\left(-\frac{[x + a/(\theta-b)]^2}{4kt}\right) - exp\left(-\frac{[x - a/(\theta-b)]^2}{4kt}\right)\right\} \\ &+\frac{(\theta-b)}{2}\left\{\begin{matrix}\left(x + \dfrac{a}{(\theta-b)}\right)\cdot erf\left(\dfrac{x + a/(\theta-b)}{[4kt]^{1/2}}\right) \\ -\left(x + \dfrac{a}{(\theta-b)}\right)\cdot erf\left(\dfrac{x - a/(\theta-b)}{[4kt]^{1/2}}\right)\end{matrix}\right\} + b\cdot x \end{aligned} \qquad \text{(A.3.9)}$$

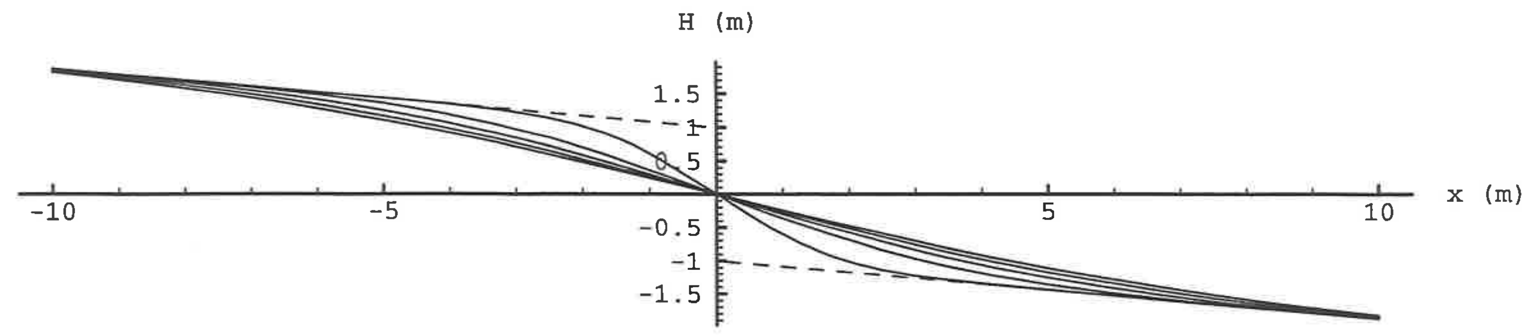


Figure A.3. The geometry of a simple fault scarp (dashed line is time = 0), and its subsequent degradation with time (t = 1 ka, 4 ka, 7 ka, 10, ka , 13 ka; a = 1 m; k = 1 m^2/ka; b = 5 degrees).

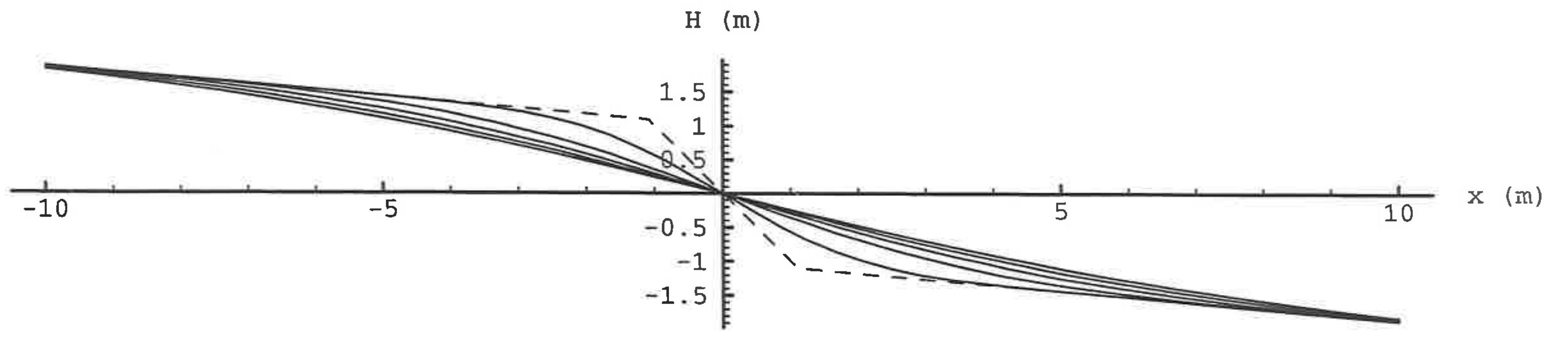


Figure A.4. The geometry of a finite value initial slope fault scarp (dashed line is time = 0), and its subsequent degradation with time (t = 1 ka, 4 ka, 7 ka, 10, ka , 13 ka; a = 1; k = 1 m^2/ka; b = 5 degrees, and θ = 45 degrees).

in which *H* is elevation, *x* is the horizontal dimension, *t* is time, *b* is the far-field or fan slope, and *k* is the erosive diffusivity. *Figure A.3* shows the geometry of a simple fault scarp, and a graphical solution to (A.3.9) for a step of 2 m at time zero, a far field or fan slope of 5°, and an initial scarp slope of 45° and its subsequent degradation with time is shown in *Figure A.4*.

A.3.4. Comments

Although the diffusive representation of erosion ignores the mechanics of sediment particle motion, it adequately describes scarp degradation in non-dissecting alluvial terrains over a wide range of scales ($0 \leq 2a \leq 50$ m, and $3 \leq$ age ≤ 400 ka; e.g., [*Hanks et al.*, 1984; *Nash,* 1980]). That conservation of mass holds on a local scale is an important assumption. This cannot be invoked for entrenched stream channels, and approaches are presented elsewhere for that case. The results of diffusion erosion analyses depend strongly on a well constrained value for the diffusion coefficient *k*.

A.3.5. Numerical solutions to the basic linear diffusion erosion case

Here we present an explicit finite difference method for solving equation (A.3.7) for arbitrary initial conditions (see *Figure A.5* for the temporal and spatial discretization grid and computational molecule). First, approximate the time derivative (left hand side of (A.3.7)) with a forward divided difference (A.2.4):

$$\frac{\partial H}{\partial t} \approx \frac{H_i^{l+1} - H_i^l}{\Delta t} \tag{A.3.10}$$

where we have assumed no tectonic input and a transport-limited case (infinitely thick soil). Secondly, approximate the space derivative by a centered divided finite difference (A.2.13):

$$\frac{\partial^2 H}{\partial x^2} \approx k\frac{\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right)}{\Delta x^2} \tag{A.3.11}$$

Substitute (A.3.10) and (A.3.11) into (A.3.7) to get

$$\begin{aligned} \frac{H_i^{l+1} - H_i^l}{\Delta t} &= k\frac{\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right)}{\Delta x^2} \\ H_i^{l+1} &= H_i^l + \frac{k\Delta t}{\Delta x^2}\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right) \\ H_i^{l+1} &= H_i^l + \lambda\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right) \end{aligned} \tag{A.3.12}$$

where $\lambda = \frac{k\Delta t}{\Delta x^2}$. This equation is written for all interior nodes of the profile (*Figure A.5*). It then provides an explicit means of computing values at each node for a future time, based

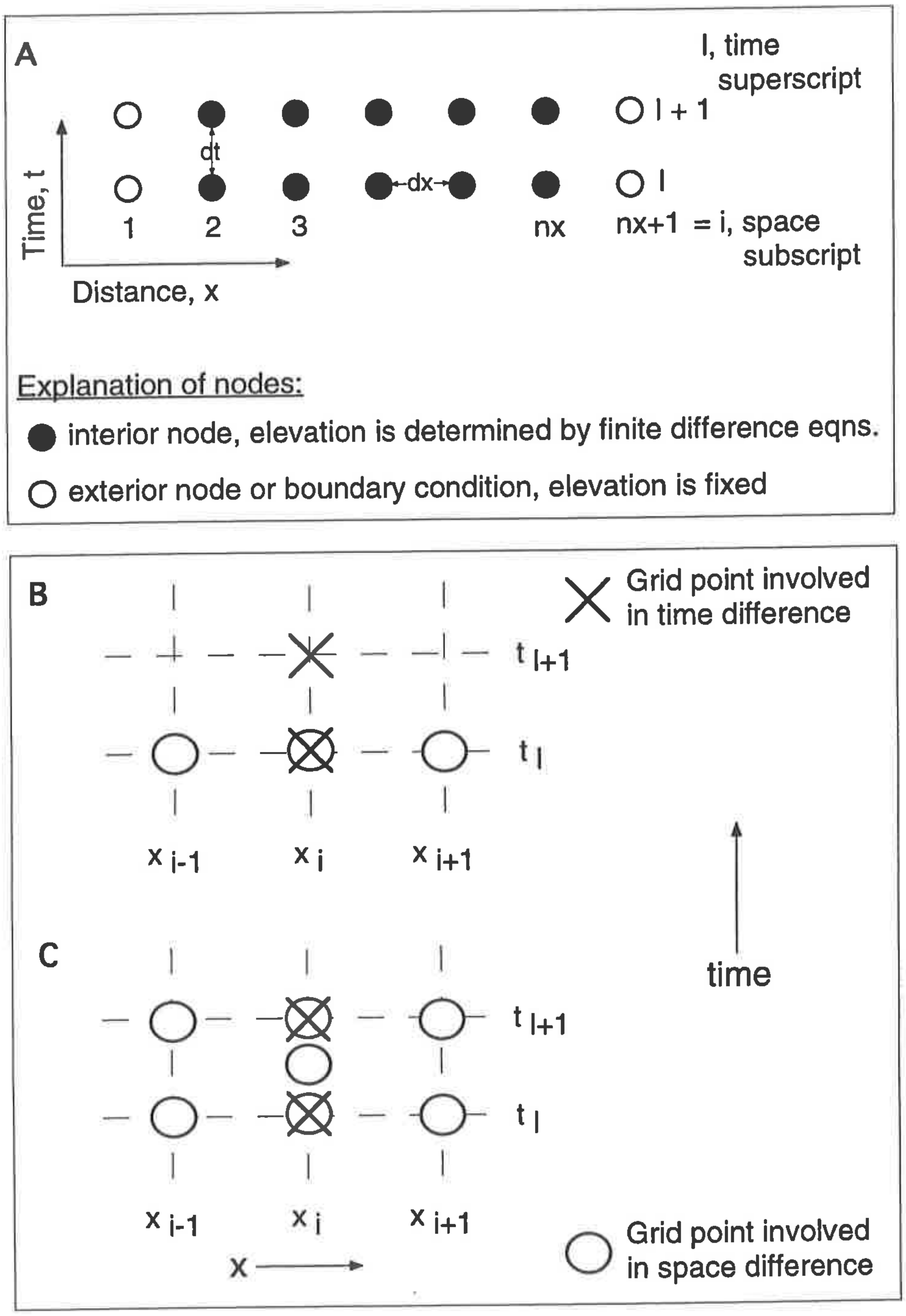

A
l, time superscript
l + 1
dt
l
dx
Time, t
1
2
3
nx
nx+1 = i, space subscript
Distance, x
Explanation of nodes:
● interior node, elevation is determined by finite difference eqns.
○ exterior node or boundary condition, elevation is fixed
B
Grid point involved in time difference
t_{l+1}
t_l
x_{i-1}
x_i
x_{i+1}
time
C
t_{l+1}
t_l
x_{i-1}
x_i
x_{i+1}
x
Grid point involved in space difference

Figure A.5. A) Grid definition and geometry and node type definition. B) Computational molecule for 1-D explicit finite difference method. C) Computational molecule for 1-D implicit Crank-Nicholson Method.

upon the present values. This explicit method is convergent and stable for $\lambda \leq \frac{1}{2}$, or $\Delta t \leq \frac{1}{2}\frac{\Delta x^2}{k}$ [*Ferziger,* 1981].

A.3.5.1. An example application of the 1-D explicit finite difference method solved using a spreadsheet (Microsoft Excel)

Table A.1 is an example spreadsheet used to solve (A.3.7) by (A.3.12). The constants in solid boxes are input, and the spreadsheet determines the rest.

TABLE A.1. Example Microsoft Excel Spreadsheet for explicit finite difference. The values inside the bold boxes can be changed and the result plotted in a chart.

Constants:		units
dx =	4	m
dt =	0.05	ka
k =	1	m^2/ka
lambda =	0.003125	
Age =	5	ka

		I (time)	
		0	0.05
x	**i (distance)**	1	2
-10	0	1	1
-6	1	1	1
-2	2	1	0.99375
2	3	-1	-0.99375
6	4	-1	-1
10	5	-1	-1

Table A.2 is the same spreadsheet with the equivalent formulae for Table A.1. In order to calculate to a given age, we must copy the D column to the right enough times to keep the calculation stable. For example, to determine the profile shown in *Figure A.6A*, 100 time steps, or 100 columns were used. For the given parameters such a result is well within the stability criterion (unlike *Figure A.6B*; a nice example of an unstable solution).

TABLE A.2. Example Microsoft Excel Spreadsheet for explicit finite difference with formulae shown.

Row #	Column A	B	C	D
			I (time)	
			=C43*dt-dt	=D43*dt-dt
	x	i (distance)	1	2
44	=-10+B44*dx	0	1	=C44
45	=-10+B45*dx	1	1	=C45+lambda*(C46-2*C45+C44)
46	=-10+B46*dx	2	1	=C46+lambda*(C47-2*C46+C45)
47	=-10+B47*dx	3	-1	=C47+lambda*(C48-2*C47+C46)
48	=-10+B48*dx	4	-1	=C48+lambda*(C49-2*C48+C47)
49	=-10+B49*dx	5	-1	=C49

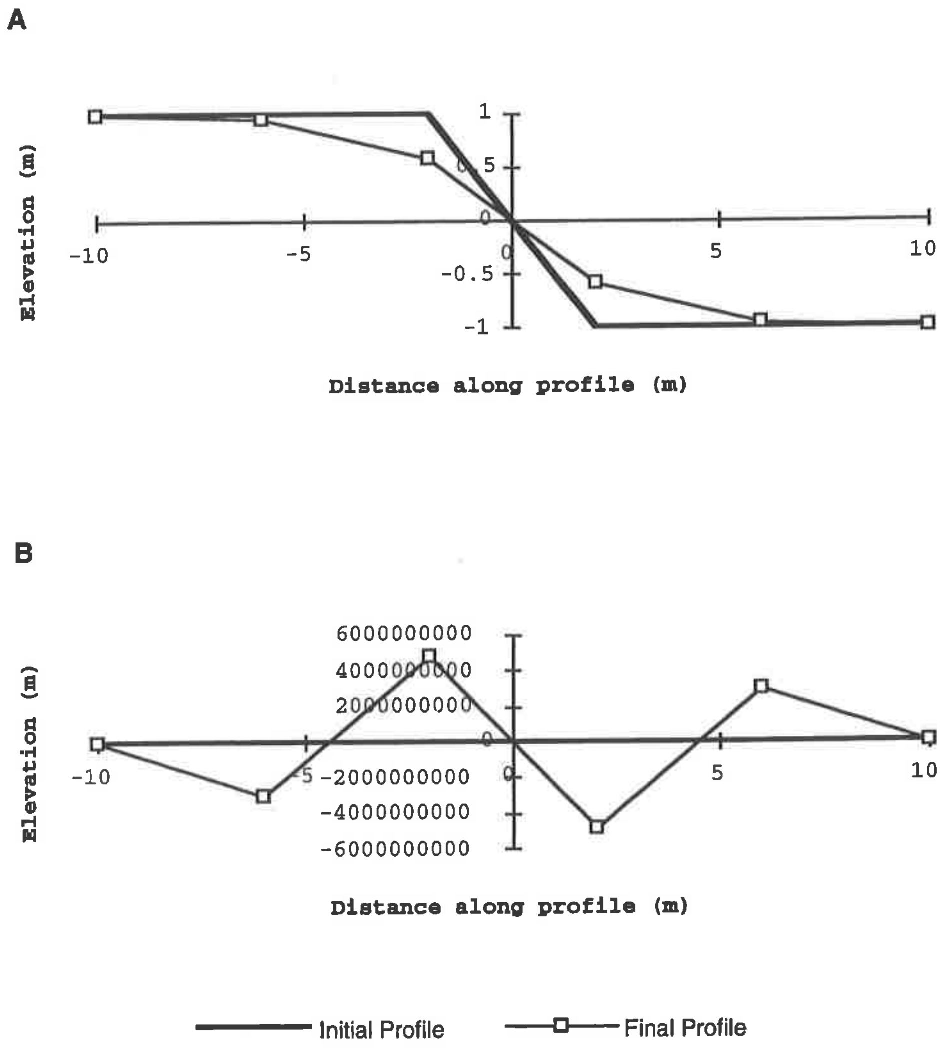


Figure A.6. Microsoft Excel plots showing the initial and final profiles determined using the sample spreadsheet (Tables A.1 and A.2). A) Diffusion erosion solution for degraded step; 1-D explicit finite difference technique, $t = 5$ ka, $k = 1$ m^2/ka. B) Unstable diffusion erosion solution for degraded step; 1-D explicit finite difference technique, $t = 1000$ ka, $k = 1$ m^2/ka (with a large $\lambda = 0.625$; exceeding the stability criterion).

A.3.5.2. Sample computer code for numerical simulation of linear diffusion

In this section we present the sample routine that determines the changes in elevation with time based upon (A.3.6). Note that this is a subroutine in the program DIFFUSE (Section A.5). This type of explicit method is occasionally called an "Euler method;" thus the name of the subroutine [*Ferziger,* 1981]. See *Atkinson et al.* [1989] or *Metcalf* [1989] for references to programming in FORTRAN.

Definition of variables

nt is the number of times the program determines the change in elevation as it goes through the main Do loop. This number depends upon the value of the timestep **dt**, which is defined by the stability criterion $\Delta t \leq \frac{1}{2}\frac{\Delta x^2}{k}$.

i is the space counter.

l is the time counter.

data is the vector containing the elevations of all nodes both interior and exterior. It is assumed that the elevation vector has been initialized with the starting profile before this routine is called. It is returned with the elevations of the degraded profile when the routine finishes.

lambda = $\lambda = \frac{k\Delta t}{\Delta x^2}$.

```
      Subroutine euler (nx, size, data, nt, lambda)

C     Calculates new elevation vector by explicit centered
C     finite difference

C     i --> Time counter
C     l --> Space counter
C     size --> Long dimension of vector
C     temp --> Temporary elevation vector

      integer i, l, nt, nx, size
      real data(size,2), temp(1001,2), lambda

C     Calculate the change in elevation nt times
      do 10 l=1,nt

C           calculate the new elevations for this increment
            do 20 i=2,nx
                  temp(i,2)=data(i,2)+lambda*
     +                 (data(i+1,2)-(2*data(i,2))+data(i-1,2))
20          continue

C     Assign new elevation vector to elevation vector
            do 30 i= 2,nx
                  data(i,2) = temp(i,2)
30          continue
10    continue
      END
C     end euler
```

A.3.5.3. The Crank-Nicholson Method

The Crank-Nicholson method is an alternate implicit scheme that is second order accurate in both space and time: the difference equations are developed for the middle of the increment. It is also stable for larger values of λ than the explicit method. This example hints at the sophistication possible for some applications of finite differences. The temporal first derivative of (A.3.7) is approximated at $t^{l+1/2}$ by

$$\frac{\partial H}{\partial t} = \frac{H_i^{l+1} - H_i^l}{\Delta t} \qquad \text{(A.3.13)}$$

The spatial second derivative for the midpoint is determined by averaging the difference approximations at t^l and t^{l+1} (*Figure A.5*):

$$k\frac{\partial^2 H}{\partial x^2} \approx k\frac{\left(H_{i+1}^{l+1} - 2H_i^{l+1} + H_{i-1}^{l+1}\right) - \left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right)}{2\Delta x^2} \qquad \text{(A.3.14)}$$

Substituting (A.3.13) and (A.3.14) into (A.3.7) and collecting terms gives

$$-\lambda H_{i-1}^{l+1} + 2(1+\lambda)H_i^{l+1} - \lambda H_{i+1}^{l+1} = \lambda H_{i-1}^l + 2(1+\lambda)H_i^l + \lambda H_{i+1}^l \qquad \text{(A.3.15)}$$

where $\lambda = \dfrac{k\Delta t}{\Delta x^2}$.

Boundary conditions of $H_1^{l+1} = f_1\left(t^{l+1}\right)$ and $H_{nx+1}^{l+1} = f_{nx+1}\left(t^{l+1}\right)$ can be prescribed to derive equations for the first interior node (i = 2):

$$2(1+\lambda)H_2^{l+1} - \lambda H_3^{l+1} = \lambda f_1\left(t^l\right) + \lambda f_1\left(t^{l+1}\right) + 2(1-\lambda)H_2^l + \lambda H_3^l \qquad \text{(A.3.16)}$$

Which is simplified as

$$2(1+\lambda)H_2^{l+1} - \lambda H_3^{l+1} = 2\lambda f_1(t) + 2(1-\lambda)H_2^l + \lambda H_3^l \qquad \text{(A.3.17)}$$

and for the last interior node (i = nx):

$$-\lambda H_{nx-1}^{l+1} + 2(1+\lambda)H_{nx}^{l+1} = \lambda H_{nx-1}^l + 2(1+\lambda)H_{nx}^l + \lambda f_{nx+1}\left(t^l\right) + \lambda f_{nx+1}\left(t^{l+1}\right) \qquad \text{(A.3.18)}$$

(the above equation is wrong in *Chapra and Canale* [1988]). (A.3.18) is simplified as

$$-\lambda H_{nx-1}^{l+1} + 2(1+\lambda)H_{nx}^{l+1} = \lambda H_{nx-1}^l + 2(1+\lambda)H_{nx}^l + 2\lambda f_{nx+1}(t) \qquad \text{(A.3.19)}$$

(A.3.15) can be written in algebraic notation as

$$H^{l+1}\mathbf{M} = rhs \qquad \text{(A.3.20)}$$

where H^{l+1} is the vector or elevations at time $l+1$ (the unknown), **M** is the tridiagonal matrix of the form:

node # in equation -> node # of equation below	**i = 2**				**i = nx**
i = 2	2(1+λ)	−λ			
	−λ	2(1+λ)	−λ		
		−λ	2(1+λ)	−λ	
				etc....	
i = nx				−λ	2(1+λ)

and *rhs* is the vector of known elevations (H^l) multiplied by the appropriate constants (λ or $2(1-\lambda)$, i.e., A.3.15). (A.3.20) is then solved for H^{l+1} using a tridiagonal matrix solution technique [*Atkinson et al.*, 1989; *Ferziger*, 1981].

A.3.5.4. Sample computer code for 1-D Crank-Nicholson implicit finite difference scheme

In this section we present the sample routine that determines the changes in elevation with time based upon (A.3.15 and A.3.20). Note that this is a subroutine in the program DIFFUSE (Section A.5). The subroutine **`vctr`** determines the value of the vector **`rhs`** for the tridiagonal solution of (A.3.20):

```
      Subroutine vctr (rhs, data, cnlambda, nx, size)

C     The subroutine vctr determines the value of the
C     vector rhs for the tridiagonal solution of the
C     matrix equation of the crank-nicholson scheme:

      integer nx, size
      real data(size, 2), cnlambda, rhs(size)

C     update first interior node
            rhs(1) = 2*cnlambda*data(1,2)
     +            + 2*(1-cnlambda)*data(2,2)
     +            + cnlambda*data(3,2)

C     update fully interior nodes
       do 54 i=3,nx-1
            rhs(i-1) = cnlambda*(data(i-1,2)
     +            + data(i+1,2))
     +            + 2*(1-cnlambda)*data(i,2)
54    continue

C     update last interior node
            rhs(nx-1) = cnlambda*data(nx-1,2)
     +            + 2*(1-cnlambda)*data(nx,2)
     +            + 2*cnlambda*data(nx+1,2)

       END
C     end vctr
```

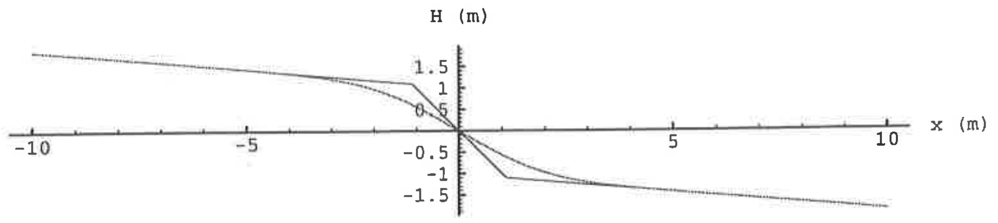
The subroutine **crank** solves (A.3.15) constrained by initial and boundary conditions and variables defined above by using a tridiagonal solver to return the final values of **data**:

```
      Subroutine crank (nx, size, data, k, dx, tf)

C     This subroutine determines the new elevations by using the
C     Crank-Nicholson implicit scheme.

C     i --> Time counter
C     l --> Space counter
C     size --> Long dimension of vector
C     p --> Temporary elevation vector
C     cnlambda --> Crank-Nicholson lambda
C     rhs --> Right-hand side vector of Crank-Nicholson equation
C     band --> Vector for upper and lower bands of tridiagonal matrix
C     main --> Main band of tridiagonal matrix

      integer i, nt, l, size
      real data(size, 2), dx, k, tf, cnlambda, rhs(1001),
     +      band(1001), main(1001), dt, p(1001)

C     Determine new cnlambda for greater stability of CN method
C     Determine a stable nt and dx.  Note that the first dt
C     is provisional

      dt = (1*(dx**2))/k
      nt = int(tf/dt)+1
      dt = tf/nt
      cnlambda = (k*dt)/(dx**2)

      print *, 'Crank-Nicholson nt =', nt
      print *, 'Crank-Nicholson lambda =', cnlambda

C     Build bands for matrix
      do 51 i=1,nx-2
            band(i) = (-cnlambda)
51    continue

C     Build main band
      do 52 i=1,nx-1
            main(i) = 2*(1 + cnlambda)
52    continue

C     Calculate elevation change nt times
      do 53  l=1,nt
            call vctr (rhs, data, cnlambda, nx, size)
            call trdiag (nx-1, band, main, band, p, rhs, size)
            do 55 i=2,nx
                  data(i,2)=p(i-1)
55          continue
53    continue
      END
C     end crank
```

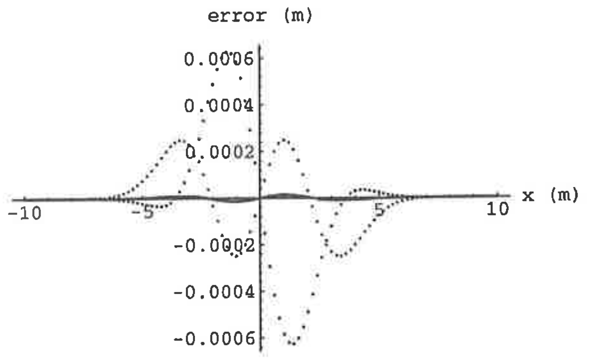
A.3.6. Verification of numerical techniques: Comparison with analytic solutions

In order to verify the numerical techniques presented above, we compare the results for several numerical simulations with the corresponding analytic solution with the same initial and boundary conditions and values for the constants. *Figure A.7* graphically indicates our initial success with this numerical approximation. The small errors are typical of the method [*Ferziger,* 1981] and the even smaller differences between the techniques are overshadowed by the error caused by the infinite curvature of the initial condition (*Ferziger* [1981] points out that the global error of the approximation of a function is proportional to the second derivative of that function, so the first time step has a large error). The source of the greatest error is probably the discretization of the initial profile in which the sharp corner of the ramp may not coincide with a node and thus will be cut off. Another result to note is that decreasing Δt (increasing *nt*--the number of time steps) and increasing Δx (decreasing *nx*--the number of space steps) diminish error (except that increasing Δx causes the discretization error to increase). This error dependence results from $\lambda = \frac{k\Delta t}{\Delta x^2}$. Clearly, decreasing Δx makes λ increase and thus promotes a divergent (unstable) solution [*Ferziger,* 1981].

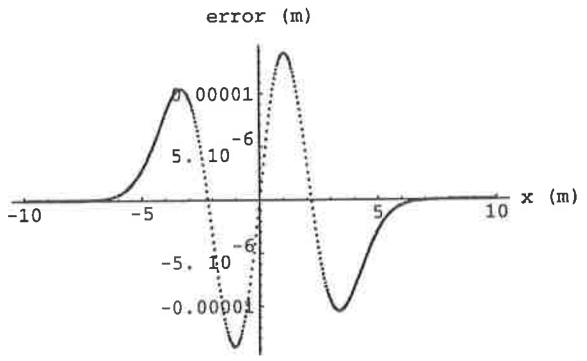
A



B



C



D

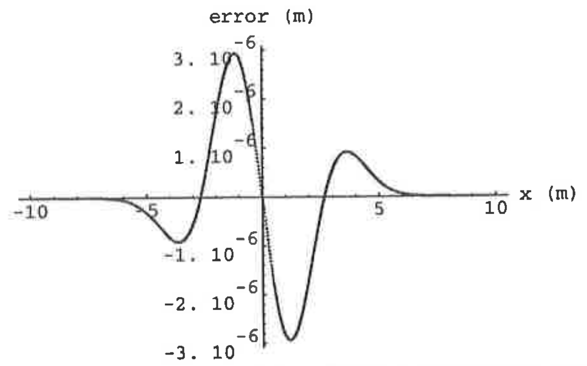


Figure A.7. Residual error (analytic - numeric) for different numerical approximations to the diffusion erosion problem with these parameters specified: $k = 1$ m^2/ka, $b = 5°$, $\theta = 45°$, and $a = 1$ m, and the final time is 1 ka. Note how the error decreases with both the technique (explicit or Crank-Nicholson implicit--CNImplicit), and increasing number of timesteps (*nt*); while increasing the number space steps (*nx*) actually pushes the error up. Consider how D might plot in C and in B, and that the errors are arbitrarily small (depend only on *nt* and *nx*) and acceptable. The results are reported in terms of *nx* x *nt*. A) Graphical plot of solution. B) Comparison of error for explicit method (100 x 100; largest error), CNImplicit (100 x 20; medium error) and CNImplicit (500 x 500; least error). C) Blow-up of the least error in B: CNImplicit (500 x 500). D) Residual error for the extreme example of Explicit (1000 x 10,000).

A.4. Viability of the slope length = horizontal distance approximation

In the previous section, we investigated the variety of possibilities available with the linear diffusion case for erosion. Recall the equations:

$$\frac{\partial H}{\partial t} = k\frac{d}{ds}\left(\partial H/_{\partial x}\right) \quad \text{(true slope length case)} \qquad \text{(A.4.1)}$$

If $ds \approx dx$, then (A.4.1) is simplified as

$$\frac{\partial H}{\partial t} = k\frac{d^2 H}{dx^2} \quad \text{(linear diffusion case)} \qquad \text{(A.4.2)}$$

The simplification of $ds \approx dx$ is reasonable in that the angles over which these processes occur are generally small. However, we may investigate this approximation further. First, we evaluate the geometric consequences of this approximation.

The source of error results from the variation of *ds* as a function of *dx* and *dH*. In other words, the change in mass transport (*dQ*) occurs over a constant *ds*, and that is not a constant *dx* when projected onto the horizontal; rather it varies with *dH* (*Figure A.8*).

Therefore, the use of (A.4.2) incorporates an approximation, the effects of which may cause an overprediction of elevation change with time because *dQ/dx > dQ/ds*. In other words,

$$ds = \sqrt{dx^2 + dH^2} \qquad \text{(A.4.3)}$$

and if we hold *dx* fixed, then *ds* will vary from a minimum of *dx* when $dH = 0$ to ∞ as *dH* increases. We can reformulate this in terms of the slope angle, θ (slope angle measured down from horizontal):

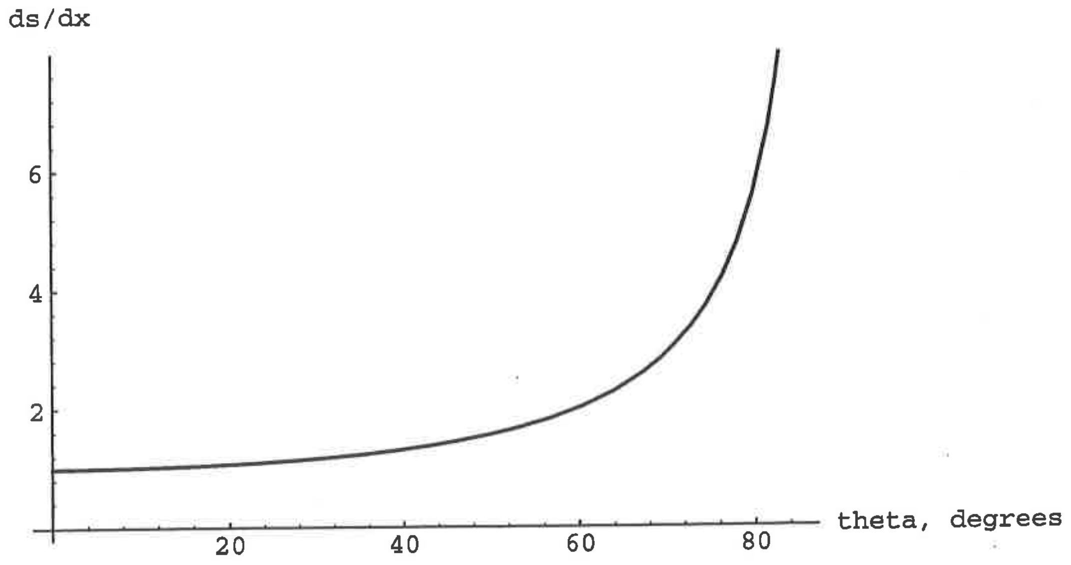
$$ds = \frac{dx}{\cos\theta} \qquad \text{(A.4.4)}$$

As cos θ varies from 1 to 0 as θ varies from 0 to 90°, *ds* in terms of *dx* will vary from *dx* to ∞, indicating that only for $dH = 0$, will $ds = dx$, and (A.4.2) explicitly hold (*Figure A.8*). However, for $dH = 0$, there would be no slope, and thus not material transport, violating our assumption of constitutive behavior:

$$Q = -\alpha\, \partial H/_{\partial x}\ . \qquad \text{(A.4.5)}$$

For the case of $\theta < 45°$ (typical of slope investigations in which the diffusive processes will operate), the profile changes determined using (A.4.2) should indicate faster erosion than for (A.4.1); however, the < 41% variation may not significantly effect the resulting profile given the possible temporal and spatial sampling errors of real profiles.

Investigation of our hypothesis based upon the above analysis indicates that the linear diffusion simplification will overestimate the material transport rate, and thus the



Definition of variables:

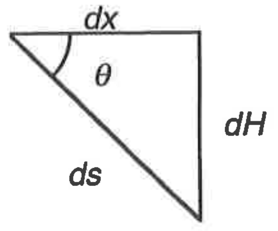


Figure A.8. Effect of slope angle on *ds/dx*.

erosion rate. Using the method of finite differences, then, we can quantitatively simulate the development of profiles governed by (A.4.1) and (A.4.2). We will use the same formulation for the approximation of (A.4.1) as was presented in Section A.3 for (A.4.2).

The following is the formulation of the finite difference approximation of (A.4.1): First, approximate the time derivative (left hand side of (A.4.1)) with a forward divided difference (A.2.4):

$$\frac{\partial H}{\partial t} \approx \frac{H_i^{l+1} - H_i^l}{\Delta t} \tag{A.4.6}$$

where we have assumed no tectonic input and a transport-limited case (infinitely thick soil). Secondly, approximate the space derivative by a centered divided finite difference (A.2.8):

$$k\frac{\partial Q}{\partial s} \approx k\frac{Q_{i+\frac{1}{2}}^l - Q_{i-\frac{1}{2}}^l}{\Delta s} \tag{A.4.7}$$

In (A.4.7), we have introduced imaginary nodes at i ± 1/2, and will approximate the material transport rate at those points based upon our assumption of the constitutive equation (A.4.5):

$$Q_{i+\frac{1}{2}}^l = \left(\frac{H_{i+1}^l - H_i^l}{\Delta x}\right) \tag{A.4.8}$$

$$Q_{i-\frac{1}{2}}^l = \left(\frac{H_i^l - H_{i-1}^l}{\Delta x}\right)$$

Now, we substitute (A.4.8) into (A.4.7):

$$k\frac{\partial Q}{\partial s} \approx k\frac{\left(\frac{H_{i+1}^l - H_i^l}{\Delta x}\right) - \left(\frac{H_i^l - H_{i-1}^l}{\Delta x}\right)}{\Delta s} = k\left(\frac{H_{i+1}^l - 2H_i^l + H_{i-1}^l}{\Delta x \cdot \Delta s}\right) \tag{A.4.9}$$

Simplifying, collecting terms, and combining with (A.4.6):

$$H_i^{l+1} = H_i^l + \frac{k\Delta t}{\Delta x \cdot \Delta s}\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right) \tag{A.4.10}$$

Clearly, if $\Delta x = \Delta s$, (A.4.10) reduces to the explicit finite difference formulation for linear diffusion (A.3.12). Finally, substitute finite difference version of (A.4.3) into (A.4.10) to get our explicit finite difference approximation of (A.4.1):

$$H_i^{l+1} = H_i^l + \frac{2k\Delta t}{\Delta x \cdot \sqrt{(2\Delta x)^2 + \left(H_{i+1}^l - H_{i-1}^l\right)^2}}\left(H_{i+1}^l - 2H_i^l + H_{i-1}^l\right) \tag{A.4.11}$$

Note that we have approximated the slope length from nodes *(i+1)* to *(i-1)* for greater accuracy. *Figure A.9* shows the difference in the resulting profiles and indicates that the approximation is probably warranted, especially for larger *kt* values (i.e., lower slopes and *ds -> dx*).

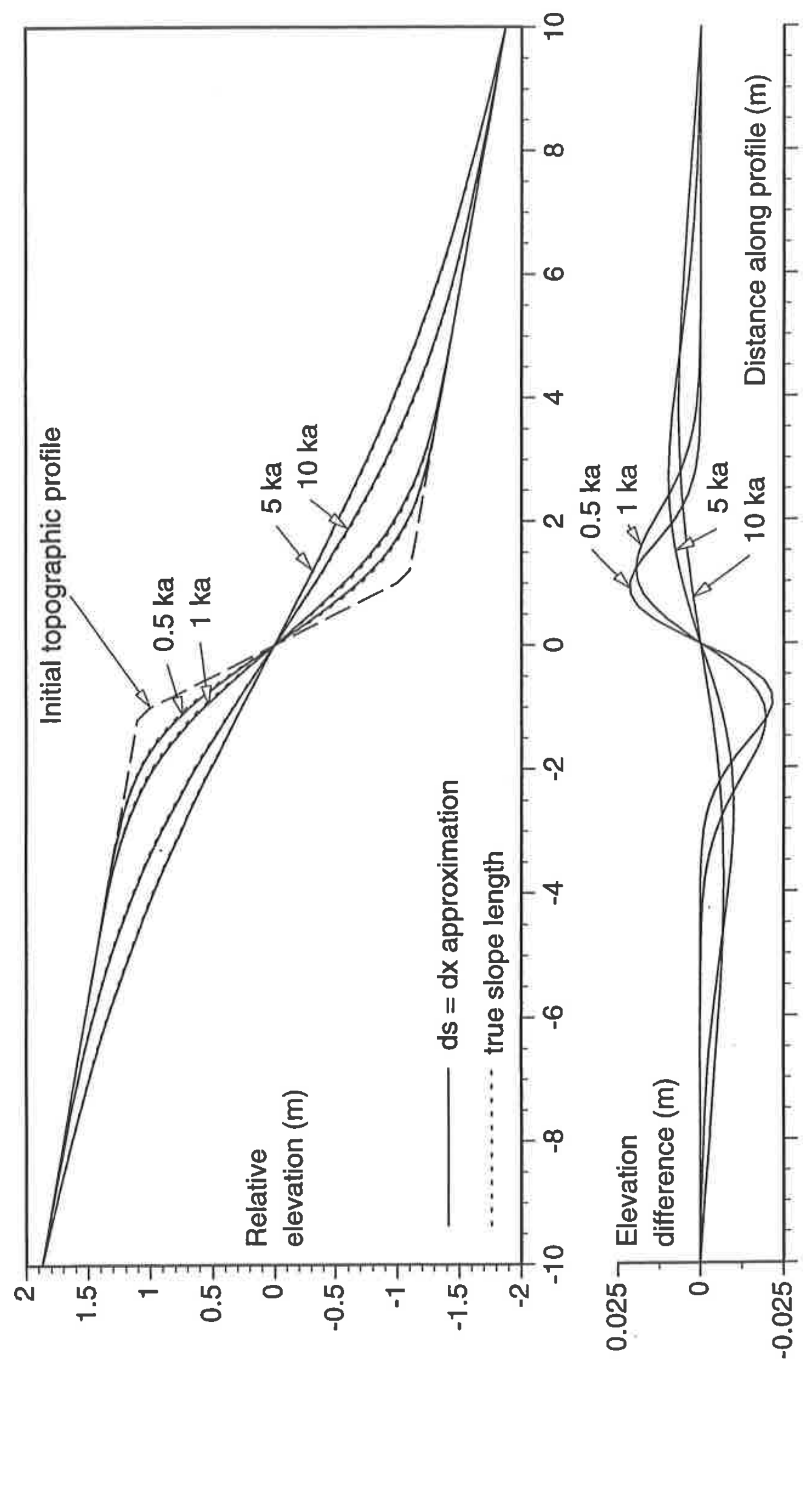


Figure A.9. Illustration of slope profiles at 0.5, 1, 5, and 10 ka (with $k = 1$ m^2/ka), with all other parameters the same, determined using explicit finite difference method and including slope length = horizontal length approximation (solid profiles), and true slope length (small dashing along profiles). The ramp initial shape is shown. The lower plot shows the elevation difference between the profiles for each final time. As expected, the true slope length cases do not erode as quickly; however, the difference is not great.

A.4.1. Sample computer code for numerical simulation of the true slope length case

In this section we present the sample routine that determines the changes in elevation with time based upon (A.4.10). It is a subroutine of the program DIFFUSE presented in Section A.5.

```
      Subroutine eulerDs (nx, size, data, dt, k, dx, nt)

C     Calculates new elevation vector by explicit centered
C     difference, keeping track of true slope length, ds

      integer i, l, nt, nx, size
      real data(size, 2), temp(1001, 2), k, dt, ds, dx

C     Calculate the change in elevation nt times
      do 12 l=1,nt

C     calculate the new elevations for this increment
      do 25 i=2,nx
      ds = sqrt(((2*dx)**2)+
     +      (data(i+1,2)-data(i-1,2))**2)/2

      temp(i,2)=data(i,2)+((dt*k)/(dx*ds))*
     +      (data(i+1,2)-(2*data(i,2))+data(i-1,2))

25          continue

C     Assign new elevation vector to elevation vector
      do 35 i= 2,nx
            data(i,2) = temp(i,2)
35          continue
12    continue
       END
C     end eulerDs
```

A.5. DIFFUSE user's guide

A.5.1 Documentation

The FORTRAN 77 program, DIFFUSE, determines topographic profile development of an arbitrary shaped initial profile, subject to constant elevation boundary conditions, no tectonic elevation changes, and specified final time, rate constant (k), and spatial step size. The temporal step size is determined automatically subject to the stability criterion that $\lambda < 0.25$. This criterion ensures that the results are stable--it is one-half the explicit stability criterion defined by *Ferziger* [1981]. This section of the document should help a user run DIFFUSE.

Diffuse was developed from 1991-1994 using Mathematica for the Macintosh. In August, 1994, it was translated to FORTRAN, and has successfully been compiled on pangea (DEC5800), shiprock (Sun Sparc 2), and bishop (SGI Indigo 2 xl).

Please contact with questions, bugs, or to get a copy of the program:

Ramón Arrowsmith
Department of Geological and Environmental Sciences
Leland Stanford Junior University
Stanford, CA 94305-2115
(415) 725-0573 office or (415) 497-4251 home

ramon@pangea.stanford.edu
ramon@keck.whittier.edu

DIFFUSE has only been tested on workstations operating under UNIX. To compile the text source code titled **diffuse.f**, type the following at the machine prompt:

```
pangea> f77 -o DIFFUSE diffuse.f
```

That should provide an executable program called **DIFFUSE**.

A.5.1.1 Input

DIFFUSE operates by reading and writing from the standard input/output. Prepare a properly formatted input file (**in**, see below), and type the following command using UNIX redirects at the machine prompt to have the output written to the file **out**:

```
pangea> DIFFUSE <in >out
```

A properly formatted input file should look like this:

```
'Ramp-shaped profile'
n,  nx, tf, k
5, 20, 1, 1
Initial profile
-10., 1.875
-1.09589, 1.09589
0, 0.
1.09589, -1.09589
10., -1.875
```

The program reads the first line to get the title. Note that the title (maximum length, 70 characters) must be surrounded by single quotes. It skips the second line, and then reads the third to define the user specified parameters: `n` is the number of elements in the initial profile and should agree with the number of lines following the line that says `Initial profile.` `nx` is the number of space steps desired in the analysis. DIFFUSE will determine a correspondingly stable number of time steps. `tf` is the final time, and `k` is the rate constant or "diffusivity." The fourth line is skipped, and the computer reads the next `n` lines as the initial, uninterpolated profile. Note that the units of space and time should be consistent. If the profile is in meters and the final time in ka, then `k` should have units of m^2/ka.

A.5.1.2 Output

The output file determined by DIFFUSE based upon the above input file looks like this:

```
Ramp-shaped profile

Model parameters
nx =           20
dx =    1.000000
nt =           41
dt =   0.2439024
k =    1.000000
Final time is    10.00000
lambda =   0.2439024

Crank-Nicholson nt =          11
Crank-Nicholson lambda =  0.9090909

                                Profiles
--------------------- ---------- ----------           ------------
      Initial,,           Euler,     Crank,           True slope length
--------------------- ---------- ----------           ------------
-10.00000,   1.87500,   1.87500,   1.87500,                1.87500
 -9.00000,   1.78750,   1.75616,   1.75647,                1.75752
 -8.00000,   1.70000,   1.63150,   1.63222,                1.63425
 -7.00000,   1.61250,   1.49541,   1.49670,                1.49960
 -6.00000,   1.52500,   1.34286,   1.34483,                1.34844
 -5.00000,   1.43750,   1.16990,   1.17251,                1.17658
 -4.00000,   1.35000,   0.97421,   0.97721,                0.98141
 -3.00000,   1.26250,   0.75567,   0.75863,                0.76251
 -2.00000,   1.17500,   0.51676,   0.51915,                0.52217
 -1.00000,   1.00000,   0.26251,   0.26385,                0.26551
  0.00000,   0.00000,   0.00000,   0.00000,                0.00000
  1.00000,  -1.00000,  -0.26251,  -0.26385,               -0.26551
  2.00000,  -1.17500,  -0.51676,  -0.51915,               -0.52217
  3.00000,  -1.26250,  -0.75567,  -0.75863,               -0.76251
  4.00000,  -1.35000,  -0.97421,  -0.97721,               -0.98141
  5.00000,  -1.43750,  -1.16990,  -1.17251,               -1.17658
  6.00000,  -1.52500,  -1.34286,  -1.34483,               -1.34844
  7.00000,  -1.61250,  -1.49541,  -1.49670,               -1.49960
  8.00000,  -1.70000,  -1.63150,  -1.63222,               -1.63425
  9.00000,  -1.78750,  -1.75616,  -1.75647,               -1.75752
 10.00000,  -1.87500,  -1.87500,  -1.87500,               -1.87500
```

The resulting output file is specially formatted to be pasted into or open by Microsoft Excel for manipulation and plotting. To get Microsoft Excel version 4 to recognize the commas as field separators, select Open from the File menu, and push the 'Text' button in the dialogue box. Then select comma as field separator. Then, open your text file or push cancel and then paste.

A.6 References

Atkinson, L. V., P. J. Harley and J. D. Hudson, *Numerical methods with fortran 77: a practical introduction*, 395 p., Addison-Wesley Publishing Company, Menlo Park, California, 1989.

Avouac, J. P., Analysis of scarp profiles: evaluation of errors in morphologic dating, *Journal of Geophysical Research, 98*, 6745-6754, 1993.

Carson, M. A. and M. J. Kirkby, *Hillslope form and process*, 475 p., Cambridge University Press, Cambridge, 1972.

Chapra, S. C. and P. C. Canale, *Numerical Methods for Engineers*, 812 p., McGraw-Hill, San Francisco, 1988.

Colman, S. M., Limits and constraints of the diffusion equation in modeling geological processes of scarp degradation, in *Directions in paleoseismology*, edited by A. J. Crone and E. M. Omdahl, pp. 311-316, U. S. Geological Survey Open File Report, 1987.

Culling, W. E. H., Soil creep and the development of hillside slopes, *Journal of Geology, 71*, 127-161, 1963.

Ferziger, J. H., *Numerical Methods for Engineering Application*, 270 p., Wiley, New York, 1981.

Hanks, T. C. and D. J. Andrews, Effect of far-field slope on morphologic dating of scarplike landforms, *Journal of Geophysical Research, 94*, 565-573, 1989.

Hanks, T. C., R. C. Bucknam, K. R. Lajoie and R. E. Wallace, Modification of wave-cut and fault-controlled landforms, *Journal of Geophysical Research, 89*, 5771-5790, 1984.

Hirano, M., A mathematical model of slope development: an approach to the analytical theory of erosional topography, *Journal of Geosciences, Osaka City University, 11*, 13-52, 1968.

Kirkby, M. J., Hillslope process-response models based upon the continuity equation, in *Slopes: form and process*, edited by D. Brunsden, pp. 15 - 30, Inst. Br. Geog. Spec. Pub., 1971.

Kirkby, M. J., P. S. Naden, T. P. Burt and D. P. Butcher, *Computer simulation in physical geography*, 180 p., John Wiley and Sons, New York, 1993.

Margenau, H., *The nature of physical reality*, 479 p., McGraw Hill, New York, 1950.

Metcalf, M., *Effective fortran 77*, 231 p., Oxford University Press, New York, 1989.

Mitchell, J. K., *Fundamentals of soil behavior*, 422 p., Wiley, New York, 1976.

Nash, D. B., Morphologic dating of degraded normal fault scarps, *Journal of Geology, 88*, 353-360, 1980.

Nash, D. B., Fault: a fortran program for modeling the degradation of active normal fault scarps, *Computers & geosciences, 7*, 249-266, 1981.

Selby, M. J., *Hillslope materials and processes*, 264 p., Oxford University Press, Oxford, 1982.

Selby, M. J., *Earth's changing surface*, 607 p., Clarendon Press, Oxford, 1985.

A.7 Program listing

```
      PROGRAM DIFFUSE

C     This program does diffusion erosion by euler, crank-
C       nicholson, and true slope length methods
C     The initial shape and parameters are specified in the
C     input file.  Constant elevation boundary and transport-limited
C     conditions are assumed

C     Program history:
C     Developed 1991-1994 using Mathematica for the Macintosh
C     Mon Aug  8 23:15:41 PDT 1994
C     Finished FORTRAN Translation
C     Successful compilation on pangea (DEC5800), shiprock (Sun
C       Sparc 2), and bishop (SGI Indigo 2 xl).

C     Variable definitions
C     nx --> Number of space steps
C     dx --> Size of space increment
C     nt --> Number of time steps
C     dt --> Size of time increment
C     tf --> Final time
C     maxi --> Maximum number of nodes (1001)
C     data --> Topographic profile
C     initial --> Initial topographic profile
C     eulerData --> Profile modified by euler method
C     crankData --> Profile modified by Crank-Nicholson method
C     eulerDSData --> Profile modified by true slope length method
C     k --> Rate constant ("Diffusivity")
C     lambda --> measure of stability

      integer nx, nt, maxi
            parameter (maxi=1001)
      real data(maxi, 2), initial(maxi, 2), dt, k, dx, lambda,
     +      eulerData(maxi,2), crankData(maxi,2), eulerDSData(maxi,2)

      call input  (nx, maxi, data, initial, dt, k, dx, tf, nt, lambda)

C     Define elevation vectors
      do 40 i=1,nx+1
            eulerData(i,1)=data(i,1)
```

```
             eulerData(i,2)=data(i,2)
             crankData(i,1)=data(i,1)
             crankData(i,2)=data(i,2)
             eulerDSData(i,1)=data(i,1)
             eulerDSData(i,2)=data(i,2)
40     continue

       call euler  (nx, maxi, eulerData, nt, lambda)

       call crank  (nx, maxi, crankData, k, dx, tf)

       call eulerDs(nx, maxi, eulerDSdata, dt, k, dx, nt)

       call output (nx, maxi, initial, eulerData,
      +      crankData, eulerDSData)

       END
C      end main program


C      Begin subroutines

       Subroutine euler (nx, size, data, nt, lambda)

C      Calculates new elevation vector by explicit centered
C      finite difference

C      i --> Time counter
C      l --> Space counter
C      size --> Long dimension of vector
C      temp --> Temporary elevation vector

       integer i, l, nt, nx, size
       real data(size,2), temp(1001,2), lambda

C      Calculate the change in elevation nt times
       do 10 l=1,nt

C            calculate the new elevations for this increment
             do 20 i=2,nx
                   temp(i,2)=data(i,2)+lambda*
      +                       (data(i+1,2)-(2*data(i,2))+data(i-1,2))
20           continue

C      Assign new elevation vector to elevation vector
             do 30 i= 2,nx
                   data(i,2) = temp(i,2)
30           continue
10     continue
       END
C      end euler


       Subroutine crank (nx, size, data, k, dx, tf)

C      This subroutine determines the new elevations by using the
C      Crank-Nicholson implicit scheme.
```

```
C     i --> Time counter
C     l --> Space counter
C     size --> Long dimension of vector
C     p --> Temporary elevation vector
C     cnlambda --> Crank-Nicholson lambda
C     rhs --> Right-hand side vector of Crank-Nicholson equation
C     band --> Vector for upper and lower bands of tridiagonal matrix
C     main --> Main band of tridiagonal matrix

      integer i, nt, l, size
      real data(size, 2), dx, k, tf, cnlambda, rhs(1001),
     +      band(1001), main(1001), dt, p(1001)

C     Determine new cnlambda for greater stability of CN method
C     Determine a stable nt and dx.  Note that the first dt is
C     provisional

      dt = (1*(dx**2))/k
      nt = int(tf/dt)+1
      dt = tf/nt
      cnlambda = (k*dt)/(dx**2)

      print *, 'Crank-Nicholson nt =', nt
      print *, 'Crank-Nicholson lambda =', cnlambda

C     Build bands for matrix
      do 51 i=1,nx-2
            band(i) = (-cnlambda)
51    continue

C     Build main band
      do 52 i=1,nx-1
            main(i) = 2*(1 + cnlambda)
52    continue

C     Calculate elevation change nt times
      do 53  l=1,nt
            call vctr (rhs, data, cnlambda, nx, size)
            call trdiag (nx-1, band, main, band, p, rhs, size)
            do 55 i=2,nx
                  data(i,2)=p(i-1)
55          continue
53    continue
      END
C     end crank


      Subroutine vctr (rhs, data, cnlambda, nx, size)

C     The subroutine vctr determines the value of the
C     vector rhs for the tridiagonal solution of the
C     matrix equation of the crank-nicholson scheme:

      integer nx, size
      real data(size, 2), cnlambda, rhs(size)

C     update first interior node
       rhs(1) = 2*cnlambda*data(1,2) + 2*(1-cnlambda)*data(2,2)
     +            + cnlambda*data(3,2)

C     update fully interior nodes
       do 54 i=3,nx-1
                  rhs(i-1) = cnlambda*(data(i-1,2) + data(i+1,2)) +
     +            2*(1-cnlambda)*data(i,2)
54    continue

C     update last interior node
       rhs(nx-1) = cnlambda*data(nx-1,2) +
     +            2*(1-cnlambda)*data(nx,2) + 2*cnlambda*data(nx+1,2)

       END
C     end vctr


      Subroutine eulerDs (nx, size, data, dt, k, dx, nt)

C     Calculates new elevation vector by explicit centered
C     difference, keeping track of true slope length, ds

      integer i, l, nt, nx, size
      real data(size, 2), temp(1001, 2), k, dt, ds, dx

C     Calculate the change in elevation nt times
      do 12 l=1,nt

C     calculate the new elevations for this increment
      do 25 i=2,nx
      ds = sqrt(((2*dx)**2)+
     +      (data(i+1,2)-data(i-1,2))**2)/2

      temp(i,2)=data(i,2)+((dt*k)/(dx*ds))*
     +      (data(i+1,2)-(2*data(i,2))+data(i-1,2))

25          continue

C     Assign new elevation vector to elevation vector
      do 35 i= 2,nx
            data(i,2) = temp(i,2)
35          continue
12    continue
       END
C     end eulerDs


      Subroutine input (nx, size, data, initial, dt, k, dx, tf, nt,
+           lambda)

C     It needs an input file with the following format:
C     'title' of maximum length, 70 characters
C     Note that the title must be surrounded by single quotes
C     number of elements in initial profile, nx, tf, k
C     Uninterpolated initial profile
C     data: xmin, H0
C     data: x2, H2
C     :
C     data: xnx, Hnx
```

```
C     data: xmax, HF
C
C     Below is sample input file:
C     'Ramp-shaped profile'
C     n,  nx, tf, k
C     5, 100, 1, 1
C     Initial profile
C     -10., 1.875
C     -1.09589, 1.09589
C     0, 0.
C     1.09589, -1.09589
C     10., -1.875

C     prof is the initial, uninterpolated profile

      integer i, j, n, nx, nt, size
      real data(size,2), prof(1001,2), initial(size,2), dx, k,
     +      lambda, dt, tf
      character title*70

      read *, title
      read *
      read *, n, nx, tf, k
      read *
      read *, ((prof(i,j), j=1,2), i=1,n)

      call interpolate (data, prof, size, n, nx, dx)

C     determine a stable nt and dx.  Note that the first dt is
C     provisional
      dt = (0.25*(dx**2))/k
      nt = int(tf/dt)+1
      dt = tf/nt
      lambda = (k*dt)/(dx**2)

      print *, title
      print *
C     Print input parameters
      print*, 'Model parameters'
      print *, 'nx = ', nx
      print *, 'dx = ', dx
      print *, 'nt = ', nt
      print *, 'dt = ', dt
      print *, 'k = ', k
      print *, 'Final time is ', tf
      Print *, 'lambda = ', lambda

C     Determine initial profile
      do 5 i=1,nx+1
            initial(i,1) = data(i,1)
            initial(i,2) = data(i,2)
5     continue
      print *
      END
C     end input
```

```
      Subroutine interpolate (data, prof, size, n, nx, dx)

C     This subroutine takes a set of x, H points of arbitrary spacing
C     but sequential order, prof (i,j), their number, n, and the desired
C     number of space steps, nx, and returns data (i,j) of regularly
C     spaced, linearly interpolated points and a new dx.

C     loc --> Index of profile
C     proflength --> Profile length
C     dist --> Distance from current data vector node to current prof
C       node
C     slope --> Local slope

      integer loc, i, n, nx, size
      real data(size, 2), prof(size, 2), proflength, dx, dist, slope

C     Determine dx
      proflength = Abs(prof(n,1)-prof(1,1))
      dx = proflength/nx

C     upper end
      data(1,1) = prof(1,1)
      data(1,2) = prof(1,2)

C     interpolate

      loc=2
      do 50 i=2,nx
            data(i,1) = data(1,1)+(i-1)*dx
            if (data(i,1).gt.prof(loc,1)) loc=loc+1
            if (data(i,1).lt.prof(loc,1)) then
                  slope=((prof(loc-1,2)-prof(loc,2))/
     +                   (prof(loc-1,1)-prof(loc,1)))
                  dist=data(i,1)-prof(loc-1,1)
                  data(i,2) = dist*slope + prof(loc-1,2)
            end if
            if (data(i,1).eq.prof(loc,1)) then
                  data(i,2)=prof(i,2)
                  loc=loc+1
            end if
50    continue

C     lower end
      data(nx+1,1) = prof(n,1)
      data(nx+1,2) = prof(n,2)

      END
C     end interpolate


      subroutine output (nx, size, initial, eulerData,
     +      crankData, eulerDSData)

      integer i, j, nx, size
      real eulerData(size,2), crankData(size,2), eulerDSData(size,2),
     +      initial(size,2), all(1001,5)
```

```
C     First, put all of the data into one array (all) so it
C     will write properly

      do 475 i=1,nx+1
            all(i,1) = initial(i,1)
            all(i,2) = initial(i,2)
            all(i,3) = eulerData(i,2)
            all(i,4) = crankData(i,2)
            all(i,5) = eulerDSData(i,2)
475   continue

C     Some of thie funky formatting below is desinged for easy
C     cut and paste of output into Microsoft Excel or some
C     other plotting package

      Print *
      Print *, '                        Profiles'
      Print *, '---------------------- ----------- -----------
     +   -------------'
      Print *, '  Initial,,      Euler,     Crank,
     +        True slope length'

      Print *, '---------------------- ----------- -----------
     +   -------------'

      write(*,1500), ((all(i,j),j=1,5),i=1,nx+1)

1500  format (f10.5, ',' f10.5, ',', f10.5, ',', f10.5, ',
     +      ', f10.5)

      END
C     end output


      Subroutine trdiag (n, a, b, c, x, g, size)

C     This subroutine solves tridiagonal systems of equations
C     by Gauss elimination.
C     The problem solved is mx=g where m=tri(a,b,c)
C     This routine does not destroy the original matrix
C     and may be called a number of times without redefining the
C     matrix.
C     Modified from Ferziger, 1981.
C     n = number of equations to be solved
C     t = multiplier

      integer i, j, n, size
      real a(size), b(size), c(size), x(size), g(size), bb(1001),
     +      t

C     forward elimination
C     bb is a scratch array needed to avoid destroying b array

      do 1 i=1,n
              bb(i) = b(i)
1     continue

      do 2 i=2,n
```

```
                t = a(i-1)/bb(i-1)
                bb(i) = bb(i) - c(i-1)*t
                g(i) = g(i) - g(i-1)*t
2       continue

C       back substitution
        x(n) = g(n)/bb(n)
        do 3 i=1,n-1
                j=n-i
                x(j) = (g(j)-c(j)*x(j+1))/bb(j)
3       continue

        END
C       end trdiag
```