2-Dimensional Transport and Production Limited Analysis of Fault Scarps:

Landlab Implementation and Examples from Western US

by

Abdel Hafiz

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2023 by the
Graduate Supervisory Committee:

Ramon Arrowsmith, Chair
Kelin Whipple
Chelsea Scott

ARIZONA STATE UNIVERSITY

May 2023

ABSTRACT

This study presents an analysis of fault scarps, with a focus on implementing the Landlab computational toolkit to model fault scarp evolution and analyzing fault scarps under transport and production-limited conditions with linear and nonlinear diffusive transport laws. The aim of the study is to expand diffusion modeling of fault scarps from 1D to 2D by using Landlab toolkit.

The study evaluated two fault scarps in western US (NE California): one representing an old fault scarp (Twin Butte) and the other representing a young fault scarp (Active Hat Creek Fault). High-resolution digital elevation models (DEMs) were used to generate 2D surfaces of the fault scarps, which were then converted to 1D profiles for morphological modeling and analysis. The accuracy of the models was evaluated using Root Mean Squared Error (RMSE), and the best-fit models were selected for further examination.

The grid search of the non-linear diffusion model of the Twin Butte and Active Hat Creek fault scarps showed optimum values for transport constant (k) and scarp age (t) that aligned with the apparent ages of the rocks and associated fault scarps. For both fault scarps, the optimum k value was around 7.5 m^2/kyr, while the optimum t value was around 110 kyr for the Twin Butte scarp and around 26 kyr for the Active Hat Creek scarp. The results suggest that the geomorphic processes (influenced by climate and rock types) in both fault scarps are similar, despite the difference in age and location. Integrating tectonic displacement in the model helps to better capture the observed patterns of tectonic deformation.

The expansion of the fault scarps diffusion model from 1D to 2D opens up a range of fascinating possibilities, as it enables us to model the lateral movement of particles that the 1D model typically overlooks. By incorporating this additional dimension, we can better understand the complex interplay between vertical and horizontal displacements, providing a more accurate representation of the geological processes at work. This advancement ultimately allows for a more comprehensive analysis of fault scarps and their development over time, enhancing our understanding of Earth's dynamic crustal movements.

ACKNOWLEDGMENTS

First and foremost, I would like to extend my heartfelt thanks to the Indonesia Endowment Fund for Education (LPDP) for providing the financial support for my master's degree. Your investment in my education has been crucial to the successful completion of this thesis.

I am also immensely grateful to Pacific Gas and Electric (PG&E) for their generous support during the field trips, which allowed me to collect essential data for this research. Their assistance in making these trips possible has been invaluable in advancing my understanding of the subject matter.

I would like to express my sincere appreciation to Dr. Ramon Arrowsmith, my committee chair, for his continuous guidance, insightful feedback, and unwavering dedication to my academic growth. His expertise and wisdom have been vital in shaping my research and understanding of the field.

My deep gratitude goes to my committee members, Dr. Kelin Whipple and Dr. Chelsea Scott, for their invaluable input, suggestions, and constructive criticism throughout the thesis development process. Their commitment to academic excellence has been inspiring, and their contributions have significantly improved the quality of my research.

Lastly, I would like to extend my appreciation to my friends, family, and colleagues for their support, encouragement, and understanding throughout this challenging but rewarding journey. Without you, this accomplishment would not have been possible.

Thank you all for playing such significant roles in my academic and personal growth.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1. Motivation

The study of fault scarps is critical in understanding tectonic activity and earthquake hazards in regions with active tectonics. The importance of fault scarps is mainly due to their association with faults, meaning that it could potentially pose earthquake hazards if said faults were further displaced after its last movement that generated the fault scarps. Fault scarps are landforms that develop where the topography on one side of a fault has moved vertically with respect to the other side. By studying the characteristics of fault scarps, such as their height, length, and morphology, we can infer information about the behavior of active faults, such as the amount and rate of slip, and the recurrence interval of the causative earthquakes.

Fault scarps are usually classified based on the materials that they cut. If the fault cuts into loose earthy materials, we referred to it as alluvial scarp (e.g., Hanks, 2000; Hanks et al., 1984). Conversely, if a fault cut into hard rocks (also known as regolith or bedrock), the fault scarp that was generated from it would be referred to as bedrock scarp (e.g., Arrowsmith et al., 1996), or fault-plane-exposed scarp if the fault plane was exposed (e.g., Scott, 2020). This division is useful but incomplete, because it does not consider the surface evolution of fault scarp over a long time, where a bedrock scarp can look like alluvial scarp given enough time, such as the case in Arrowsmith et al. (1996)

There are several methods used to analyze fault scarps and estimate the rate of tectonic uplift or subsidence, earthquake recurrence intervals, and seismic hazard. Here are a few examples:

- Cosmogenic nuclide dating: Cosmogenic nuclides are isotopes produced by cosmic ray interactions with minerals at the Earth's surface. By measuring the concentration of cosmogenic nuclides in rocks exposed on a fault scarp, it is possible to estimate the time since the rocks were last uplifted or exposed to the surface. Cosmogenic nuclide dating has been used to estimate slip rates and earthquake recurrence intervals on a variety of fault systems, including the San Andreas Fault in California and the Altyn Tagh Fault in Tibet (Bull, 1996; Mériaux et al., 2012)
- Diffusion Modeling: This method involves modeling the evolution of fault scarps over time by considering the rate of sediment transport and deposition/erosion. By comparing the shape of a fault scarp at the present day with its theoretical shape predicted by the diffusion model, it is possible to estimate the fault slip rate and earthquake recurrence interval (with properly calibrated rate constants). The diffusion model has been used to analyze fault scarps in a variety of settings (e.g., Arrowsmith et al., 1996; Hanks et al., 1984; Nash, 1980)

These methods, along with others such as trenching and paleoseismology, provide valuable insights into the behavior of active faults and the potential earthquake hazards they pose. Diffusion modeling offers substantial potential as an approach to fault scarp analysis due to its accessibility. Unlike methods such as cosmogenic nuclide dating and fault trenching that require specialized equipment, diffusion modeling relies on standard software packages, making it more widely available. Results can be obtained relatively quickly, further enhancing its appeal. However, it is important to consider that diffusion modeling is sensitive to uncertainties in sediment transport, erosion parameters, and assumptions about the equilibrium between uplift and erosion. Despite these limitations,

the overall potential and accessibility of diffusion modeling in various fault scarp scenarios remain highly valuable.

Given the advantages and limitations of diffusion modeling, it is important to continue to refine and improve the method, and to compare the results with those obtained from other methods, in order to better understand the behavior of active faults and the seismic hazard they pose. Further research on diffusion modeling and other fault scarp analysis methods can help improve our understanding of the mechanics of tectonic deformation and mitigate the impact of earthquakes on society.

2. Research Aims

The aim of this thesis is to expand the current 1D (profile-based) diffusion modeling of fault scarps to 2D (raster or map-based). Fault scarps are an important indicator of tectonic activity and can provide valuable information about the rates and patterns of fault slip, which are critical for earthquake hazard assessment and mitigation. However, accurately modeling fault scarps and their evolution over time can be challenging due to the complex interplay between tectonic forces, erosion, and sediment transport.

In this study, I will use Landlab toolkit to expand fault scarps diffusion model from 1D to 2D, taking into account the effects of tectonic displacement. Expanding the fault scarps diffusion model from 1D to 2D offers several advantages, allowing for a more comprehensive understanding of complex geological processes. One key benefit is the ability to account for convergence and divergence of sediments on fault scarps, which would otherwise be neglected in a 1D model. By incorporating these phenomena, the 2D model can provide a more accurate representation of the spatial distribution and movement of sediments, leading to a deeper understanding of fault scarp development and evolution.

The transition to a 2D model opens up new avenues for investigating fault scarps and offers valuable insights into the intricacies of Earth's dynamic crustal movements.

The ultimate goal of this research is to develop a more comprehensive approach for modeling fault scarps and their evolution over time, which can help improve our understanding of tectonic deformation and earthquake hazards. The results of this study can be used to inform earthquake risk assessments and hazard mitigation strategies, and can also contribute to the development of more advanced models of fault behavior and seismicity.

3. Background Knowledge

Landscape diffusion modeling is a method used to study and understand the processes that shape the Earth's surface over time (e.g., Culling, 1960; Roering, 2008 and many more). It is an approach that combines mathematical models, field observations, and empirical data to predict and describe the evolution of landscapes in response to erosion, deposition, and other geomorphic processes. Two important concepts in landscape diffusion modeling are transport-limited conditions and production-limited conditions. These terms describe different states that affect how landforms, particularly hillslopes or escarpments (scarps), evolve over time.

Transport-limited scarps, refers to a landscape where the rate of erosion and sediment transport is the primary limiting factor in the evolution of a hillslope or escarpment (Whipple & Tucker, 2002). In this case, the amount of material that can be moved downslope is constrained by the transport capacity of the geomorphic processes involved, such as fluvial, glacial, or gravitational processes (Anderson & Anderson, 2010). This limitation results in a slower rate of hillslope or escarpment retreat, leading to a more

gradual and smoother landscape morphology (Dietrich et al., 2003). In transport-limited systems, an increase in the efficiency of sediment transport, such as through changes in climate or tectonic activity, could accelerate the rate of landscape evolution (England & Molnar, 1990)

On the other hand, a production-limited scarp is characterized by a landscape where the generation of erodible material at the scarp face is the main limiting factor in its evolution (Sklar & Dietrich, 2001). In this scenario, the rate at which the hillslope or escarpment retreats is controlled by the rate of weathering and the production of sediment (Riebe et al., 2001). The transport capacity of the geomorphic processes may be greater than the amount of material produced, leading to a more abrupt and steeper scarp morphology (Hurst et al., 2012). In production-limited systems, changes in climate or other factors that affect weathering rates can significantly influence the pace of landscape evolution (Gabet et al., 2003).

Both transport-limited and production-limited scarps are important concepts in landscape diffusion modeling because they provide insight into the dominant processes shaping a given landscape (Tucker & Slingerland, 1997). By identifying whether a hillslope or escarpment is transport-limited or production-limited, we can better understand the factors controlling landscape evolution and make more accurate predictions about future changes (Roering et al., 2007). Moreover, analyzing the distribution and characteristics of these different scarp types can help to reconstruct past climatic and tectonic conditions, providing valuable information about the Earth's history and the factors that have shaped its surface.

CHAPTER 2

LANDSCAPE EVOLUTION MODEL FOR FAULT SCARPS IN TWO DIMENSIONS

1. Introduction

The development of earth's surface in area of active tectonics is mostly controlled by tectonic and geomorphic displacements, thus a landscape evolution model addressing such regions should weight both processes accordingly. My focus here is on the 10s to 100s meter length scale faulted landscapes and the development of fault scarps. Topographic analysis of fault scarps has been done before in Arrowsmith et al. (1996 & 1998), Hanks et al. (1984), Nash (1980), and many others, using a 1D (profile-based) linear diffusion model to simulate both tectonic and geomorphic displacement of fault scarps. The 1D approach described in Arrowsmith et al. (1996 &1998), was especially powerful when examining small-scale active tectonics features like fault scarps, but it still falls short when used to examine scarps with a complex fault system, because the model only simulates the movement of materials in one-dimension, ignoring any lateral mass transport that might affect scarp topography.

Topographic analysis using diffusion model are based on the law of conservation of masses. The change in elevation over time in a simple landscape evolution model can be expressed in continuity equation below:

$$\frac{\Delta z}{\Delta t} = \nabla q_s \tag{2.1}$$

The sediment flux (q_s) in (2.1.) can be obtained using transport law equation below:

$$q_s = -kA^m S^n \tag{2.2}$$

where q_s is a function of sediment flux, k is transport constant in [L^2T^{-1}], A is distance from the divide in [L] raised to the power of m, and S is local slope in [L/L] raised to the power of n. In hillslope condition where the main mode of erosion comes from rain splash and creep (m = 0 and n = 1), the equation can be simplified into:

$$q_s = -kS \tag{2.3}$$

The function of sediment flux expressed in (2.3.) results in linear model where q_s increases linearly with slope, as shown in Figure 2.1. While we assumed that rain splash and soil creep were the main factors in hillslope development, resulting in a relationship shown in (2.3), other processes such as animal induced disturbances and chemical weathering also contribute. All these processes that modulate the rate of Earth's surface change over time were simplified into a rate constant called k in (2.3).

The sediment flux expressed in (2.3) works in hillslope condition with unconsolidated materials and uniform fine grain size. In a condition where the landscape being modeled consists of consolidated materials or otherwise deviate from linear dependence on S, the critical slope Sc must be included in sediment flux calculation as follow:

$$q_s = -kS\left(1 + \left(\frac{S}{S_c}\right)^2 + \left(\frac{S}{S_c}\right)^4 + \cdots \left(\frac{S}{S_c}\right)^{2(N-1)}\right) \tag{2.4}$$

Equation (2.4) uses Taylor Series expansion derived by Ganti et al. (2012) to represent the rapid increase in q_s when S approaches Sc.

The general solution use to calculate the change in elevation dz over time step dt when considering uplift rate U is:

$$\frac{dz}{dt} = U - \nabla q_s \tag{2.5}$$

∇q_s is the continuity equation for sediment transport balancing influx to a hillslope element with outflux and increase or decrease of mass (and thus elevation).

In this chapter, I have explored 2D solution of fault scarp evolution using diffusion and weathering modules developed by Barnhart et al. (2019) that can be found in landlab. Landlab is a Python-based toolkit for modeling earth surface processes, such as erosion, sediment transport, and landscape evolution. It provides a set of modular components that can be combined to create a wide range of models, from simple one-dimensional systems to complex three-dimensional landscapes.

The models explored in this chapter are the idealized version of fault scarp evolution with the assumption that run-off was insignificant to the system, thus changes on the scarp surfaces highly dependent on k (local erosion rates; controlled by particle size, shape, and cohesion), and w (local weathering rates; controlled by bedrock density and soil thickness). From the simulation that had been done in this chapter, it was evident that landlab is a powerful toolkit that can be used to simulate a wide range of processes controlling fault scarp development, including both tectonic and geomorphic ones.

2. Landscape Evolution Models for Fault Scarps

2.1. Transport vs. Production-Limited Conditions

Many factors control how fault scarps and other landscape elements evolve over time. Here we divide them into two broad categories: transport-limited, and production-limited. These categories were based on the limitations that control the availability of transportable material and thus morphological changes in a scarp.

In transport-limited case, the scarp has sufficient transportable material. The forces that transport these materials downslope with dependence on local slope are many: rain

splash, animal induced disturbances, fallen trees, etc. All these phenomena affect mass transport and the topography of a scarp. Because it is impossible to consider every single force that affects a scarp, it is more practical to compound all these phenomena into a constant that represents the average erosion rate of that scarp (k). We may calibrate this constant from well-constrained scarps with known ages and use it to date other scarps around it, assuming the morphological process do not differ significantly. This approach had been done many times and proven to be reasonable, as shown in Arrowsmith et al. (1996 & 1998), Hanks et al. (1984), Nash (1980), Xu et al. (2021) and many others.

In the production-limited case, availability of transportable material is limited—it must be produced from the underlying bedrock. The potential mass change in the hillslope element indicated by continuity is limited by the local and uphill availability of material. The transport processes are similar to those described above for the transport-limited case; just limited by availability of material. This concept is explained in Anderson & Humphrey (1989) and Arrowsmith et al. (1996) for example as seen in Figure 2.2.

The change in elevation over time is shown for both production-limited and transport-limited scarps in Figure 2.3. The first two graphs illustrate how the differences in the ratio of bedrock and soil densities affect the final elevations of production-limited scarp. When the ratio between bedrock density and soil density are not equal to one, the weathered bedrock would have a different volume from the soil produced from it.

There are two types of weathering that directly influence the rate of soil production in a regolith, and these are physical and chemical weathering. Physical weathering involves the mechanical breakdown of rocks into smaller fragments due to the action of physical

forces like wind, water, and temperature changes. On the other hand, chemical weathering involves the chemical breakdown of rocks through the reactions of minerals with atmospheric gases, such as oxygen and carbon dioxide. Physical weathering can create fresh surfaces for chemical weathering to occur, while chemical weathering can transform rocks into new minerals and soil components. Both physical and chemical weathering are essential in soil production, as they contribute to the breakdown of rocks and the formation of soil.

To account for the production of transportable material, we assume (like many others (e.g., Anderson & Humphrey, 1989; Heimsath et al., 1997, 2012) that mechanical or physical weathering produces a maximum rate of conversion of rock to regolith at a zero soil thickness (we do not account for chemical weathering which results in a humped soil production function as seen in Dixon & von Blanckenburg, 2012). The sensitivity for soil production to soil thickness (H) is assumed to be inversely exponential.

Along with the soil production sensitivity to soil production decay depth *Hp*, recent models for soil transport account for a soil transport decay depth *Ht*. The equation for sediment flux in linear slope dependence hillslopes was derived from Johnstone & Hilley (2015), as described below:

$$q_s = -KSH_t\,(1 - e^{-\frac{H}{H_t}}) \tag{2.6}$$

The equation for non-linear depth-dependent sediment transport is similar to (2.6), except for the fact that there was a Taylor Series expansion to represent the nonlinearity in slope dependence. The non-linear depth-dependent transport law derived from Ganti et al. (2012), and Johnstone & Hilley (2015) is:

$$q_s = -KSH_t\left(1 + \left(\frac{S}{S_c}\right)^2 + \left(\frac{S}{S_c}\right)^4 + \cdots \left(\frac{S}{S_c}\right)^{2(N-1)}\right)\left(1 - e^{-\frac{H}{H_t}}\right) \quad (2.7)$$

Soil production rate [LT^{-1}] for a depth-dependent hillslope can be expressed as:

$$w = \frac{\rho_{br}}{\rho_s}\sqrt{(Sw_0)^2 + (w_0)^2}\, e^{-\frac{H}{H_p}} \quad (2.8)$$

Where w_0 is the maximum soil production rate at zero soil thickness. Finally, the change in elevation over time for production-limited scarp is:

$$\frac{dz}{dt} = U - \nabla q_s + \left(\frac{\rho_{br}}{\rho_s} - 1\right)\sqrt{(Sw_0)^2 + (w_0)^2}\, e^{-\frac{H}{H_p}} \quad (2.9)$$

Where ρ_{br} is bedrock's density and ρ_s is soil density.

2.2. 1D vs. 2D Implementation

Modeling hillslope development using diffusion models has been done many times (e.g., Arrowsmith et al., 1996; Hanks, 2000; Roering, 2008; Roering et al., 1999; Xu et al., 2021), but most of them was done in 1D, making it harder to examine a complex fault scarp. There are a few methods that can be used to get a better result from 1D diffusion model, such as fitting the model with swath from DEM, taking multiple profiles to measure, and adjusting internal parameters like pixel size and timestep. However, these methods require human intervention for it to work properly, thus making it harder to reproduce.

In contrast, adding the second horizontal dimension to make it 2D diffusion can enhance the analysis, especially in the case of convergent and divergent flow. In 1D, the movement of particles is restricted in one general direction, whereas in 2D the particles do not have the same restriction. 2D diffusion models also allow a more complex fault trace to be analyzed.

Figure 2.4 effectively demonstrates the advantages of utilizing a 2D diffusion model over a 1D model, particularly in the context of convergence and divergence of sediment on a fault scarp with an undulated fault surface. The figure visually presents how the 2D model captures the spatial variations in sediment transport, effectively representing areas where sediment accumulates (convergence) and areas where it disperses (divergence) along the irregular fault surface. In contrast, a 1D model would fail to adequately represent these complexities, offering only a simplified understanding of the fault scarp's behavior.

Despite its potential, 2D diffusion model requires more computing, especially if the initial model is a complex fault system. This means that in some cases, one would prefer 1D diffusion model over 2D one, despite the obvious advantages of 2D diffusion model. The simplicity of 1D model also make it easier to analyze and in some cases, one can even do it directly on the field, as shown in Hanks (2000).

2.3. 2D Landscape Evolution Modeling Using Landlab

Landlab is an open-source Python package used to model Earth surface processes (Barnhart et al., 2019; Hobley et al., 2017). It provides a framework for building and running models of landscapes and their evolution over time, including processes such as erosion, sediment transport, and tectonic uplift. It models the landscape as a grid of cells, with each cell representing a portion of the surface. The cells can be connected to their neighbors in various ways, such as a regular grid, a Voronoi diagram, or a Delaunay triangulation. This connectivity allows for the simulation of diffusion-like processes, such as hillslope erosion and sediment transport.

In this chapter, we used four landlab components related to diffusion, one component related to weathering, and one component related to fault displacement.

LinearDiffuser, *TaylorNonLinearDiffuser*, *DepthDependentDiffuser*, and *DepthDependentTaylorDiffuser* are four components in landlab that simulate hillslope erosion and soil transport that are useful for fault scarp analysis and for which the underlying process formulations were described above. Each component has its own set of assumptions and implementations, making them suitable for increasingly complex applications. In this section, I present the essential snippets of code that were used (the appendix contains the entire set of Jupyter notebooks that I developed).

To import the components mentioned before, I used the following python commands:

```
from landlab.components import LinearDiffuser
from landlab.components import TaylorNonLinearDiffuser
from landlab.components import DepthDependentDiffuser
from landlab.components import DepthDependentTaylorDiffuser
```

LinearDiffuser or *TaylorNonLinearDiffuser* can be used to simulate transport-limited conditions while *DepthDependentDiffuser* or *DepthDependentTaylorDiffuser* can be used to simulate production-limited hillslopes. When simulating a production-limited scarp, we import landlab component called *ExponentialWeatheringIntegrated* to simulate soil production from bedrock. To import this module, we used the following python command:

```
from landlab.components import ExponentialWeathererIntegrated
```

Because we integrate tectonics and geomorphic displacement in our model, we use another landlab component called *NormalFault* to simulate vertical fault displacement. We can import this component by using the following command:

```
from landlab.components import NormalFault
```

After importing the core modules needed to make our model, we create a grid object using the *RasterModelGrid* class in landlab. This command specifies the number of rows and columns, the grid spacing, and other parameters as needed.

```
from landlab import RasterModelGrid
grid = RasterModelGrid((nrows, ncols), xy_spacing)
```

Next, we add layers to our previous grid. We can do this by setting up the field that we want to add. The following command create the layers that we used in our model:

```
z = grid.add_zeros("topographic__elevation",at="node")
BRz = grid.add_zeros("bedrock__elevation",at="node")
H = grid.add_zeros("soil__depth",at="node")
w = grid.add_zeros("soil_production__rate",at="node")
#z is elevation at ground surface
#BRz is elevation at soil-bedrock interface
#H is soil depth
#w is soil production rate
```

After setting up the layers to our grid, I then set up initial parameters to build our initial model. Table 2.1. shows the parameters used in my model.

After setting up the parameters for the model, I select and initiate the diffusion module that we used in our model, as shown in following commands:

```
def diffuser_mode(mg,k,Ht = 0.12,Sc = 1):
    ld = LinearDiffuser(mg, linear_diffusivity=k)
    td = TaylorNonLinearDiffuser(mg,linear_diffusivity = k,
                                 slope_crit = Sc, dynamic_dt=True)
    ddld = DepthDependentDiffuser(mg,linear_diffusivity = k,
                                  soil_transport_decay_depth=Ht)
    ddtd = DepthDependentTaylorDiffuser(mg,soil_transport_velocity=k,
          slope_crit=Sc,soil_transport_decay_depth=Ht,dynamic_dt=True)
    return [ld,td,ddld,ddtd]
#call diffuser_mode function
diffuser = diffuser_mode(model,k)
eroder = diffuser[i] #pick diffuser mode that you want to use
#[0]LinearDiffuser[1]TaylorNonlinearDiffuser[2]DepthDependentDiffuser
#[3]DepthDependentTaylorDiffuser
```

For production-limited cases, I use this additional command to set up soil production rate (see (2.8.) and (2.9.)) to our model grid, as shown below:

```
expweath = ExponentialWeathererIntegrated(grid,
soil_production__maximum_rate=w0, soil_production__decay_depth=Hp)
```

After setting up modules needed for geomorphic displacement, I calculate tectonic displacement in the model using the following command:

```
nf = NormalFault(grid, faulted_surface, fault_throw_rate_through_time,
                 fault_trace)
```

I ran the model using three different displacement modes as seen in Figure 2.5. I use *run_one_step* method in landlab to run the model for each timestep. Different series of for loops run the model in different fault displacement modes. The following command runs our model in single earthquake mode (single vertical offset at time zero):

```
nf.run_one_step(total_time)
for i in range(nt):
     expweath.run_one_step(dt) #initiate bedrock weathering
     eroder.run_one_step(dt)   #initiate diffusion
```

The next command was used to run our model in multiple earthquakes mode:

```
for eq in range(len(time_eqs)-1):
     dtf = time_eqs[i+1]- time_eqs[i] #time step of faulting
     nt = int(dtf/dt) #the number of time step ran for one faulting
                      #event
     nf.run_one_step(dtf)
     for t in range(nt):
          expweath.run_one_step(dt) #used in depth dependent case.
          eroder.run_one_step(dt)
```

The following command was used to run continuous slip mode (steady offset constant for each time step to accumulate the required total vertical offset):

```
for i in range(nt):
     nf.run_one_step(dt)
     expweath.run_one_step(dt) #used in depth dependent case.
     eroder.run_one_step(dt)
```

The final model that had been diffused can be saved into various formats, and here I save it to esri ascii format using this command:

```
from landlab.io import write_esri_ascii
exported_raster = write_esri_ascii(path, grid)
```

3. Results

3.1. Transport-Limited Scarp

Using parameters listed in Table 2.1., I simulated geomorphic and tectonic processes that shape fault scarps over time. The simulations were done using the linear

diffusion and non-linear diffusion modules from landlab and were run separately based on its diffusion and displacement mode, as shown in Figures 2.6 and 2.7.

In general, there was no significant difference between linear diffusion and non-linear diffusion when we applied it to our three-meter scarp. The greatest sensitivity in response is early in the history of the scarp. The topographic difference between linear diffusion and non-linear diffusion is shown in Figure 2.8. As more time passed and more displacements occur, linear and non-linear diffusion models become more identical to one another. Linear diffusion is thus adequate for many situations to model transport-limited fault scarps in area of active tectonics.

3.1.1. Linear Diffusion

Figure 2.6 shows the resulted model from our simulation using linear diffusion in three displacement mode. The resulting topography is similar across different displacement modes. However, looking closer at the shape of the scarp of the same age with different displacement modes, we found that the slope is steeper with displacements events across the scarp. In contrast, the relationship between slope and time was the opposite, where the slope grew gentler as more time passed.

3.1.2. Non-Linear Diffusion

Figure 2.7 shows the resulting model from the simulation using non-linear diffusion in three displacement mode. The resulting topography from this diffusion mode is similar to linear diffusion mode in our model. The relationship between slope vs. displacements event and slope vs. time were also similar with linear diffusion in our model. Slope increases as more displacements event happened but would grow gentler as more time passed.

3.2. Production-Limited Scarp

In this simulation, I added bedrock component to our model to simulate the effect of production limitation on transportable material to surface evolution of fault scarps. The parameters that we used were identical to transport-limited scarp with the addition of soil production parameters. The simulations were done using depth dependent linear diffusion and depth dependent non-linear diffusion modules from landlab and were ran separately into three displacement mode.

The models generated from depth dependent linear diffusion and depth dependent non-linear diffusion, as shown in Figure 2.9 and Figure 2.10 were similar. However, when we look at the resulting soil thickness between the two, as shown in Figure 2.13 and Figure 2.14, we found that the soil was thicker in non-linear diffusion, and with more pronounced variation in single displacement event. In contrast, the soil thickness was similar in continuous slip case for either linear or non-linear model. This means that in an active fault scarp where the displacement events occur often, it does not make much difference whether we use linear diffusion or non-linear diffusion for our production-limited model.

3.2.1. Depth Dependent Linear Diffusion

Figure 2.9 shows the resulting model from our simulation using depth dependent linear diffusion in three displacement mode. The exposure of bedrock in our production-limited model made noticeable difference from its transport-limited counterpart. Linear diffusion applied to production-limited model creates a steeper scarp surface compared to transport-limited model, especially in the area where the bedrock was exposed. The relationship between slope vs. time and slope vs. displacement events were similar to its transport-limited counterpart, where the slope increases with the number of displacement

event and decrease with the amount of time passed. The area of exposed bedrock in this model also grew proportional to scarp surface slope, where it would be more prominent as the slope increases and become less noticeable as the slope decreases. The relationship between bedrock exposure with time and displacements event can be inferred from previous statement, where less bedrock would be exposed as more time passed, and more bedrock would be exposed as more displacement events occur.

3.2.2. Depth Dependent Non-Linear Diffusion

In Figure 2.10, we simulate the production-limited scarp using depth dependent non-linear diffusion run in three different displacement modes. The resulting models show similar appearances to the ones run using depth dependent linear diffusion. The non-linear diffusion model in our simulation resulted in thicker soil compared to its linear diffusion counterpart, as shown in Figure 2.13 and Figure 2.14. The differences between the two were prominent in the early phase of scarp evolution, which was expected due to the attainment of Sc in non-linear model where more materials would be transported when the local slope approaching Sc (see Figure 2.1)

4. Discussion

The fault scarps from the simulations show distinctions between transport-limited and production-limited scarps and how the non-linear model becomes less prominent as more time passed and more displacements occurred. Figures 2.8, 2.11, and 2.12 show a decrease in topographic differences over time, which were consistent across all diffusion models in the simulations. The decrease in topographic differences were also observed in relation to the increase in displacement events, where single displacement event would have bigger differences than multiple events or continuous offset.

While we expect the non-linear model to become less prominent over its linear counterpart with the passage of sufficient time, we did not expect that the same can be said for the number of displacements. The fact that it did means that displacement events can be used to constraint other parameters, such as the amount of time passed, which in turns open more possibilities to a more accurate assessment of fault scarp evolution. For example, we could differentiate whether a fault scarp is young and inactive or old and active by comparing the surface of the scarp with models that were ran in multiple times and displacements mode. If the resulting comparison pointed toward a young and inactive scarp, then we need to rerun our model in non-linear diffusion mode if we have not done so, whereas if the scarp looks more toward old and active model, we can choose whether to build non-linear model or not, because there was not much difference between linear and non-linear models in old and active fault scarp. Apparent relationship between displacement mode and scarp age that we observed in our simulation is promising, but more data were needed to confirm this conclusion.

Figures

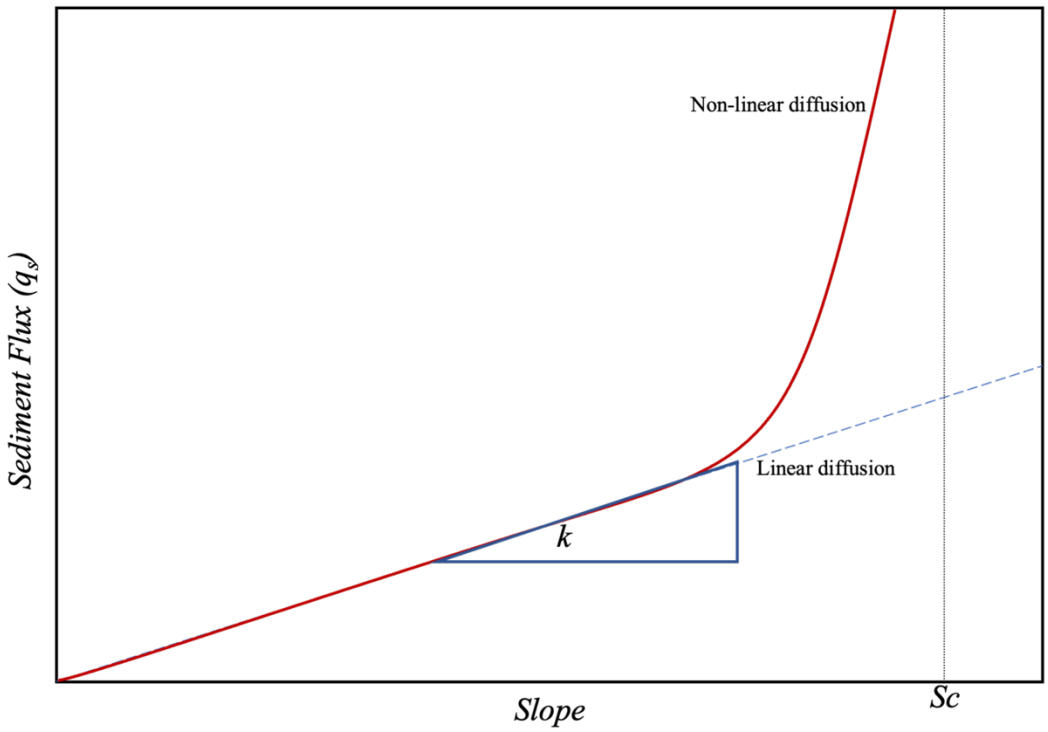


Figure 2.1. Sediment flux (q_s) versus local slope as potentially applied to diffusive landscapes. The dashed yellow line represents the relationship expected from a linear model, whereas the solid red line shows the expected value for non-linear model. In the non-linear model, the sediment flux becomes infinite as the local slope approaches critical gradient (Sc).

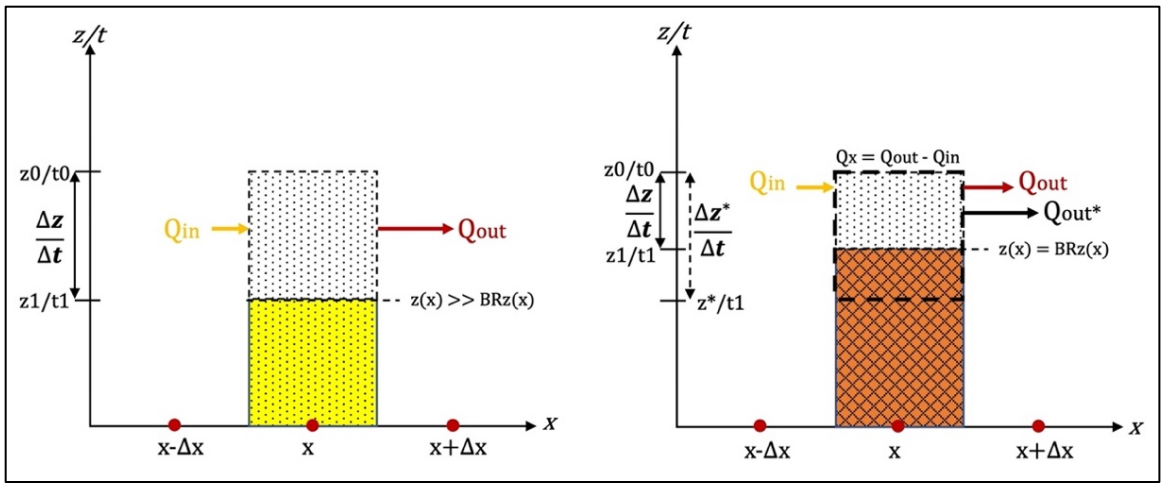


Figure 2.2. Illustration of continuity equation that differentiates transport-limited process with production-limited one. (a) Transport-limited process where the rate of geomorphic displacement ($\Delta z/(\Delta t$) depends on continuity of sediment transport at given point x. (b) Production-limited process where the presence of bedrock obstructs the availability of transportable material. The rate of geomorphic displacement is reduced to what available at that time, making the actual geomorphic displacement rate ($\Delta z/(\Delta t$) is less than the potential geomorphic displacement ($\Delta z^*/(\Delta t$) that it could have at a given slope. The decrease in actual geomorphic displacement rate (($\Delta z/(\Delta t$) directly translates to a decrease in actual material transport rate downslope [$Q(x+\Delta x)$; Qout] from the potential material transport rate downslope [$Qt(x+\Delta x)$; Qout*]. (Adapted from Arrowsmith et al., 1996).

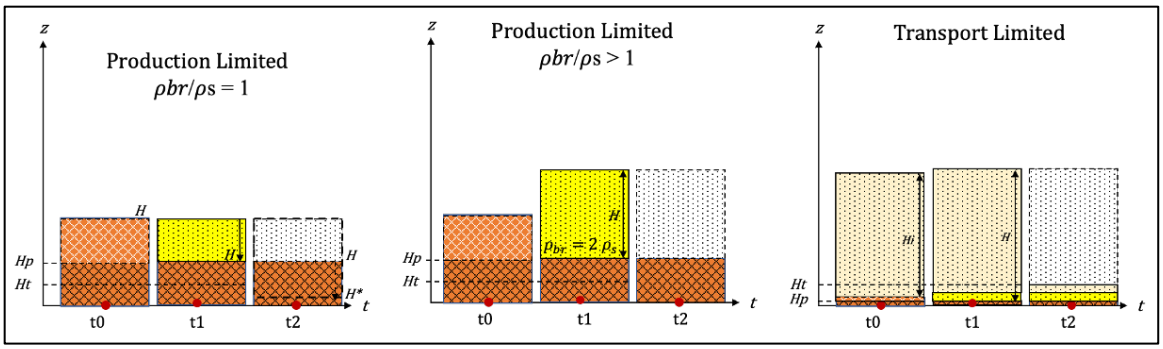


Figure 2.3. Comparison of changes in elevation over time between production-limited vs. transport-limited scarps. The bottom most stratum is the bedrock layer, while the yellow layer is the soil produced from weathered bedrock. The beige layer labeled as *Hi* in transport limited is the initial soil depth to bedrock.

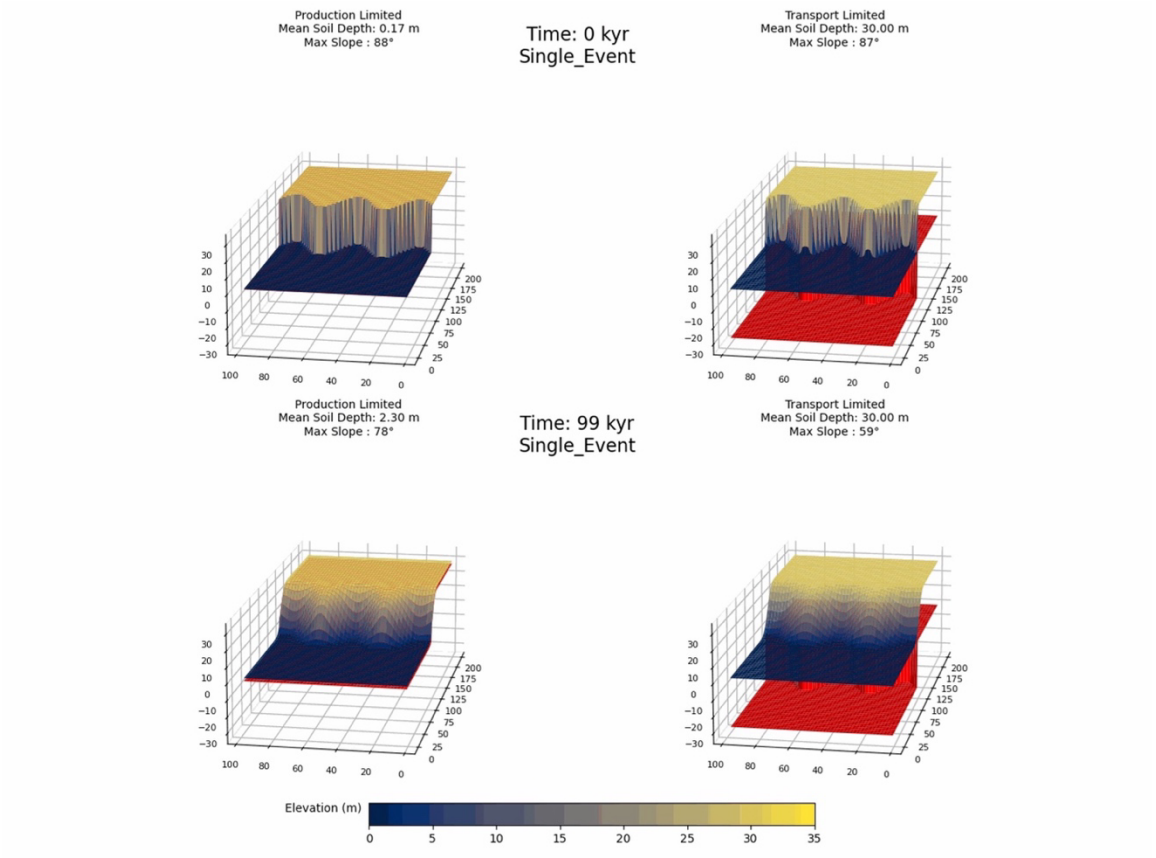
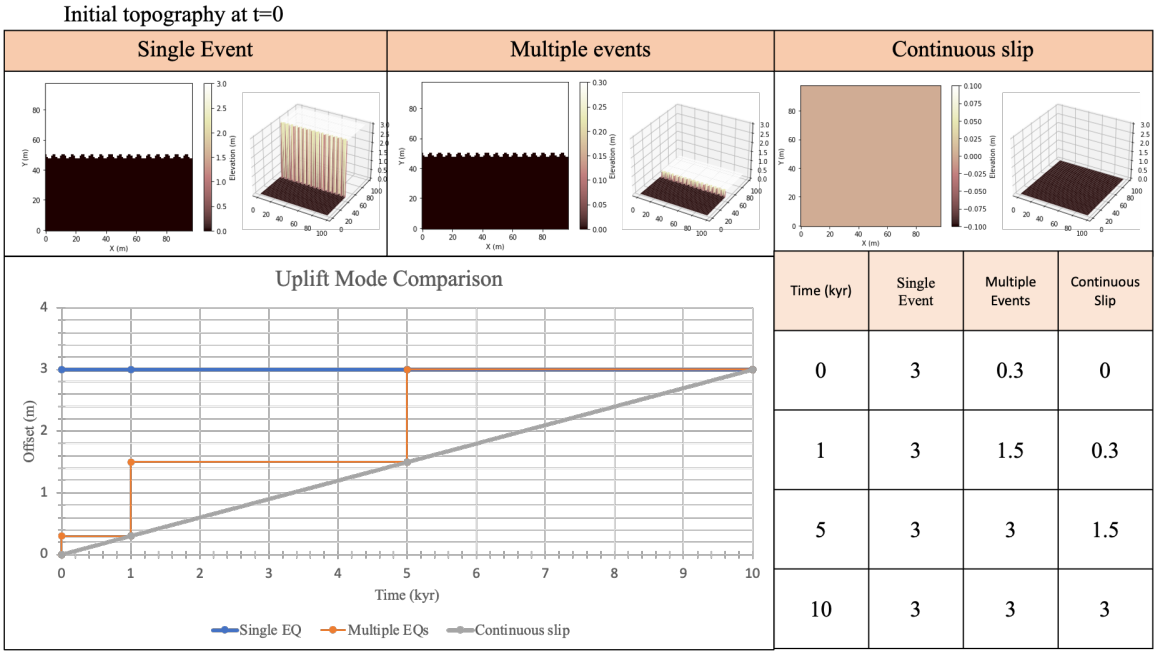


Figure 2.4. Convergence and divergence of sediment on an undulated fault scarp as revealed by a 2D diffusion model. This illustration highlights the spatial variations in

sediment transport, effectively capturing areas of accumulation (convergence) and dispersal (divergence) along the irregular fault surface, demonstrating the advantages of utilizing a 2D model over a 1D approach for a more comprehensive understanding of fault scarp dynamics.



Time (kyr)	Single Event	Multiple Events	Continuous Slip
0	3	0.3	0
1	3	1.5	0.3
5	3	3	1.5
10	3	3	3

Figure 2.5. Initial model setting for various scarp displacement scenarios. For the single event scenario, the displacement occurs only once, while in multiple events case, the displacement happens multiple times but not continuously. In continuous slip case, the displacement is steady. (see Table 2.1 for model parameters)

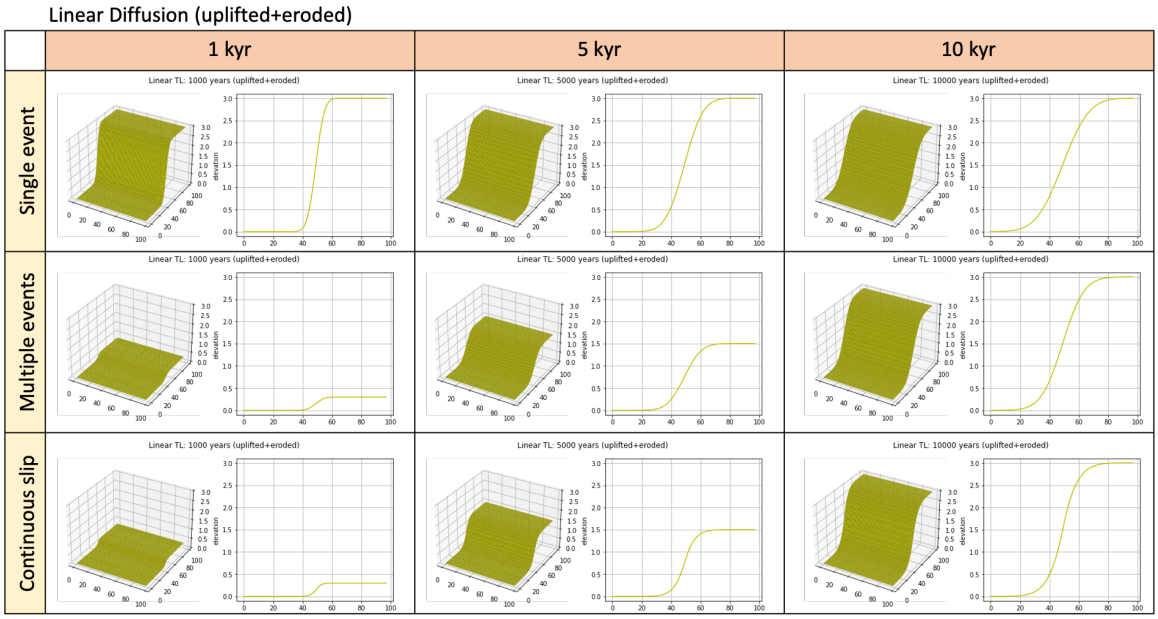


Figure 2.6. Linear diffusion model on various time and uplift mode. The profile shown on the right of each plot was cut from the middle of the scarp.

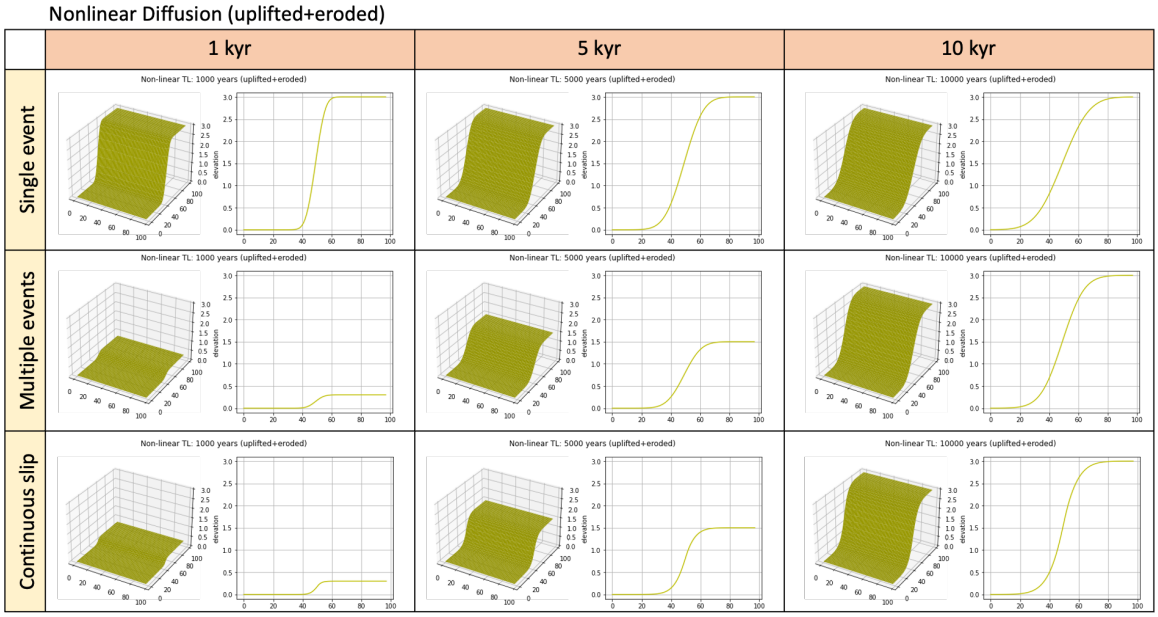


Figure 2.7. Non-linear diffusion model on various time and uplift mode.

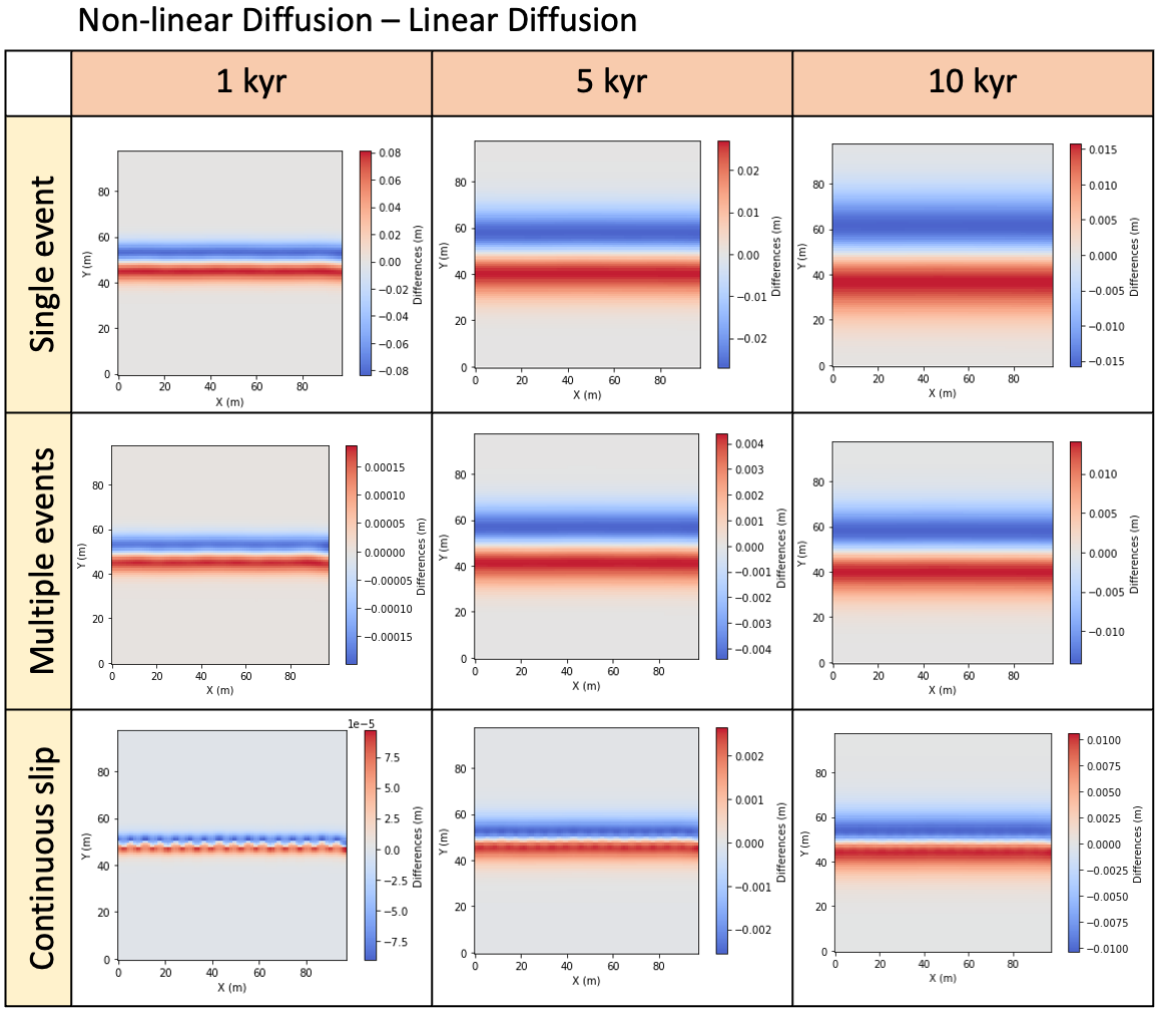


Figure 2.8. Topographic differences between non-linear diffusion and linear diffusion

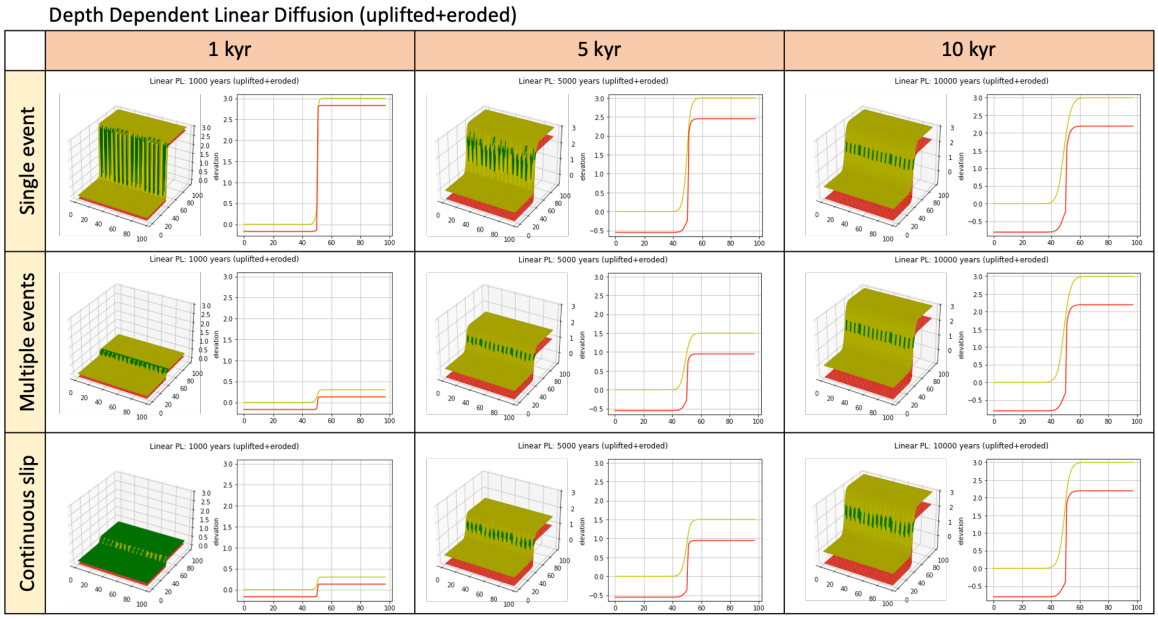


Figure 2.9. Depth dependent linear diffusion model on various time and uplift mode.

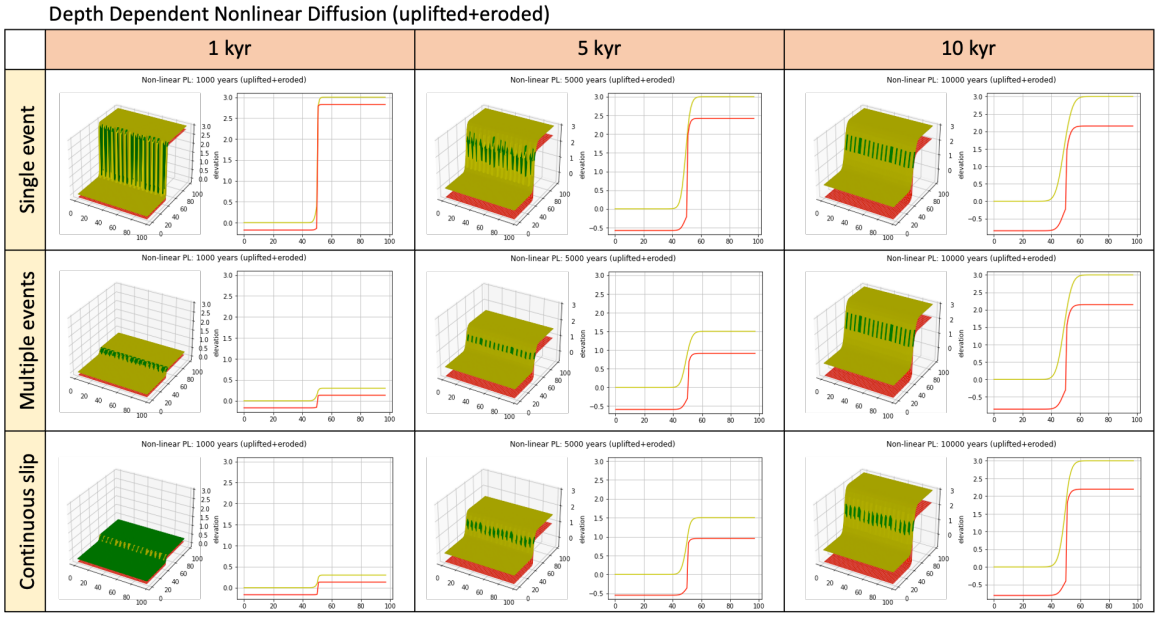


Figure 2.10. Depth dependent non-linear diffusion model on various time and uplift mode.

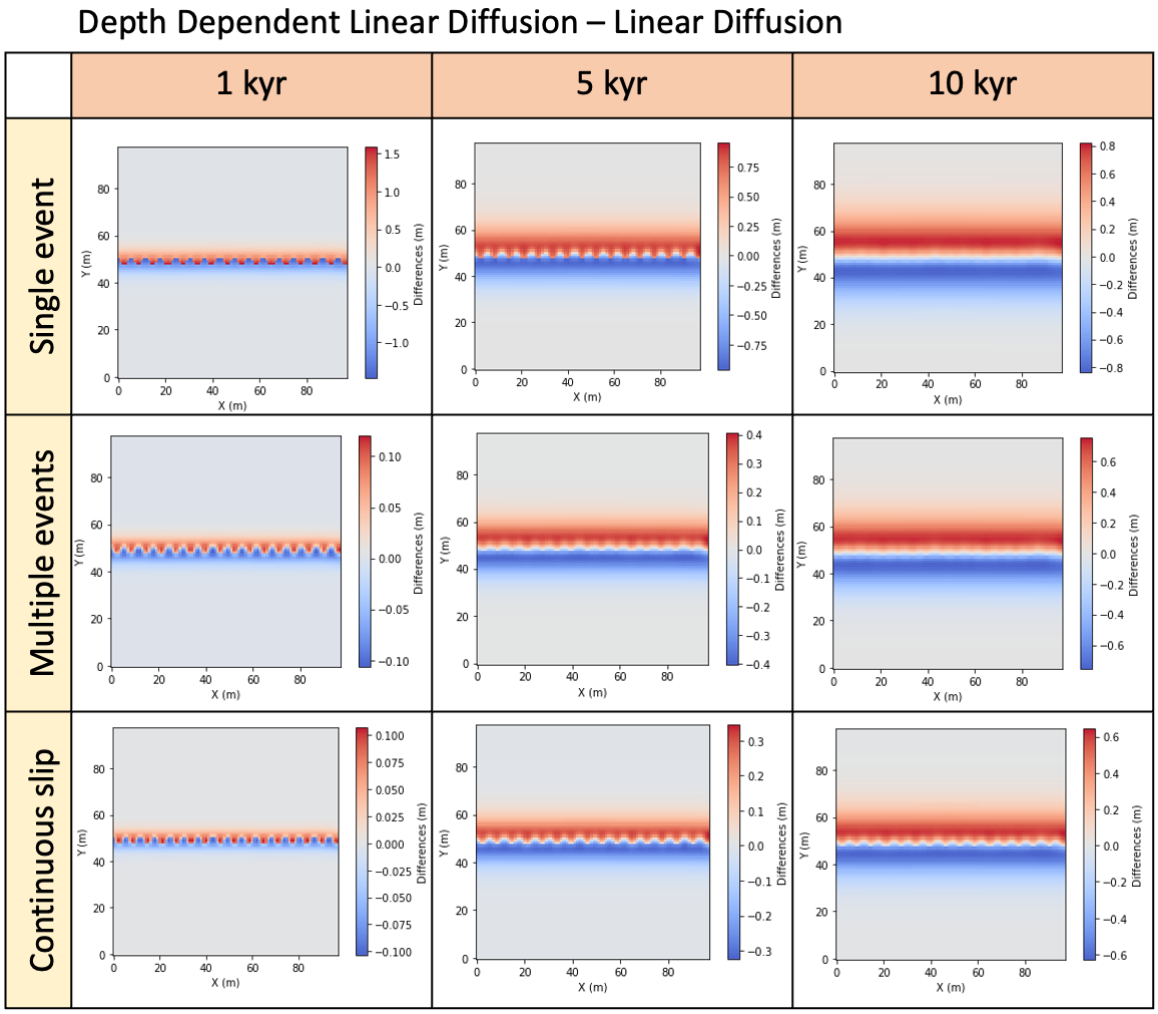


Figure 2.11. Topographic differences between linear diffusion PL and linear diffusion

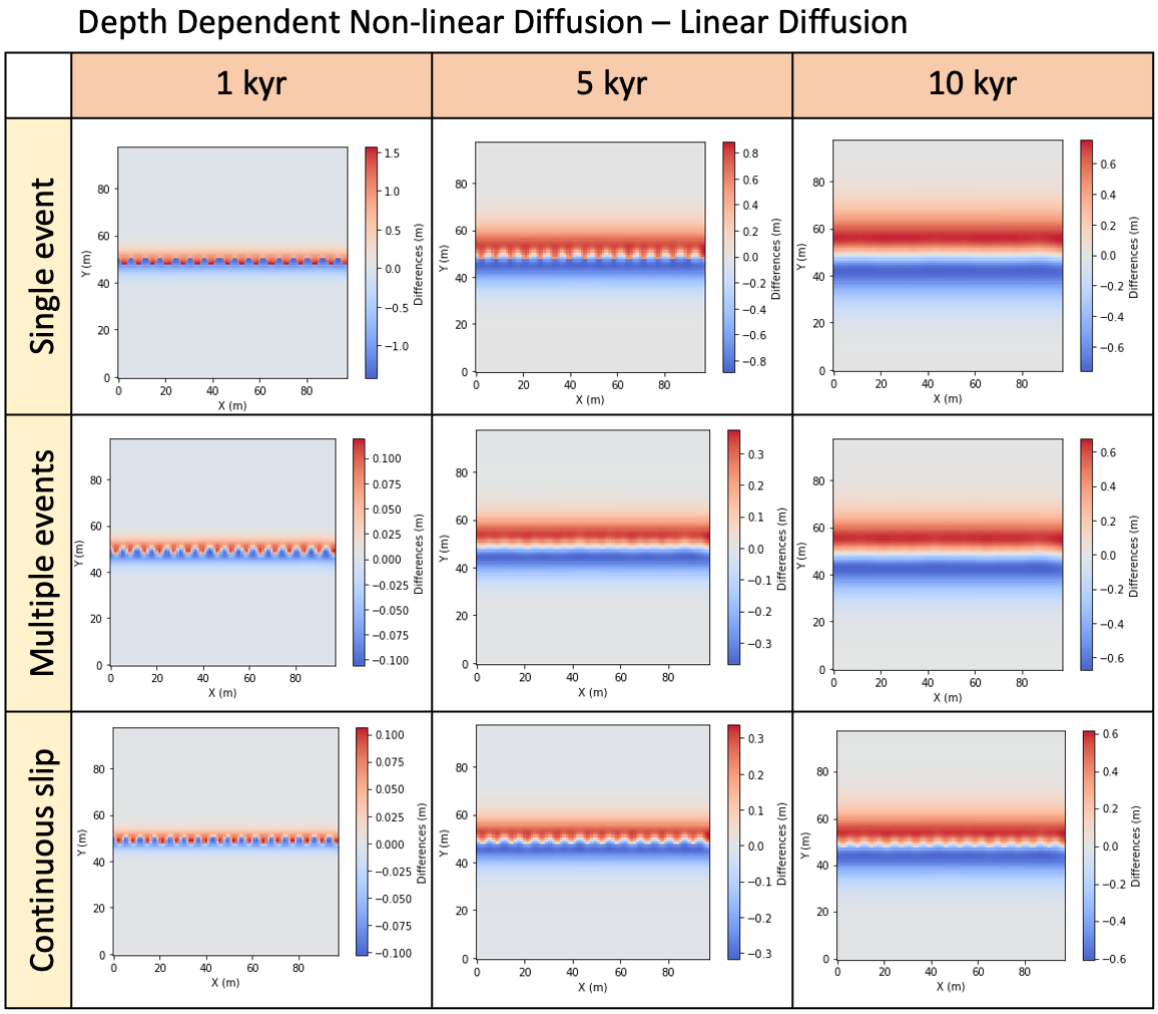


Figure 2.12. Topographic differences between non-linear diffusion PL and linear diffusion

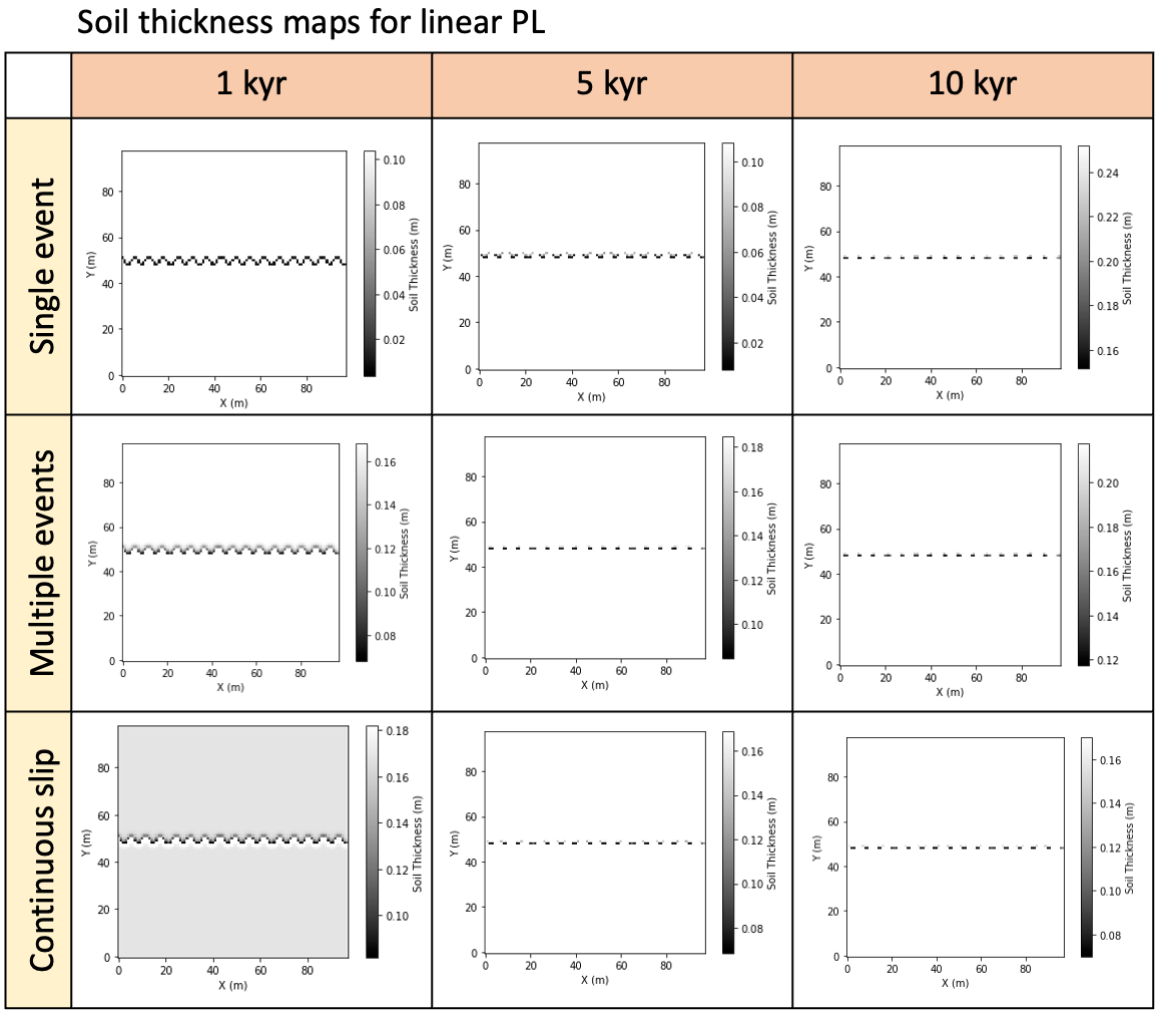


Figure 2.13. Soil thickness maps for linear PL

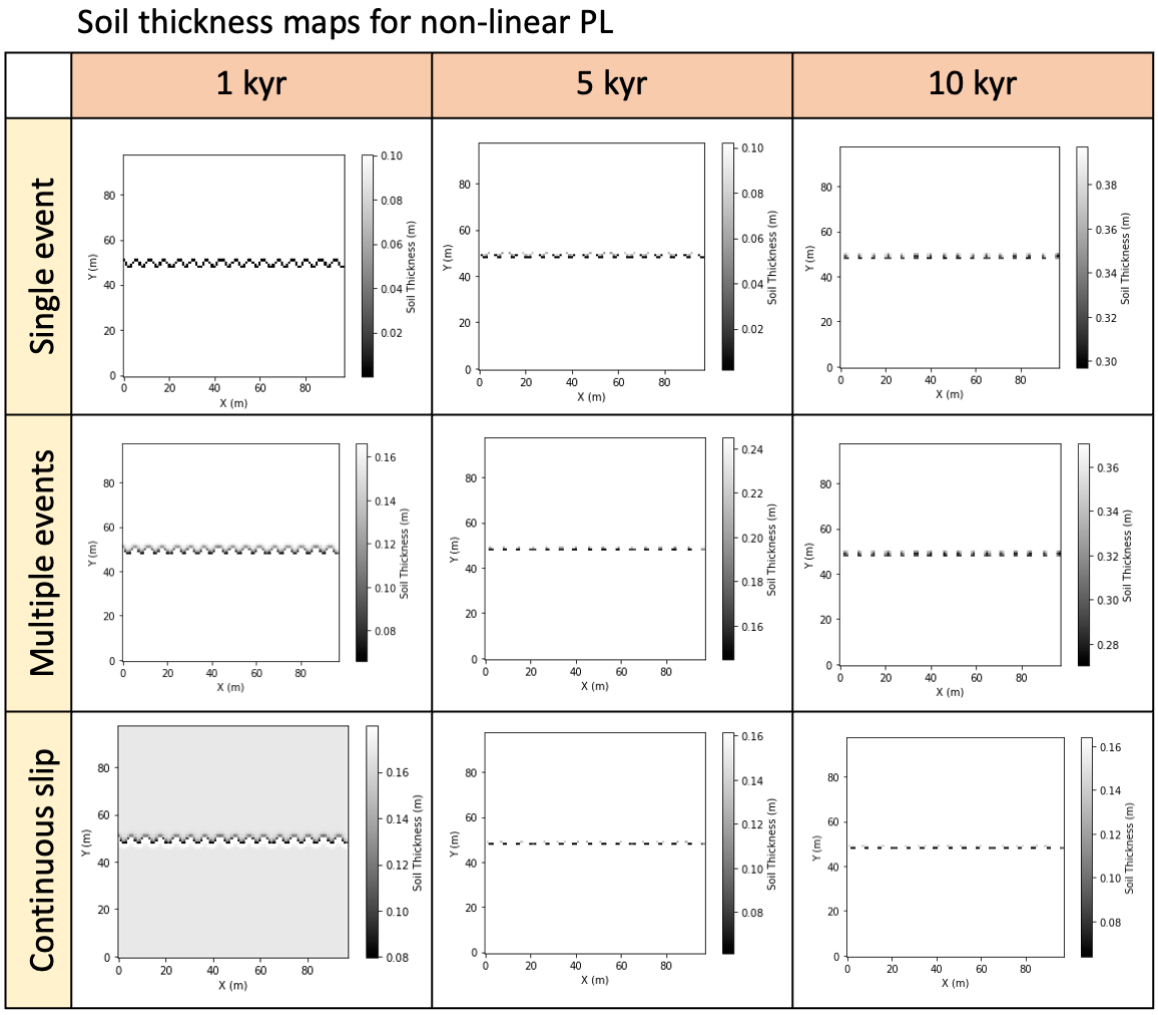


Figure 2.14. Soil thickness maps for non-linear PL

Tables

Table 2.1. Parameter used in landlab for fault scarp diffusion using four different diffusion modules. The value used for the parameters in this table was derived from a few sources. The *k* used here was derived from Arrowsmith et al. (1996, 1998), *Ht* was derived from Johnstone & Hilley (2015), and w_0 was derived from Dixon & von Blanckenburg (2012).

Model Parameters	Linear TL	Nonlinear TL	Linear PL	Nonlinear PL
Initial surface slope (Si)	0^o	0^o	0^o	0^o
Fault dip (Sf)	90^o	90^o	90^o	90^o
Critical slope (Sc)	-	35^o	-	35^o
Transport constant (k)	0.01 m/year	0.01 m/year	0.01 m/year	0.01 m/year
Maximum soil production rate (w0)	-	-	0.0002 m/year	0.0002 m/year
Initial soil depth (Hi)	-	-	0 m	0 m
Soil transport decay depth (Ht)	-	-	0.12 m	0.12 m
Soil production decay depth (Hp)	-	-	0.5 m	0.5 m
Soil production expansion factor ($\rho br/\rho s$)	-	-	1.0	1.0
Number of terms (N)	-	2	-	2

CHAPTER 3

LANDSCAPE EVOLUTION MODEL APPLICATION TO AREA OF ACTIVE TECTONICS

1. Introduction

The Earth's surface is constantly evolving due to a variety of factors such as tectonic activity, erosion, weathering, and sedimentation. In areas of active tectonics, the combination of these processes can lead to rapid changes in the landscape. Landscape evolution models, such as diffusion modeling (Chapter 2), can provide valuable insights into the formation and evolution of certain features in area of active tectonics (e.g., fault scarps). These models simulate the movement of sediment and rock across a landscape over time and can be used to investigate how fault scarps evolve in response to tectonic and geomorphic processes.

In this chapter, we explore the application of landscape evolution models to areas of active tectonics. Specifically, we will examine how these models can be used to study the dynamic landscape changes that occur over a fault scarp. We will also discuss the challenges of modeling landscapes in areas of active tectonics, including the need to accurately represent complex geological features and the difficulty of predicting the timing and magnitude of tectonic events.

The use of landscape evolution models in areas of active tectonics holds great potential for advancing our understanding of the Earth's surface processes and the hazards associated with tectonic activity. By improving our ability to model landscape evolution, we can gain valuable insights into the complex interplay between geological

processes, and ultimately, better prepare for and mitigate the impact of natural disasters such as earthquakes, landslides, and volcanic eruptions.

2. Tools and Methods

2.1. Digital Landscape Reconstruction Using SfM

The topographic model used as input for the diffusion modeling in this chapter was generated using a technique called Structure from Motion (SfM) photogrammetry (e.g., James & Robson, 2012; Johnson et al., 2014; Westoby et al., 2012). This technique involves taking multiple overlapping photographs of a landscape from different angles using a drone, and then using specialized software to create a 3D model from these photographs. In this case, the software used was Agisoft Metashape.

SfM works by using algorithms to identify and match common features in the overlapping photographs, such as distinct points on the landscape or features on buildings or structures. By comparing the position of these features in each photograph, the software can then calculate the position and orientation of the drone at each point in time and reconstruct a 3D model of the landscape. This method has become increasingly popular in recent years due to the ease and affordability of drone technology and the availability of specialized software for photogrammetry processing (James et al., 2017; Westoby et al., 2012). The resulting 3D model can be used as 2D raster input for landscape evolution models like diffusion modeling, which simulate how the landscape evolves over time due to tectonic and geomorphic processes.

2.2. Model Initiation from Field Observation and Digital Reconstruction

Once the 3D model was generated, it was combined with data from field observation to initiate our model. The model that I developed in this chapter calculates

both geomorphic and tectonic displacement. The flow diagram in Figure 3.1 outlines the various steps involved in initiating the model, including importing modules, setting up model from DEM, setting up geomorphic process, setting up tectonic displacements, and doing the grid search.

Most of the modules used in this simulation are available in landlab, but there were some modules that I modified to fit the modeling needs. The main module that I used to simulate tectonic displacements is *NormalFault*, which can only do vertical displacement, thus the following script was made to do horizontal displacement before the topography was displaced vertically:

```
#Horizontal Displacement Setup
def  shift_elevation(grid,  layers_1d,  start_distance,  end_distance,
shift_distance):
    layers = layers_1d.reshape(grid.shape)
    start_column = int(start_distance / grid.dx)
    end_column = int(end_distance / grid.dx)
    shift_amount = int(shift_distance / grid.dx)
    wrapped_columns = layers[:,end_column-shift_amount:
                             end_column].copy()
    layers[:, start_column:end_column - shift_amount] =
                      layers[:,start_column + shift_amount:end_column]
    layers[:, end_column - shift_amount:end_column] = wrapped_columns
    return layers.flatten()
```

For geomorphic displacement, I used a modified version of *ExponentialWeathererIntegrated* and a few diffuser modules to simulate different approximation to geomorphic process on fault scarps. The following script shows a function that I made to initiate the geomorphic process parameters:

```
#Exponential Weatherer
def weatherer_mode(mg,w0,Hp=0.5,ef=1):
    weatherer = ExponentialWeathererIntegrated(mg,
                soil_production__maximum_rate = w0 ,
                soil_production__decay_depth = Hp,
                soil_production__expansion_factor = ef)
    return weatherer

#Diffuser mode
def diffuser_mode(mg,k,Ht = 0.12,Sc = 1):
```

```
    ld = LinearDiffuser(mg, linear_diffusivity=k)
    td = TaylorNonLinearDiffuser(mg,
         linear_diffusivity=k,slope_crit=Sc,dynamic_dt=True)
    ddld = DepthDependentDiffuser(mg, linear_diffusivity = k,
           soil_transport_decay_depth = Ht)
    ddtd = DepthDependentTaylorDiffuser(mg, soil_transport_velocity =
           k, slope_crit = Sc, soil_transport_decay_depth = Ht,
           dynamic_dt=True)
    return [ld,td,ddld,ddtd]
```

The depth dependent diffuser modules such as *DepthDependentDiffuser* and *DepthDependentTaylorDiffuser* were used to simulate production-limited conditions on fault scarp. I also used *LinearDiffuser* and *TaylorNonLinearDiffuser* to simulate a transport-limited conditions and comparing the results with its production-limited pair to see which conditions fit our data the best.

2.3. RMSE Implementation in Finding Best Fit Model

In the context of diffusion modeling of fault scarps, the Root Mean Square Error (RMSE) is a commonly used metric to assess the goodness of fit of the model. RMSE is a statistical measure of the differences between the predicted values of a model and the actual values of the data. In other words, RMSE measures the accuracy of the model by quantifying the difference between the predicted values and the observed values. It is calculated as the square root of the average of the squared differences between predicted and observed values. The formula for RMSE is:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(observed - predicted)^2}{n}} \tag{3.1}$$

where n is the number of data points, observed are the actual values, and predicted are the predicted values.

When using diffusion modeling to analyze fault scarps, the goal is to find the model parameters that best fit the observed topographic data of the fault scarp. This involves running the diffusion model multiple times with different parameter values and comparing the resulting predicted topography to the observed data using RMSE. The model with the lowest RMSE is considered the best fit model.

In this chapter, we calculate RMSE from *mean_squared_error* module that can be found in *sklearn.metrics* which was part of scikit-learn packages in python. We used this module to simplify our code, but if needed be, we can still get RMSE value by making a function to apply the equation described in (3.1). The following snippet shows how to calculate RMSE using *mean_squared_error* from *sklearn.metrics*.

```
from sklearn.metrics import mean_squared_error
import numpy as np

# observed and predicted data
observed = np.array([1, 2, 3, 4, 5])
predicted = np.array([1.5, 2.5, 3.5, 4.5, 5.5])
# calculate mean squared error
mse = mean_squared_error(observed, predicted)
# calculate RMSE
rmse = np.sqrt(mse)
print("RMSE:", rmse)
```

The advantage of using RMSE to find the best fit model is that it provides a quantitative measure of model accuracy, allowing for objective comparisons of different parameter combinations. However, it is important to note that the RMSE is just one measure of model accuracy, and it does not necessarily capture all aspects of model performance.

3. Landscape Evolution Model in Old and Inactive Fault Scarp

3.1. Background

Twin Butte is located in the Hat Creek Valley, a region of active tectonics and volcanic activity in northeastern California (Smith & Doe, 2012). The Hat Creek Valley lies within the Cascades volcanic arc, a chain of volcanic mountains that extends from northern California to southern British Columbia (Hildreth, 2007). The volcanic arc is created by the subduction of the Juan de Fuca plate beneath the North American plate, which causes magma to rise to the surface and create volcanic activity (Wilson, 2015).

The Hat Creek Valley is also home to the Hat Creek fault zone, a system of faults that accommodates the movement of the two tectonic blocks (Anderson et al., 2010). The Hat Creek fault zone is characterized by strike-slip faults, which are caused by horizontal movement along the fault (Furlong et al., 2011). The movement along the faults creates tension and compression in the Earth's crust, leading to the formation of fault scarps (Smith & Doe, 2012).

The Twin Butte area is particularly interesting because it contains both volcanic and tectonic features (Johnson & Smith, 2016). The buttes themselves are composed of volcanic tuff. The tuff was deposited during the eruption of the nearby Medicine Lake volcano, which is one of the largest shield volcanoes in the Cascade Range (Hildreth, 2007).

In addition to the volcanic features, the Twin Butte area also contains numerous fault scarps, including the Twin Butte fault scarp (Anderson et al., 2010). These fault scarps are evidence of tectonic activity in the region and provide valuable information about the movement of the Earth's crust in the area (Furlong et al., 2011). By studying the fault scarps

using techniques such as diffusion modeling, we can gain a better understanding of the tectonic activity and earthquake hazards in the Hat Creek Valley (Johnson & Smith, 2016).

In this chapter, I used DEM generated from 3D models of Twin Butte fault scarps to performed diffusion modeling on Twin Butte scarp, which can be categorized as old and inactive fault scarp. According to Clynney and Muffler (2010), the lava flow that formed Twin Butte were formed less than a hundred thousand years ago. A simplified geological map of Twin Butte and its surrounding area can be seen in Figure 3.2, where Twin Butte was classified as a part of quaternary volcanic formation (Qv). The oblique view of Twin Butte fault scarp captured by a drone can be seen in Figure 3.3.

3.2. Model Initiation

In this simulation, I integrate tectonic displacement and geomorphic displacement into landlab diffusion model to better understand the evolution of fault scarps and its response to different tectonic and geomorphic processes. The general workflow for creating the model involves importing landlab modules, setting up the initial model from a Digital Elevation Model (DEM), and defining both geomorphic and tectonic displacements. In this section, we will focus on setting up the initial model from a DEM.

The first step in setting up the initial model from a DEM is to import the elevation data into Landlab's grid system. This is achieved by reading in the DEM file, extracting the elevation data, and initializing a grid that can accommodate this information. The following script was used to import the DEM into landlab grid:

```
#import DEM
from landlab.io import read_esri_ascii
(grid_dem,grid_dem.at_node["topographic__elevation"]) =
                                    read_esri_ascii("./input/DEM.asc")
```

Next, a scarp's swath and slope map is created from the DEM. A swath profile is a technique used to analyze the cross-sectional topography along a fault scarp, while the slope map is used to visualize and quantify the gradient of the terrain. These maps provide critical information on the fault scarp's morphology and allow the model to assess the initial conditions of the landscape. The following script show how to use swath of DEM to get the initial conditions of the model:

```
#pick a surface to do linear regression on
base = [0,60]
fault = [75,95]
top = [125,164]
x_div = [base,fault,top]
#plot the selected points
x_div_range = []
zx_div_range = []
for i in range(len(x_div)):
    x_div_range.append(np.arange(x_div[i][0],x_div[i][1],xy_spacing))
    zx_div = []
    for j in (x_div_range[i]/xy_spacing):
        zx_div.append(dem_swath[int(j)])
    zx_div_range.append(zx_div)
#fit points using linear regression
c_fit = []
fx = []
m = []
c = []
for i in range(len(x_div)):
    c_fit.append(np.polyfit(x_div_range[i],zx_div_range[i],deg=1))
    fx.append(np.poly1d(c_fit[i]))
    m.append(c_fit[i][0])
    c.append(c_fit[i][1])
```

Finally, model parameters are set up to define the simulation conditions and the behavior of both geomorphic and tectonic processes. These parameters include the time step, the diffusion coefficient, and any other settings that control the simulation's progression.

3.3. Model Simulation and Fit

In this simulation, I ran the initial model of Twin Butte fault scarps over various tectonic and geomorphic conditions. We first constraint the value of *k* and *t* that will be used for further simulation by doing a grid search on single-event non-linear models. The single-event non-linear model was generated using *DepthDependentTaylorDiffuser*. I ran the model over multiple *k* and *t* values to get the optimum value for this parameter. The range of t was constrained by the age of rocks that formed this scarp, which according to Clynne & Muffler (2010) was believed to formed less than 100 kyr ago. Apart from k and t, all the other parameters in our models are constant, where maximum soil production rate at 0.0002 m/year, soil production decay depth at 0.5 m, and soil transport decay depth at 0.12 m. The results for grid search of Twin Butte's model were shown in Figure 3.4. The RMSE minimum was found at k = 7.5 m^2/kyr and t = 110 kyr. Figure 3.5 show the comparison between the model and DEM.

4. Landscape Evolution Model in Young and Active Fault Scarp

4.1. Background

The Hat Creek fault zone is located in northeastern California and is part of the larger Walker Lane tectonic zone. This fault zone is characterized by a series of en-echelon faults that are thought to have formed as a result of dextral strike-slip faulting. The Hat Creek fault zone has been the focus of numerous studies aimed at understanding its tectonic evolution and seismic hazards.

According to Muffler et al., (1994) the rocks in the Hat Creek basin and surrounding areas are predominantly of volcanic origin and are part of the extensive volcanic field that includes the Medicine Lake volcano. The Hat Creek fault zone is located within this

volcanic field and is thought to have been reactivated during the Quaternary period as a result of tectonic stresses related to regional deformation. The simplified geological map of Hat Creek and its surrounding area can be seen in Figure 3.2, where active Hat Creek fault was approximately in Qvr formation and was surrounded by many active faults. In Figure 3.6 we can also see that fault scarp in active Hat Creek fault sites mainly consist of big block of volcanic rocks. Unlike Twin Butte that had been eroded for a while, the fault scarp that we analyze in active Hat Creek fault site are relatively young, with the last eruption occurred around 24 kyr according to Blakeslee & Kattenhorn (2013).

4.2. Model Initiation

The process that I did to initiate the model in Hat Creek fault scarp was similar to initiation process in Twin Butte sites. The only major difference is that I used different DEM compared to Twin Butte sites, and the DEM on this site need to be rotated first to the north, unlike Twin Butte site that already oriented relatively North-South. After the DEM was transformed, I import the elevation data in the DEM to landlab as landlab grid surfaces. By default, when importing DEM to landlab, the x,y coordinates of the DEM would also be imported. To change the coordinates into a local coordinates with x,y origin at (0,0), I made another grid that use the shape and resolution of the DEM's grid to shift the x,y origin of our DEM.

In order to generate the initial shape of our model, we need to make a swath of our DEM, so localized noise such as bush, trees, and others can be minimized. The DEM swath is especially important in active Hat Creek fault site, because there were a lot of localized noise in the DEM, mainly due to tree coverage.

4.3. Model Simulation and Fit

In this simulation, I ran the initial model of active Hat Creek fault scarps over various tectonic and geomorphic conditions. To constraint the value of k and t for our model, I first do a grid search on single-event non-linear model. The single-event non-linear model was generated using *DepthDependentTaylorDiffuser*. The range of t was constrained by the age of the last lava flow that formed this scarps, which according to Blakeslee & Kattenhorn (2013) was around 24 kyr. Apart from k and t, all the other parameters in our models are constant, where maximum soil production rate at 0.0002 m/year, soil production decay depth at 0.5 m, and soil transport decay depth at 0.12 m. The results for grid search on Hat Creek's model are shown in Figure 3.7. The RMSE minimum was found at k = 7.5 m^2/kyr and t = 26kyr. Figure 3.8 show the comparison between the model and DEM.

Figures

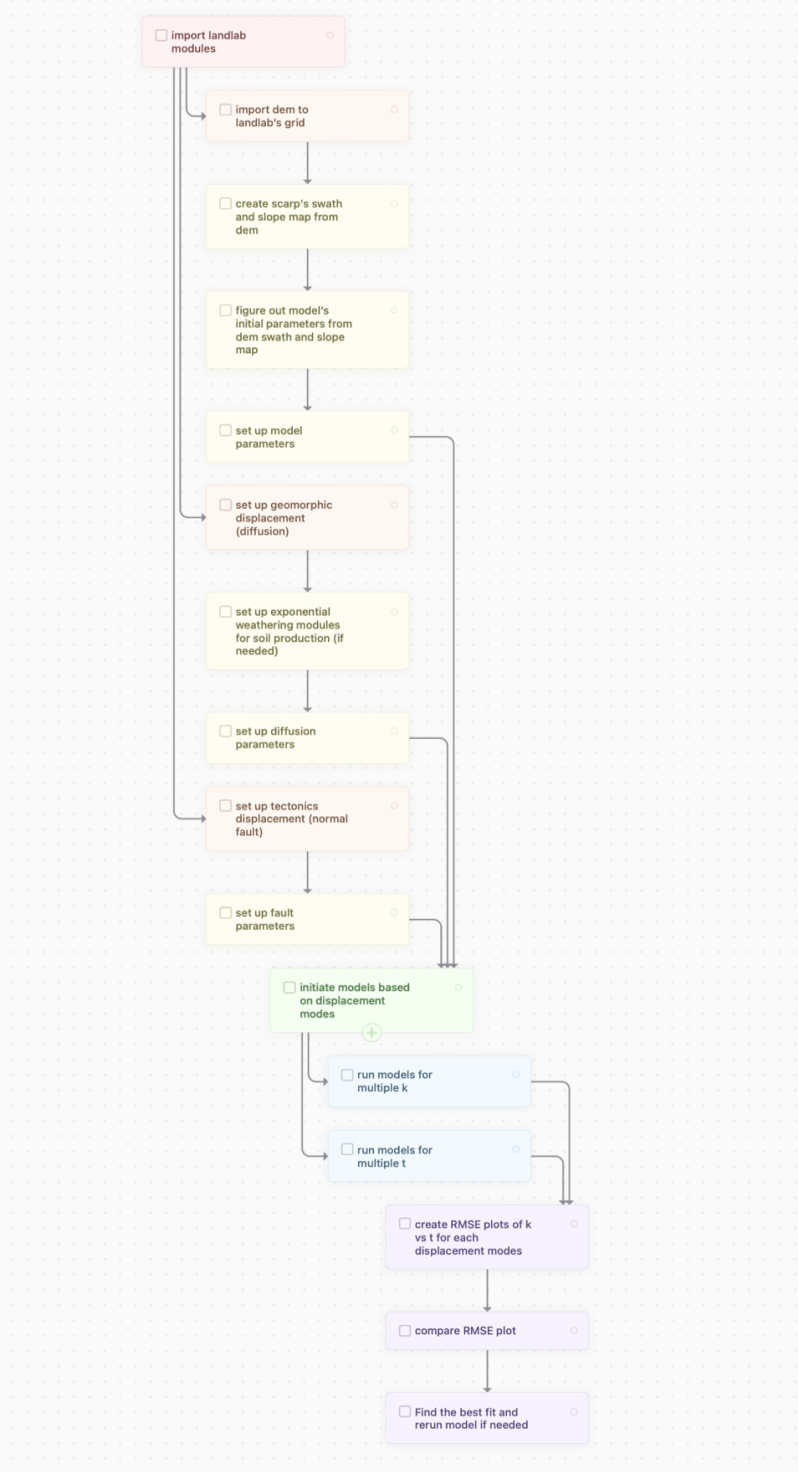


Figure 3.1. Flowchart of 2D fault scarps diffusion modeling in landlab

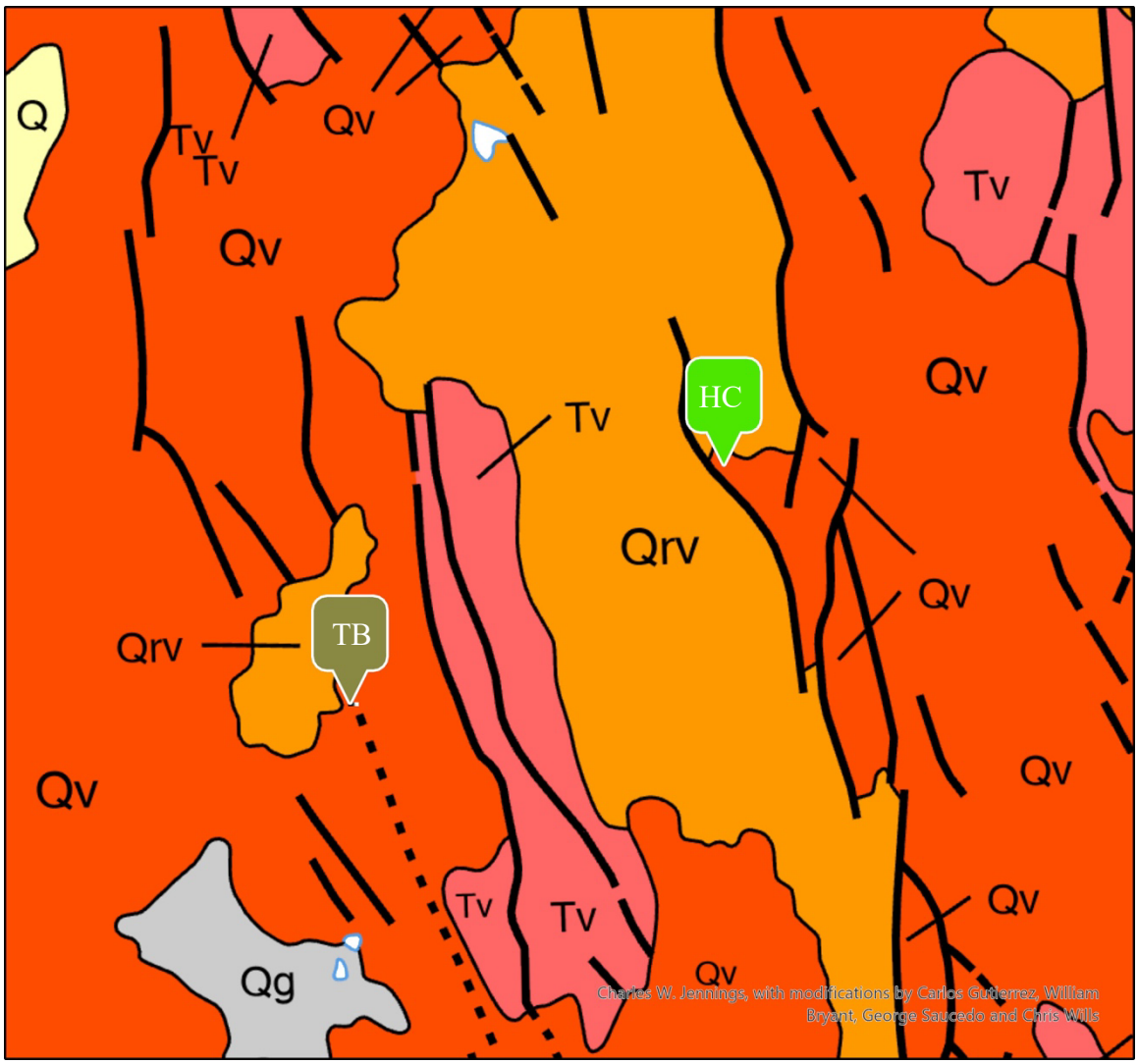


Figure 3.2. Geological map of Hat Creek (HC) and Twin Butte (TB) sites.

Figure 3.3. Oblique view of fault scarp in Twin Butte scarp. View to the southeast.

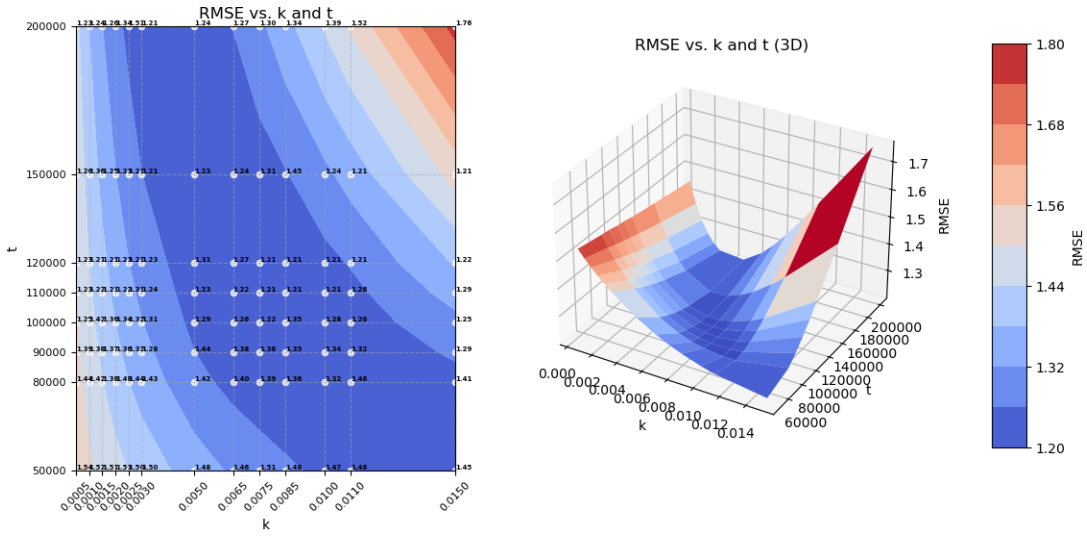


Figure 3.4. RMSE plot of single event non-linear diffusion model in Twin Butte sites.

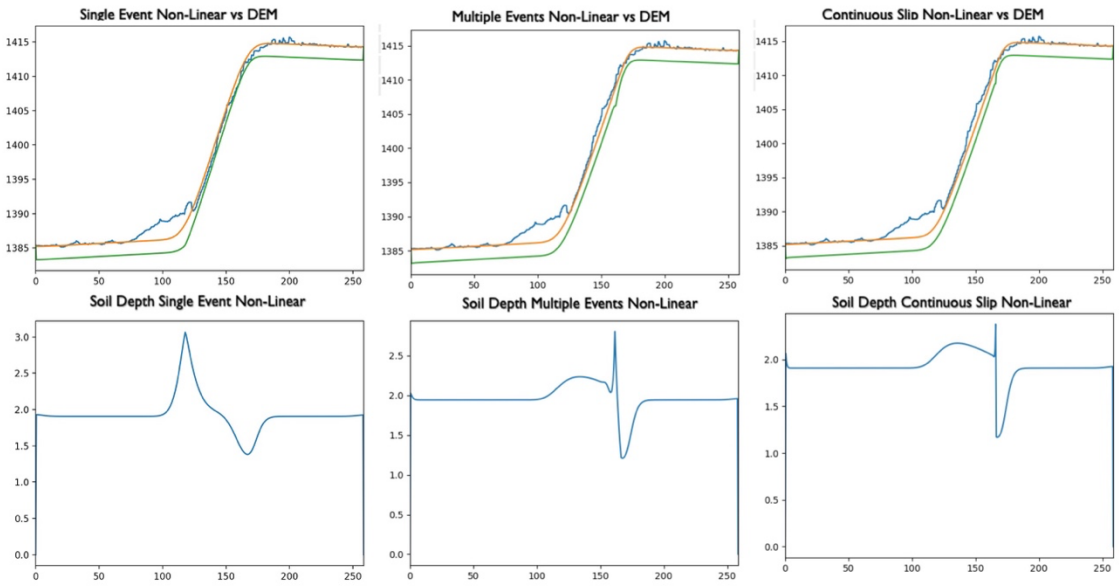


Figure 3.5. Comparison between Twin Butte's model and DEM.

Figure 3.6. Fault scarp in active Hat Creek fault sites

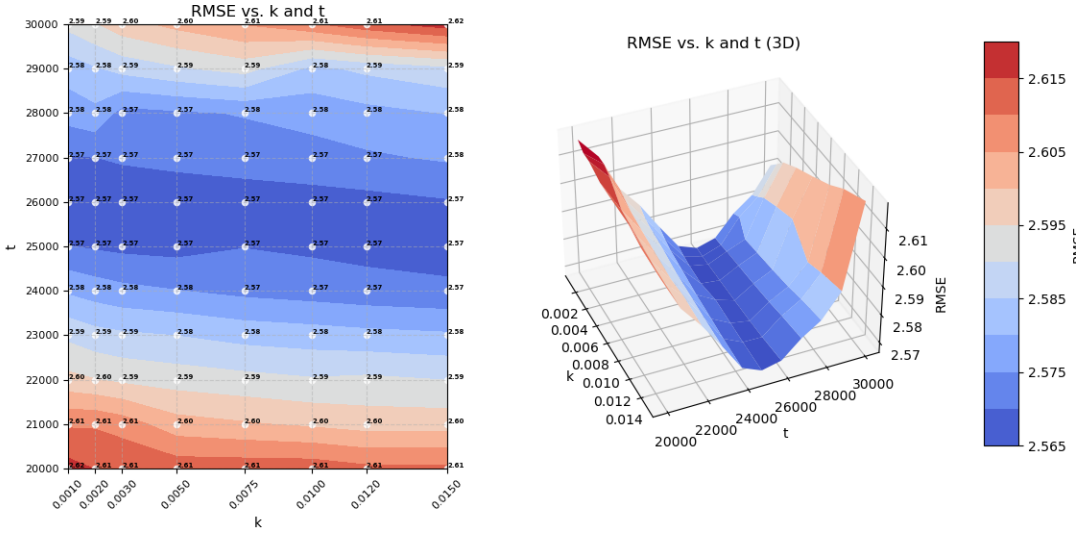


Figure 3.7. RMSE plot of single event non-linear diffusion model in Hat Creek sites.

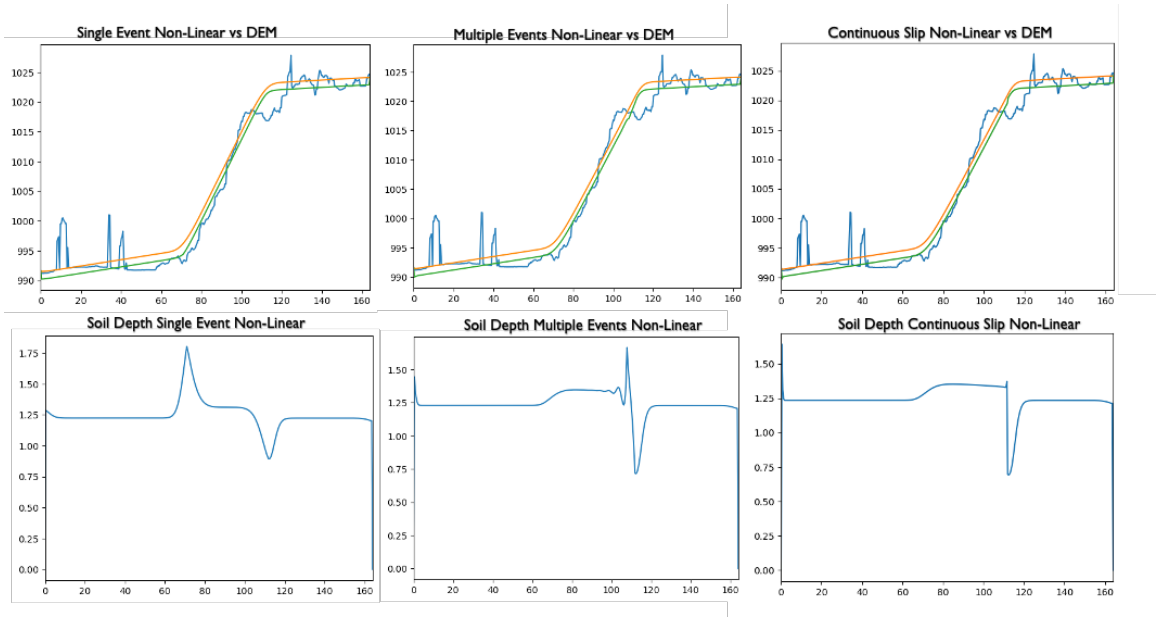


Figure 3.8. Comparison between Hat Creek's model and DEM.

CHAPTER 4

CONCLUSIONS AND CONTRIBUTIONS

1. Summary

Diffusion modeling is an efficient and cost-effective technique for fault scarp analysis. While its application has primarily been observed in macro-scale studies, such as those examining catchment-sized or mountain range-sized areas, there remains a scarcity of research focused on its use in micro-scale settings. In this thesis, I delve into the application of diffusion modeling on a fault scarp scale, ranging from tens to hundreds of meters, utilizing landlab toolkit to simulate fault scarps evolution in 2D space.

Landlab is a highly versatile tool for landscape evolution modeling and is particularly well-suited for assessing vertical fault scarps. Its capabilities, however, do not currently extend to horizontal displacement calculations, requiring users to devise their own solutions. Nonetheless, this study aims to explore the potential of diffusion modeling in micro-scale fault scarp analysis by harnessing the power of landlab.

By employing Landlab for diffusion modeling on a fault scarp scale, this study contributes valuable insights to a relatively unexplored area of study. The results gleaned from this investigation will not only advance our understanding of fault scarp dynamics at a micro-scale but also pave the way for further development of Landlab's capabilities, ultimately enhancing its effectiveness as a landscape evolution modeling tool.

2. Main Conclusions

In conclusion, the expansion of fault scarps diffusion models from 1D to 2D has been successfully achieved using the Landlab toolkit. This open-source software enables the integration of geomorphic displacement with tectonic displacement in 2D space,

providing a more comprehensive understanding of fault scarp dynamics. While incorporating these factors in the 2D model can be challenging, the benefits of a multidimensional approach are apparent in the improved accuracy and representation of complex geological processes. The availability of open-source tools like Landlab empowers researchers to explore new possibilities in fault scarp analysis and promotes the advancement of our understanding of Earth's dynamic crustal movements.

3. Main Contributions

The main contribution of this thesis is the development of a workflow for fault scarp diffusion modeling in 2D that considers both tectonic and geomorphic displacement on a fault scarp scale. This workflow was designed to provide a more comprehensive understanding of the evolution of fault scarps over time, and to help improve our ability to predict future fault behavior and potential hazards. The model can be applied to a variety of different fault systems and geological settings, and has the potential to advance our understanding of the complex interactions between tectonic forces and landscape evolution.

4. Looking Ahead

Looking ahead, there are a number of exciting opportunities to expand upon this work and further refine the fault scarp diffusion model. One potential area for future development is the integration of Okada or Cutde into Landlab, which would allow for more sophisticated modeling of tectonic displacement. This could help to better capture the complex patterns of deformation that occur in the Earth's crust during earthquake events, and could lead to more accurate predictions of seismic hazards. Additionally, further research could explore the effects of other factors on fault scarp evolution, such as changes in climate or the

presence of other geological structures. Overall, the development of this workflow represents an important step forward in our understanding of fault behavior and has the potential to inform future research in a variety of different fields.

REFERENCES

Anderson, R. S., & Anderson, S. P. (2010). *Geomorphology: the mechanics and chemistry of landscapes*. Cambridge ; New York: Cambridge University Press.

Anderson, R. S., & Humphrey, N. F. (1989). Interaction of weathering and transport processes in the evolution of arid landscapes. In T. A. Cross (Ed.), *Quantitative Dynamic Stratigraphy* (pp. 349–361). Prentice Hall.

Arrowsmith, J. R., Pollard, D. D., & Rhodes, D. D. (1996). Hillslope development in areas of active tectonics. *Journal of Geophysical Research: Solid Earth*, *101*(B3), 6255–6275. https://doi.org/10.1029/95JB02583

Arrowsmith, J. R., Rhodes, D. D., & Pollard, D. D. (1998). Morphologic dating of scarps formed by repeated slip events along the San Andreas Fault, Carrizo Plain, California. *Journal of Geophysical Research: Solid Earth*, *103*(B5), 10141–10160. https://doi.org/10.1029/98JB00505

Barnhart, K. R., Glade, R. C., Shobe, C. M., & Tucker, G. E. (2019). Terrainbento 1.0: a Python package for multi-model analysis in long-term drainage basin evolution. *Geoscientific Model Development*, *12*(4), 1267–1297. https://doi.org/10.5194/gmd-12-1267-2019

Bull, W. B. (1996). Dating San Andreas fault earthquakes with lichenometry. *Geology*, *24*(2), 111–114. https://doi.org/10.1130/0091-7613(1996)024<0111:DSAFEW>2.3.CO;2

Clynne, M. A., & Muffler, L. I. P. (2010). Geologic Map of Lassen Volcanic National Park and Vicinity, California. U.S. Geological Survey Scientific Investigations Map 2899. Retrieved from https://pubs.usgs.gov/sim/2899/

Culling, W. E. H. (1960). Analytical Theory of Erosion. *The Journal of Geology*, *68*(3), 336–344. https://doi.org/10.1086/626663

Dietrich, W. E., Bellugi, D. G., Sklar, L. S., Stock, J. D., Heimsath, A. M., & Roering, J. J. (2003). Geomorphic Transport Laws for Predicting Landscape form and Dynamics. In *Prediction in Geomorphology* (pp. 103–132). American Geophysical Union (AGU). https://doi.org/10.1029/135GM09

Dixon, J. L., & von Blanckenburg, F. (2012). Soils as pacemakers and limiters of global silicate weathering. *Comptes Rendus Geoscience*, *344*(11–12), 597–609. https://doi.org/10.1016/j.crte.2012.10.012

England, P., & Molnar, P. (1990). England, P. & Molnar, P. Surface uplift, uplift of rocks, and exhumation of rocks. Geology 18, 1173-1177. *Geology*, *18*, 1173–1177. https://doi.org/10.1130/0091-7613(1990)018<1173:SUUORA>2.3.CO;2

Gabet, E. J., Reichman, O. J., & Seabloom, E. W. (2003). The Effects of Bioturbation on Soil Processes and Sediment Transport. *Annual Review of Earth and Planetary Sciences*, *31*(1), 249–273. https://doi.org/10.1146/annurev.earth.31.100901.141314

Ganti, V., Passalacqua, P., & Foufoula-Georgiou, E. (2012). A sub-grid scale closure for nonlinear hillslope sediment transport models. *Journal of Geophysical Research: Earth Surface*, *117*(F2). https://doi.org/10.1029/2011JF002181

Hanks, T. C. (2000). The Age of Scarplike Landforms From Diffusion-Equation Analysis. In J. S. Noller, J. M. Sowers, & W. R. Lettis (Eds.), *AGU Reference Shelf* (pp. 313–338). Washington, D. C.: American Geophysical Union. https://doi.org/10.1029/RF004p0313

Hanks, T. C., Bucknam, R. C., Lajoie, K. R., & Wallace, R. E. (1984). Modification of wave-cut and faulting-controlled landforms. *Journal of Geophysical Research*, *89*(B7), 5771–5790.

Heimsath, A. M., Dietrich, W. E., Nishiizumi, K., & Finkel, R. C. (1997). The soil production function and landscape equilibrium. *Nature*, *388*(6640), 358–361. https://doi.org/10.1038/41056

Heimsath, A. M., DiBiase, R. A., & Whipple, K. X. (2012). Soil production limits and the transition to bedrock-dominated landscapes. *Nature Geoscience*, *5*(3), 210–214. https://doi.org/10.1038/ngeo1380

Hobley, D. E. J., Adams, J. M., Nudurupati, S. S., Hutton, E. W. H., Gasparini, N. M., Istanbulluoglu, E., & Tucker, G. E. (2017). Creative computing with Landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of Earth-surface dynamics. *Earth Surface Dynamics*, *5*(1), 21–46. https://doi.org/10.5194/esurf-5-21-2017

Hurst, M. D., Mudd, S. M., Walcott, R., Attal, M., & Yoo, K. (2012). Using hilltop curvature to derive the spatial distribution of erosion rates. *Journal of Geophysical Research: Earth Surface*, *117*(F2). https://doi.org/10.1029/2011JF002057

James, M. R., & Robson, S. (2012). Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application: 3D SURFACES AND TOPOGRAPHY WITH A CAMERA. *Journal of Geophysical Research: Earth Surface*, *117*(F3), n/a-n/a. https://doi.org/10.1029/2011JF002289

James, M. R., Robson, S., d'Oleire-Oltmanns, S., & Niethammer, U. (2017). Optimising UAV topographic surveys processed with structure-from-motion: Ground control quality, quantity and bundle adjustment. *Geomorphology*, *280*, 51–66. https://doi.org/10.1016/j.geomorph.2016.11.021

Johnson, K., Nissen, E., Saripalli, S., Arrowsmith, J. R., McGarey, P., Scharer, K., et al. (2014). Rapid mapping of ultrafine fault zone topography with structure from motion. *Geosphere*, *10*(5), 969–986. https://doi.org/10.1130/GES01017.1

Johnstone, S. A., & Hilley, G. E. (2015). Lithologic control on the form of soil-mantled hillslopes. *Geology*, *43*(1), 83–86. https://doi.org/10.1130/G36052.1

Mériaux, A.-S., Van der Woerd, J., Tapponnier, P., Ryerson, F. J., Finkel, R. C., Lasserre, C., & Xu, X. (2012). The Pingding segment of the Altyn Tagh Fault (91°E): Holocene slip-rate determination from cosmogenic radionuclide dating of offset fluvial terraces. *Journal of Geophysical Research: Solid Earth*, *117*(B9). https://doi.org/10.1029/2012JB009289

Nash, D. B. (1980). Morphologic Dating of Degraded Normal Fault Scarps. *The Journal of Geology*, *88*(3), 353–360.

Riebe, C. S., Kirchner, J. W., Granger, D. E., & Finkel, R. C. (2001). Strong tectonic and weak climatic control of long-term chemical weathering rates. *Geology*, *29*(6), 511. https://doi.org/10.1130/0091-7613(2001)029<0511:STAWCC>2.0.CO;2

Roering, J. J. (2008). How well can hillslope evolution models "explain" topography? Simulating soil transport and production with high-resolution topographic data. *Geological Society of America Bulletin*, *120*(9–10), 1248–1262. https://doi.org/10.1130/B26283.1

Roering, Joshua J., Kirchner, J. W., & Dietrich, W. E. (1999). Evidence for nonlinear, diffusive sediment transport on hillslopes and implications for landscape morphology. *Water Resources Research*, *35*(3), 853–870. https://doi.org/10.1029/1998WR900090

Roering, Joshua J., Perron, J. T., & Kirchner, J. W. (2007). Functional relationships between denudation and hillslope form and relief. *Earth and Planetary Science Letters*, *264*(1–2), 245–258. https://doi.org/10.1016/j.epsl.2007.09.035

Scott, T. (2020). *Rock Traits from Machine Learning: Application to Rocky Fault Scarps* (Master Thesis). Arizona State University. Retrieved from https://keep.lib.asu.edu/items/158663

Sklar, L. S., & Dietrich, W. E. (2001). Sediment and rock strength controls on river incision into bedrock. *Geology*, *29*(12), 1087–1090. https://doi.org/10.1130/0091-7613(2001)029<1087:SARSCO>2.0.CO;2

Tucker, G. E., & Slingerland, R. (1997). Drainage basin responses to climate change. *Water Resources Research*, *33*(8), 2031–2047. https://doi.org/10.1029/97WR00409

Westoby, M. J., Brasington, J., Glasser, N. F., Hambrey, M. J., & Reynolds, J. M. (2012). 'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, *179*, 300–314. https://doi.org/10.1016/j.geomorph.2012.08.021

Whipple, K. X., & Tucker, G. E. (2002). Implications of sediment-flux-dependent river incision models for landscape evolution. *Journal of Geophysical Research: Atmospheres*, *107*(2), 3-1-3–20.

Xu, J., Arrowsmith, J. R., Chen, J., Schoenbohm, L. M., Li, T., Yuan, Z., & Owen, L. A. (2021). Evaluating young fluvial terrace riser degradation using a nonlinear transport model: Application to the Kongur Normal Fault in the Pamir, northwest China. *Earth Surface Processes and Landforms*, *46*(1), 280–295. https://doi.org/10.1002/esp.5022

APPENDIX A

JUPYTER NOTEBOOK OF IDEALIZED FAULT SCARPS SIMULATION

Fault Scarps Simulations

```
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import animation, cm
from matplotlib.cm import ScalarMappable
from landlab import RasterModelGrid
from landlab.components import DepthDependentDiffuser, ExponentialWea
thererIntegrated
import copy
```

```
# Parameters
grid_size = (100, 200)
D = 0.002  # m^2/year
dt = 100  # years
fr = 100 #number of frames
nt = 10 #number of time steps per frame
```

```
# Function to initialize grid and elevation
def init_grid_elevation(soil_depth_initial):
    grid = RasterModelGrid(grid_size, xy_spacing=1)
    topographic_elevation = grid.add_zeros('topographic__elevation',
at='node')
    topographic_elevation += 0  # add small noise to elevation
    soil_depth = grid.add_zeros('soil__depth', at='node')

    soil_depth += soil_depth_initial
    spr = grid.add_zeros('soil_production__rate', at='node')

    # Create a bedrock elevation field
    bedrock_elevation = grid.add_zeros('bedrock__elevation', at='node
')
    bedrock_elevation[:] = topographic_elevation - soil_depth

    # set boundary conditions
    grid.set_closed_boundaries_at_grid_edges(True, True, True, True)

    return grid, topographic_elevation, soil_depth, bedrock_elevation
,       spr

# Initialize grids and elevations for both plots
grid_PL, topographic_elevation_PL, soil_depth_PL, bedrock_elevation_P
L, spr_PL = init_grid_elevation(0)
grid_TL, topographic_elevation_TL, soil_depth_TL, bedrock_elevation_T
```

```
L, spr_TL = init_grid_elevation(30)

initial_topographic_elevation_PL = copy.deepcopy(topographic_elevatio
n_PL)
initial_bedrock_elevation_PL = copy.deepcopy(bedrock_elevation_PL)
initial_soil_depth_PL = copy.deepcopy(soil_depth_PL)
initial_spr_PL = copy.deepcopy(spr_PL)

initial_topographic_elevation_TL = copy.deepcopy(topographic_elevatio
n_TL)
initial_bedrock_elevation_TL = copy.deepcopy(bedrock_elevation_TL)
initial_soil_depth_TL = copy.deepcopy(soil_depth_TL)
initial_spr_TL = copy.deepcopy(spr_TL)

# Create DepthDependentDiffuser components
eroder_PL = DepthDependentDiffuser(grid_PL, linear_diffusivity=D, soi
l_transport_decay_depth=0.12)
weatherer_PL = ExponentialWeathererIntegrated(grid_PL, soil_productio
n__maximum_rate=0.0002, soil_production__decay_depth=0.5)

eroder_TL = DepthDependentDiffuser(grid_TL, linear_diffusivity=D, soi
l_transport_decay_depth=0.12)
weatherer_TL = ExponentialWeathererIntegrated(grid_TL, soil_productio
n__maximum_rate=0.0002, soil_production__decay_depth=0.5)

fault_y = int(grid_size[1] / 2)
```

```
def plot_3D_animation(stack='horizontal',event='Single_Event'):
    if stack == 'horizontal':
        fig, (ax_PL, ax_TL) = plt.subplots(nrows=1, ncols=2, figsize=
(14, 7), subplot_kw={'projection': '3d'})
    elif stack == 'vertical':
        fig, (ax_PL, ax_TL) = plt.subplots(nrows=2, ncols=1, figsize=
(7, 14), subplot_kw={'projection': '3d'})
        plt.subplots_adjust(hspace=-0.4)
    # Define the displacement function
    def displacement(grid, bedrock_elevation, topographic_elevation,
soil_depth, spr, eroder, weatherer, nt, dt, t, event=event):
        if event == 'Single_Event':
            if t==0:
                topographic_elevation[grid.node_x > fault_y+15*np.sin
(grid.node_y/(np.pi*2))] = 30
                bedrock_elevation[:] = topographic_elevation - soil_d
epth
                for i in range(nt):
                    weatherer.run_one_step(dt)
                    eroder.run_one_step(dt)
```

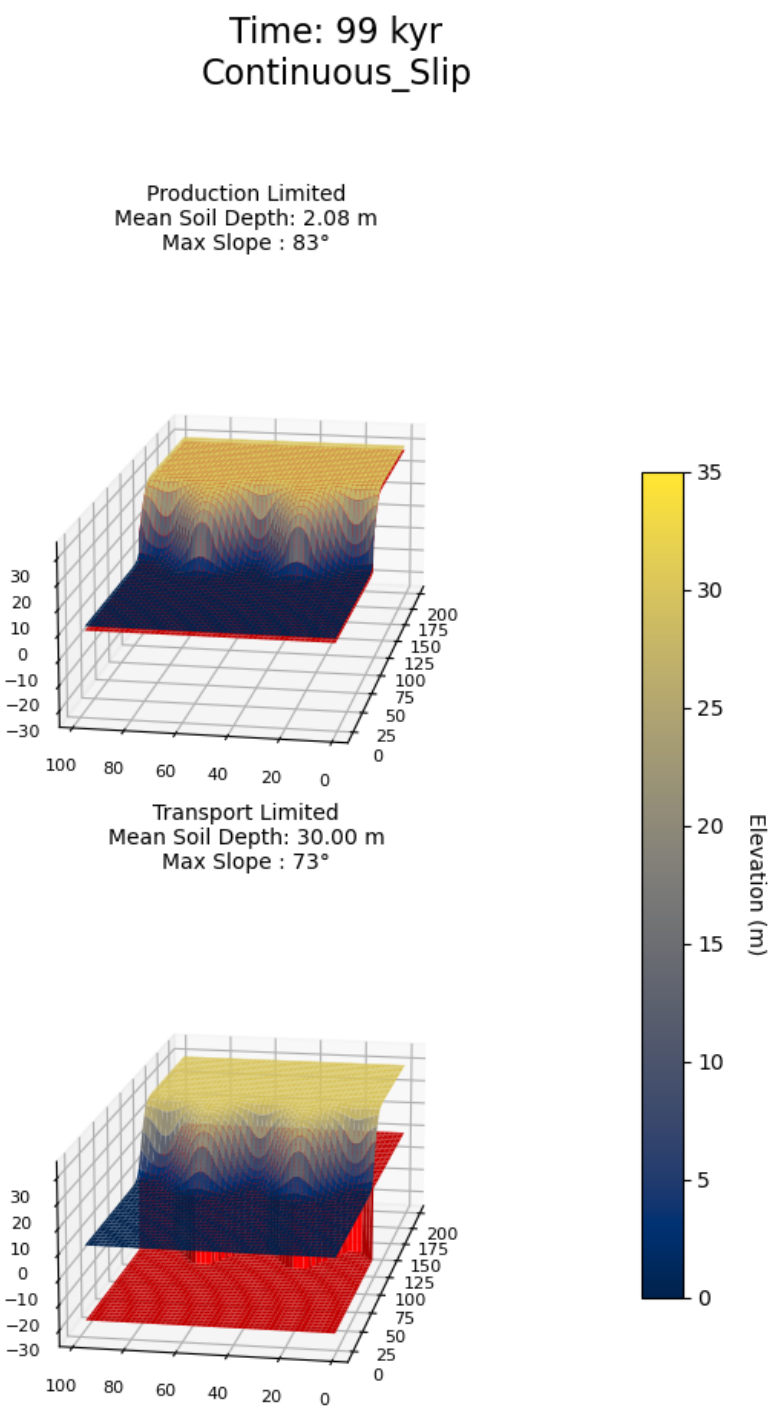
```
            else:
                for i in range(nt):
                    weatherer.run_one_step(dt)
                    eroder.run_one_step(dt)

        elif event =='Multiple_Events':
            if t%10 == 0:
                topographic_elevation[grid.node_x > fault_y+15*np.sin
(grid.node_y/(np.pi*2))] += 3
                bedrock_elevation[:] = topographic_elevation - soil_d
epth
                for i in range(nt):
                    weatherer.run_one_step(dt)
                    eroder.run_one_step(dt)

            else:
                for i in range(nt):
                    weatherer.run_one_step(dt)
                    eroder.run_one_step(dt)
        elif event == 'Continuous_Slip':
            for i in range(nt):
                topographic_elevation[grid.node_x > fault_y+15*np.sin
(grid.node_y/(np.pi*2))] += 0.03
                bedrock_elevation[:] = topographic_elevation - soil_d
epth
                weatherer.run_one_step(dt)
                eroder.run_one_step(dt)
        return grid, bedrock_elevation, topographic_elevation, soil_d
epth, spr


    # Update function for animation
    def update(t):
        global topographic_elevation_PL, bedrock_elevation_PL, soil_d
epth_PL, spr_PL
        global topographic_elevation_TL, bedrock_elevation_TL, soil_d
epth_TL, spr_TL

        ax_PL.clear()
        ax_TL.clear()

        # Reset the topographic elevation to the initial state if it'
s the first frame of a new loop
        if t % fr == 0:
            topographic_elevation_PL[:] = initial_topographic_elevati
on_PL
            soil_depth_PL[:] = initial_soil_depth_PL
            bedrock_elevation_PL[:] = topographic_elevation_PL - soil
_depth_PL
```

```
            spr_PL[:] = initial_spr_PL

            topographic_elevation_TL[:] = initial_topographic_elevati
on_TL
            soil_depth_TL[:] = initial_soil_depth_TL
            bedrock_elevation_TL[:] = topographic_elevation_TL - soil
_depth_TL
            spr_TL[:] = initial_spr_TL

        # Update the topographic elevation using the DepthDependentDi
ffuser component
        displacement(grid_PL, bedrock_elevation_PL, topographic_eleva
tion_PL, soil_depth_PL, spr_PL, eroder_PL, weatherer_PL, nt, dt,t,eve
nt=event)
        displacement(grid_TL, bedrock_elevation_TL, topographic_eleva
tion_TL, soil_depth_TL, spr_TL, eroder_TL, weatherer_TL, nt, dt,t,eve
nt=event)

        def plot_surface(ax, grid, topographic_elevation, bedrock_ele
vation, title, vmin, vmax):
            x = grid.node_x.reshape(grid.shape)
            y = grid.node_y.reshape(grid.shape)
            z_bedrock = bedrock_elevation.reshape(grid.shape)
            x_core, y_core, BRz_core = x[1:-1, 1:-1], y[1:-1, 1:-1],
z_bedrock[1:-1, 1:-1]
            ax.plot_surface(x_core, y_core, BRz_core, color='r', labe
l='Bedrock', vmin=(min(initial_topographic_elevation_PL) - 2))

            z_topographic = topographic_elevation.reshape(grid.shape)
            z_core = z_topographic[1:-1, 1:-1]
            dz_dy, dz_dx = np.gradient(z_core)
            slope = np.sqrt(dz_dy**2 + dz_dx**2)
            slope_deg = np.arctan(slope) * (180 / np.pi)
            max_slope_deg = np.max(slope_deg)

            highlight_colors = np.zeros((*slope_deg.shape, 4))
            max_slope_mask = slope_deg > 0.99 * max_slope_deg
            highlight_colors[max_slope_mask] = [0, 0, 0, 0]  # Set co
lor to black and opacity to 1 for maximum slope areas

            ax.plot_surface(x_core, y_core, z_core, cmap=cm.cividis,
label='Ground', vmin=vmin, vmax=vmax)
            ax.plot_surface(x_core, y_core, z_core, facecolors=highli
ght_colors, label='Ground', vmin=vmin, vmax=vmax)
            ax.set_zlim(-35, 35)
            ax.view_init(15, -170)
            ax.set_aspect('equal')
            ax.tick_params(axis='both', which='major', labelsize=8, p
ad=5)
```

```
            ax.set_title(title + f"\nMax Slope : {round(max_slope_deg
)}°",fontsize=10)

        plot_surface(ax_PL, grid_PL, topographic_elevation_PL, bedroc
k_elevation_PL, f"Production Limited\nMean Soil Depth: {np.mean(soil_
depth_PL):.2f} m", global_min, global_max)
        plot_surface(ax_TL, grid_TL, topographic_elevation_TL, bedroc
k_elevation_TL, f"Transport Limited\nMean Soil Depth: {np.mean(soil_d
epth_TL):.2f} m", global_min, global_max)
        fig.suptitle(f"Time: {t} kyr\n{event}", fontsize=16,y=0.9)

    def add_shared_colorbar(fig, vmin, vmax, cmap, stack=stack):
        norm = plt.Normalize(vmin, vmax)
        mappable = ScalarMappable(cmap=cmap, norm=norm)
        mappable.set_array([])

        if stack == 'horizontal':
            cbar = fig.colorbar(mappable, ax=[ax_PL, ax_TL], pad=-0.1
, shrink=0.5, location='bottom')
            cbar.ax.set_ylabel('Elevation (m)', rotation=0, labelpad=
40)
            cbar.ax.xaxis.set_label_position('top')
        elif stack == 'vertical':
            cbar = fig.colorbar(mappable, ax=[ax_PL, ax_TL], pad=0.1,
shrink=0.5)
            cbar.ax.set_ylabel('Elevation (m)', rotation=270, labelpa
d=20)

    # Define global minimum and maximum for colormap
    global_min = 0
    global_max = 35
    # Create colorbar
    add_shared_colorbar(fig, global_min, global_max, cm.cividis, stac
k=stack)
    # Create the animation
    ani = animation.FuncAnimation(fig, update, frames=fr, interval=10
0, repeat=True)
    return fig, ani
```

```
# Call the function to create the plot
fig, ani = plot_3D_animation(stack='vertical',event='Continuous_Slip'
)

# Save the animation as a GIF
ani_path = './animation/'
ani.save(ani_path+'Animation_curved_inward_cont.gif', writer='pillow'
, fps=5)
```

```
# Show the animation
#plt.show()
```



APPENDIX B

JUPYTER NOTEBOOK OF FAULT SCARPS DATING USING DIFFUSION

Import Base Modules

```
import numpy as np
import pandas as pd
import copy
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
import math
from matplotlib.colorbar import Colorbar

import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm

from osgeo import gdal

from landlab import RasterModelGrid
from landlab.io.esri_ascii import write_esri_ascii, read_esri_ascii
from landlab.plot import imshow_grid, imshowhs_grid
from landlab.components import NormalFault,LinearDiffuser,DepthDepend
entDiffuser,TaylorNonLinearDiffuser,DepthDependentTaylorDiffuser
```

Custom Code

```
#Slope-integrated Exponential Weatherer

#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Created on Fri Apr  8 08:32:48 2016.

@author: RCGlade
@author: dylanward
Integrated version created by D. Ward on Tue Oct 27 2020
Slope-integrated version created by A. Hafiz on Wed Nov 30 2022
"""

import numpy as np

from landlab import Component


class ExponentialWeathererIntegrated(Component):

    r"""
    This component implements exponential weathering of bedrock on
    hillslopes. Uses exponential soil production function in the styl
e
    of Ahnert (1976).

    Consider that :math:`w_0` is the maximum soil production rate and
```

```
    that :math:`d^*` is the characteristic soil production depth. The
    soil production rate :math:`w` is given as a function of the soil
    depth :math:`d`,

    .. math::

        w = w_0 \exp{-\frac{d}{d^*}} \;.

    The `ExponentialWeathererIntegrated` uses the analytical solution
    for the amount of soil produced by an exponential weathering
    function over a timestep dt, and returns both the thickness of
    bedrock weathered and the thickness of soil produced. The solutio
n
    accounts for the reduction in rate over the timestep due to the
    increasing depth. This enables accuracy over arbitrarily large
    timesteps, and better compatiblity with the `run_one_step()`
    interface.

    Compared to 'ExponentialWeatherer', upon which it is based...

    - This maintains the field I/O behavior of the original, but adds
      new return fields for the weathered thickness and soil produced
      thickness.
    - Density adjustments are needed inside the integral and the
      density ratio is intialized on instantiation. The default value
      of 1.0 assumes no change in density.
    - Returns both weathered depth of bedrock and produced depth of
      soil over the timestep.
    - The primary `soil__depth` field that is input is NOT updated by
      the component.

    This is left as an exercise for the model driver, as different
    applications may want to integrate soil depth and weathering in
    different sequences among other processes.

    - SHOULD maintain drop-in compatiblity with the plain
      :py:class:`ExponentialWeatherer <landlab.components.Exponential
Weatherer>`,
      just import and instantiate this one instead and existing code
      should work with no side effects other than the creation of the
      two additional (zeros) output fields.

    Examples
    --------
    >>> import numpy as np
    >>> from landlab import RasterModelGrid
    >>> from landlab.components import ExponentialWeathererIntegrated
    >>> mg = RasterModelGrid((5, 5))
    >>> soilz = mg.add_zeros("soil__depth", at="node")
```

```
    >>> soilrate = mg.add_ones("soil_production__rate", at="node")
    >>> expw = ExponentialWeathererIntegrated(mg)
    >>> dt = 1000
    >>> expw.run_one_step(dt)
    >>> np.allclose(mg.at_node['soil_production__rate'][mg.core_nodes
], 1.)
    True
    >>> np.allclose(mg.at_node['soil_production__dt_produced_depth'][
mg.core_nodes], 6.9088)
    True

    References
    ----------
    **Required Software Citation(s) Specific to this Component**

    Barnhart, K., Glade, R., Shobe, C., Tucker, G. (2019). Terrainben
to 1.0: a
    Python package for multi-model analysis in long-term drainage bas
in
    evolution. Geoscientific Model Development  12(4), 1267--1297.
    https://dx.doi.org/10.5194/gmd-12-1267-2019

    **Additional References**

    Ahnert, F. (1976). Brief description of a comprehensive three-dim
ensional
    process-response model of landform development Z. Geomorphol. Sup
pl.  25,
    29 - 49.

    Armstrong, A. (1976). A three dimensional simulation of slope for
ms.
    Zeitschrift für Geomorphologie  25, 20 - 28.

    """

    _name = "ExponentialWeathererIntegrated"

    _unit_agnostic = True

    _cite_as = """
    @article{barnhart2019terrain,
      author = {Barnhart, Katherine R and Glade, Rachel C and Shobe,
Charles M and Tucker, Gregory E},
      title = {{Terrainbento 1.0: a Python package for multi-model an
alysis in long-term drainage basin evolution}},
      doi = {10.5194/gmd-12-1267-2019},
      pages = {1267---1297},
      number = {4},
```

```
      volume = {12},
      journal = {Geoscientific Model Development},
      year = {2019},
    }
    """

    _info = {
        "soil__depth": {
            "dtype": float,
            "intent": "in",
            "optional": False,
            "units": "m",
            "mapping": "node",
            "doc": "Depth of soil or weathered bedrock",
        },
        "surface__slope": {
            "dtype": float,
            "intent": "out",
            "optional": False,
            "units": "radians",
            "mapping": "node",
            "doc": "surface slope at node",
        },
        "soil_production__rate": {
            "dtype": float,
            "intent": "out",
            "optional": False,
            "units": "m/yr",
            "mapping": "node",
            "doc": "rate of soil production at nodes",
        },
        "soil_production__dt_produced_depth": {
            "dtype": float,
            "intent": "out",
            "optional": False,
            "units": "m",
            "mapping": "node",
            "doc": "thickness of soil produced at nodes over time dt"
,
        },
        "soil_production__dt_weathered_depth": {
            "dtype": float,
            "intent": "out",
            "optional": False,
            "units": "m",
            "mapping": "node",
            "doc": "thickness of bedrock weathered at nodes over time
dt",
        },
```

```
    }

    def __init__(
        self,
        grid,
        soil_production__maximum_rate=1.0,
        soil_production__decay_depth=1.0,
        soil_production__expansion_factor=1.0,
    ):
        """
        Parameters
        ----------
        grid: ModelGrid
            Landlab ModelGrid object
        soil_production__maximum_rate : float
            Maximum weathering rate for bare bedrock
        soil_production__decay_depth : float
            Characteristic weathering depth
        soil_production__expansion_factor : float
            Expansion ratio of regolith (from relative densities of
            rock and soil)
        """
        super().__init__(grid)

        # Store grid and parameters

        self._wstar = soil_production__decay_depth
        self._w0 = soil_production__maximum_rate
        self._fexp = soil_production__expansion_factor

        # Create fields:
        # soil depth
        self._depth = grid.at_node["soil__depth"]

        # surface slope
        if "surface__slope" in grid.at_node:
            self._slope = grid.at_node["surface__slope"]
        else:
            self._slope = grid.add_zeros("surface__slope", at="node")

        # weathering rate
        if "soil_production__rate" in grid.at_node:
            self._soil_prod_rate = grid.at_node["soil_production__rat
e"]
        else:
            self._soil_prod_rate = grid.add_zeros("soil_production__r
ate", at="node")

        # soil produced total over dt
```

```
        if "soil_production__dt_produced_depth" in grid.at_node:
            self._soil_prod_total = grid.at_node["soil_production__dt
_produced_depth"]
        else:
            self._soil_prod_total = grid.add_zeros(
                "soil_production__dt_produced_depth", at="node"
            )

        # bedrock weathering total over dt
        if "soil_production__dt_weathered_depth" in grid.at_node:
            self._rock_weathered_total = grid.at_node[
                "soil_production__dt_weathered_depth"
            ]
        else:
            self._rock_weathered_total = grid.add_zeros(
                "soil_production__dt_weathered_depth", at="node"
            )


    def calc_surface_slope(self):
        """Calculate surface slope."""
        self._slope = self._grid.calc_slope_at_node()


    def calc_soil_prod_rate(self,dt):
        """Calculate soil production rate."""
        # analytical solution
        self._soil_prod_rate[self._grid.core_nodes] = self._wstar * n
p.log(
            (
                self._fexp
                * self._w0/np.cos(self._slope[self._grid.core_nodes])
* np.exp(-self._depth[self._grid.core_nodes] / self._wstar)
                * dt
                / self._wstar
            )
            + 1
        )/dt


    def _calc_dt_production_total(self, dt):
        """Calculate integrated production over 1 timestep dt"""
        # analytical solution
        self._soil_prod_total[self._grid.core_nodes] = self._wstar *
np.log(
            (
                self._fexp
                * self._w0/np.cos(self._slope[self._grid.core_nodes])
* np.exp(-self._depth[self._grid.core_nodes] / self._wstar)
```

```
                * dt
                / self._wstar
            )
            + 1
        )
        # and back-convert to find rock thickness converted over the
timestep:
        self._rock_weathered_total[self._grid.core_nodes] = (
            self._soil_prod_total[self._grid.core_nodes] / self._fexp
        )

    def run_one_step(self, dt=0):
        """
        Parameters
        ----------
        dt: float
            Used only for compatibility with standard run_one_step.
            If dt is not provided, the default of zero maintains back
ward compatibility
        """
        self.calc_surface_slope()
        self.calc_soil_prod_rate(dt)
        self._calc_dt_production_total(dt)


    @property
    def maximum_weathering_rate(self):
        """Maximum rate of weathering (m/yr)."""
        return self._w0

    @maximum_weathering_rate.setter
    def maximum_weathering_rate(self, new_val):
        if new_val <= 0:
            raise ValueError("Maximum weathering rate must be positiv
e.")
        self._w0 = new_val
```

```
from landlab import RasterModelGrid
import numpy as np

class LandlabGridGenerator:
    def __init__(self, grid_shape, xy_spacing):
        self.grid_shape = grid_shape
        self.xy_spacing = xy_spacing
```

```
    def create_grid(self):
        grid = RasterModelGrid(self.grid_shape, self.xy_spacing)

        # Initialize the fields with initial values
        topographic_elevation = np.zeros(grid.number_of_nodes)
        bedrock_elevation = np.zeros(grid.number_of_nodes)
        soil_depth = np.zeros(grid.number_of_nodes)
        soil_production_rate = np.zeros(grid.number_of_nodes)

        # Add the fields to the grid
        grid.add_field("node", "topographic__elevation", topographic_
elevation, units="m")
        grid.add_field("node", "bedrock__elevation", bedrock_elevatio
n, units="m")
        grid.add_field("node", "soil__depth", soil_depth, units="m")
        grid.add_field("node", "soil_production__rate", soil_producti
on_rate, units="m/yr")

        return grid
```

```
def interactive_view(view=bool,backend='module://ipympl.backend_nbagg
'):
    if view==True:
        matplotlib.use(backend)
    if view==False:
        matplotlib.use('module://matplotlib_inline.backend_inline')
```

```
def find_intersection(line1,line2):
    # Find the intersection point
    x_intersect = np.roots(line1 - line2)
    y_intersect = line1(x_intersect)
    xy=(float(x_intersect),float(y_intersect))
    # return intersection as list
    return (xy)
```

```
def get_n_EQs(number):
    divisors = []
    for i in range(1, number + 1):
        if number % i == 0:  # Check if the remainder of the division
is 0 (integer result)
```

```
            divisors.append(i)
    return divisors
```

```
def plot3d(grid,data,data2=None,overlap=False,cmap='pink',vmin=None,v
max=None,xlim=None,ylim=None,zlim=None,
           cmap2='pink',vmin2=None,vmax2=None):
    xy_size = grid.dx
    xlen = grid.shape[1]
    ylen = grid.shape[0]
    x0 = grid.xy_of_lower_left[0]
    y0 = grid.xy_of_lower_left[1]
    x1 = x0+(xlen*xy_size)
    y1 = y0+(ylen*xy_size)
    X = np.arange(x0,x1,xy_size)
    Y = np.arange(y0,y1,xy_size)
    X,Y = np.meshgrid(X,Y)
    Z = data.reshape(grid.shape[0],grid.shape[1])
    fig3d = plt.figure()
    ax3d = fig3d.add_subplot(111,projection='3d')
    ax3d.plot_surface(X,Y,Z,cmap=cmap,vmin=vmin,vmax=vmax)
    ax3d.set_xlim(xlim)
    ax3d.set_ylim(ylim)
    ax3d.set_zlim(zlim)
    if overlap == True:
        Z2 = data2.reshape(grid.shape[0],grid.shape[1])
        ax3d.plot_surface(X,Y,Z2,cmap=cmap2,vmin=vmin2,vmax=vmax2)
    plt.show()
```

Set up model from DEM

Set up grid

```
interactive_view(True)
```

```
#import DEM
from landlab.io import read_esri_ascii
(grid_dem,grid_dem.at_node["topographic__elevation"]) = read_esri_asc
ii("./input/DEM.asc")
grid_dem
```

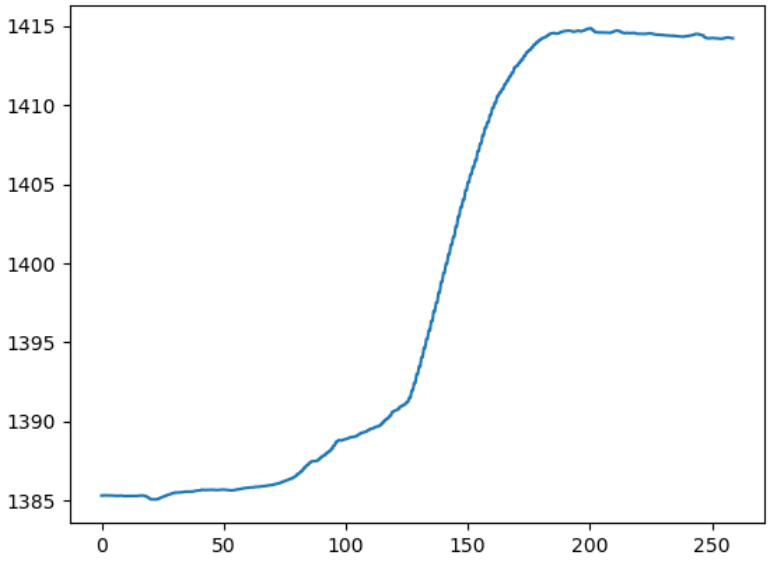
```
RasterModelGrid((208, 518), xy_spacing=(0.5, 0.5), xy_of_lower_left=(62
0834.5636909426, 4515006.6822318127))
```

```
# create grid with the shape of DEM
grid_shape = grid_dem.shape
xy_spacing = grid_dem.dx

#create new grid
grid_generator = LandlabGridGenerator(grid_shape, xy_spacing)
grid = grid_generator.create_grid()
grid.fields()
```

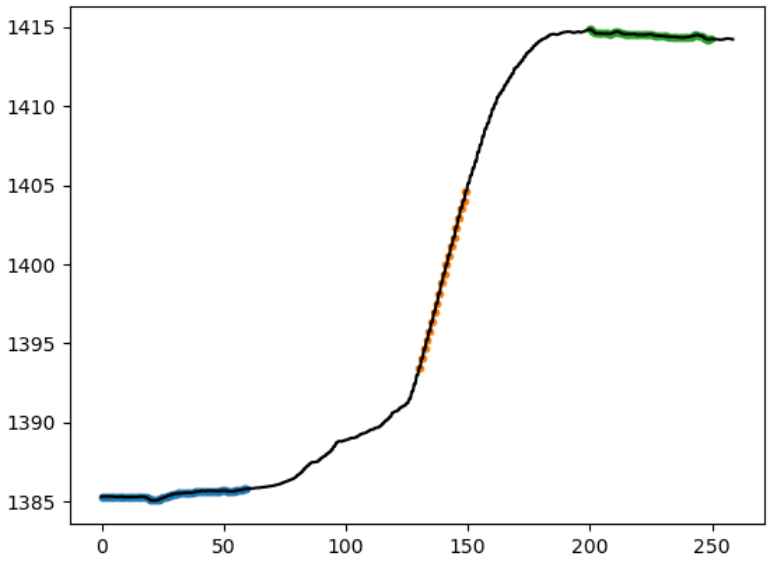
```
{'at_node:bedrock__elevation',
 'at_node:soil__depth',
 'at_node:soil_production__rate',
 'at_node:topographic__elevation'}
```

Generate swath and get initial conditions

```
#generate arrays containing node distribution on x an y axis
x_nodes = grid.node_x.reshape(grid.shape)[0]
y_nodes = grid.node_y.reshape(grid.shape).transpose()[0]

#create dem swath and plot it
dem_swath = []
for i in x_nodes:
    dem_swath.append(np.mean(grid_dem.at_node["topographic__elevation
"][grid.node_x == i]))
dem_swath = np.asarray(dem_swath)
plt.plot(x_nodes,dem_swath)
plt.show()
```

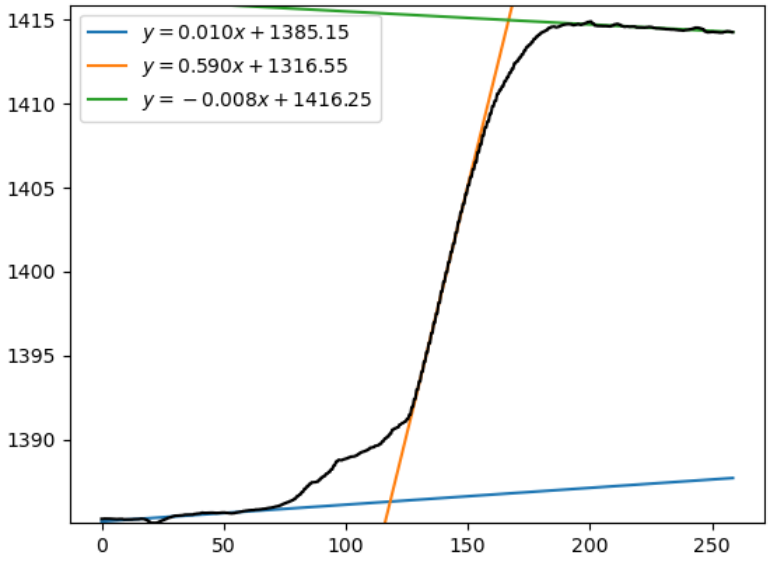


```
#pick a surface to do linear regression on
base = [0,60]
fault = [130,150]
top = [200,250]
x_div = [base,fault,top]
#plot the selected points
x_div_range = []
zx_div_range = []
for i in range(len(x_div)):
    x_div_range.append(np.arange(x_div[i][0],x_div[i][1],xy_spacing))
    zx_div = []
    for j in (x_div_range[i]/xy_spacing):
        zx_div.append(dem_swath[int(j)])
    zx_div_range.append(zx_div)
    plt.plot(x_div_range[i],zx_div_range[i],'.')
plt.plot(x_nodes,dem_swath,'black')
plt.show()
```



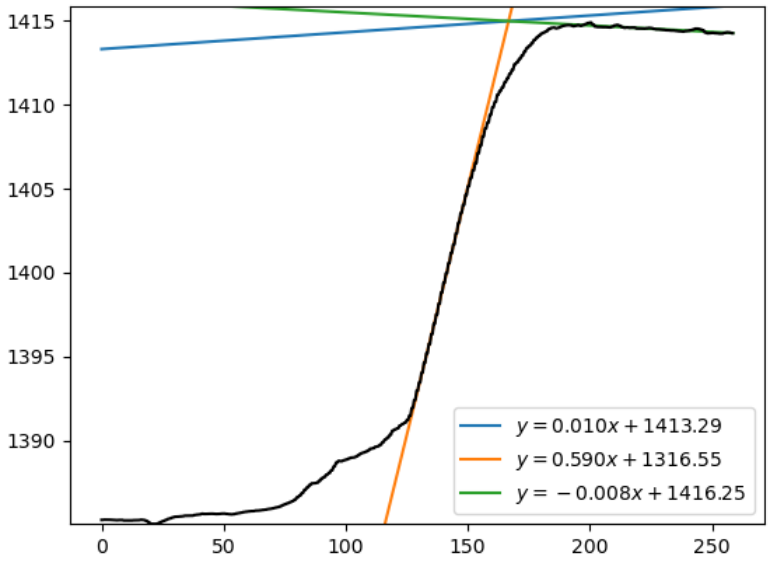

```
#fit points using linear regression
c_fit = []
fx = []
m = []
c = []
for i in range(len(x_div)):
    c_fit.append(np.polyfit(x_div_range[i],zx_div_range[i],deg=1))
    fx.append(np.poly1d(c_fit[i]))
    m.append(c_fit[i][0])
```

```
    c.append(c_fit[i][1])
#plot the fit
ax = plt.figure().add_subplot()
for i in range(len(x_div)):
    plt.plot(x_nodes,fx[i](x_nodes),'-',label=f'$y = {fx[i][1]:.3f}x
{fx[i][0]:+.2f}$')
plt.plot(x_nodes,dem_swath,'black')
ax.set_ylim(min(dem_swath),max(dem_swath)+1)
plt.legend()
plt.show()
```




```
#find intersection of two lines
int1 = find_intersection(fx[0],fx[1])
int2 = find_intersection(fx[1],fx[2])
#find c for base plane before faulting
c_init = copy.deepcopy(c)
c_init[0]=int2[1]-(int2[0]*m[0]) #c=y-xm
#update fx for base plane
fx_init = copy.deepcopy(fx)
fx_init[0][0] = c_init[0]
#find intersection between base init and top
intf = find_intersection(fx_init[0], fx_init[2])
#plot the initial surface
ax = plt.figure().add_subplot()
for i in range(len(x_div)):
    plt.plot(x_nodes,fx_init[i](x_nodes),'-',label=f'$y = {fx_init[i]
[1]:.3f}x {fx_init[i][0]:+.2f}$')
```

```
plt.plot(x_nodes,dem_swath,'black')
ax.set_ylim(min(dem_swath),max(dem_swath)+1)
plt.legend()
plt.show()
```




```
#create variable for slopes
Sb = m[0]
Sf = m[1]
St = m[2]
#find throw height and fault width
throw_height = int2[1] - int1[1]
fault_width = int2[0] - int1[0]
#Slope parameters
print("Base slope: ",math.degrees(math.atan(Sb)),"degree")
print("Fault slope: ",math.degrees(math.atan(Sf)),"degree")
print("Top slope: ",math.degrees(math.atan(St)),"degree")
#Fault parameters
print("Throw height: ",throw_height)
print("Fault width: ",fault_width)
```

```
Base slope:  0.5725017866214247 degree
Fault slope:  30.55995233957674 degree
Top slope:  -0.4441671624365435 degree
Throw height:  28.628741023135262
Fault width:  48.48587244812791
```

```
#create a list of number of nodes to intersection points
int_nodes_ = [0,1,2,3]
int_nodes_[0] = 0
int_nodes_[1] = int1[0]//grid.dx
int_nodes_[2] = int2[0]//grid.dx
int_nodes_[3] = grid.shape[1]
#generate arrays of nodes for each surface
base_nodes_ = np.arange(int_nodes_[0]*grid.dx,int_nodes_[1]*grid.dx,g
rid.dx)
fault_nodes_ = np.arange(int_nodes_[1]*grid.dx,int_nodes_[2]*grid.dx,
grid.dx)
top_nodes_ = np.arange(int_nodes_[2]*grid.dx,int_nodes_[3]*grid.dx,gr
id.dx)
model_nodes_init_ = (np.append(base_nodes_,fault_nodes_),fault_nodes_
,top_nodes_)
model_nodes_end_ = (base_nodes_,fault_nodes_,top_nodes_)

#find the number of time steps and possible EQs number
nt_max = int(int_nodes_[2] - int_nodes_[1]) #maximum number of time s
teps
dhdt = Sf*grid.dx #vertical changes per time step
EQs_num = get_n_EQs(nt_max)
print("Maximum number of time steps: ",nt_max)
print("Vertical changes per time step: ",dhdt)
print(f"The number of EQs that can be modeled for nt_max {nt_max} are
: {EQs_num}")
```

```
Maximum number of time steps:  97
Vertical changes per time step:  0.295227656817
The number of EQs that can be modeled for nt_max 97 are: [1, 97]
```

Rescale DEM (if needed)

```
#find the required dx for certain number of time steps
desired_nt = 100
dx_required = fault_width/desired_nt
dhdt_new = Sf*dx_required #vertical changes per time step
EQs_num_new = get_n_EQs(desired_nt)
print(f"The xy spacing needed for {desired_nt} timestep is: {dx_requi
red}")
print("Vertical changes per time step: ",dhdt_new)
print(f"The number of EQs that can be modeled for nt_max {desired_nt}
are: {EQs_num_new}")
```

```
The xy spacing needed for 100 timestep is: 0.4848587244812791
Vertical changes per time step:  0.286287410231
The number of EQs that can be modeled for nt_max 100 are: [1, 2, 4, 5,
10, 20, 25, 50, 100]
```

```
dx_rescaled = 0.485
```

```
gdal.Warp('./input/DEM_rescaled_{desired_nt}nt.asc','./input/DEM.asc'
,xRes=dx_rescaled,yRes=dx_rescaled)
```

```
<osgeo.gdal.Dataset; proxy of <Swig Object of type 'GDALDatasetShadow *
' at 0x16377eac0> >
```

Generate initial model

```
#import DEM rescaled
from landlab.io import read_esri_ascii
(grid_dem_rescaled,grid_dem_rescaled.at_node["topographic__elevation"
]) = read_esri_ascii('./input/DEM_rescaled_{desired_nt}nt.asc')

z_dem = grid_dem_rescaled.at_node["topographic__elevation"]

# create grid with the shape of DEM
grid_shape_rescaled = grid_dem_rescaled.shape
xy_spacing_rescaled = grid_dem_rescaled.dx
#create new grid
grid_generator = LandlabGridGenerator(grid_shape_rescaled, xy_spacing
_rescaled)
mg = grid_generator.create_grid()
mg.set_closed_boundaries_at_grid_edges(right_is_closed=True,top_is_cl
osed=True,left_is_closed=True,bottom_is_closed=True)
mg
```

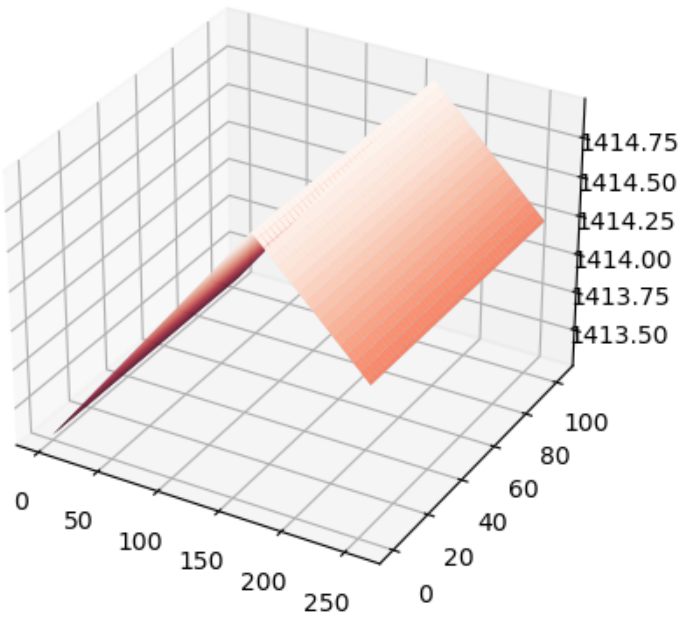
```
RasterModelGrid((214, 534), xy_spacing=(0.48499999999999999, 0.48499999
999999999), xy_of_lower_left=(0.0, 0.0))
```

```
#set up surface divider
div_init = (intf[0]//mg.dx)*mg.dx
div_end = (int1[0]//mg.dx)*mg.dx
n_faults = (div_init-div_end)/mg.dx
end_nodes = mg.shape[1]*mg.dx - (mg.dx/2)
n_faults
```

```
100.0
```

```
#generate elevation values
x_base = np.arange(0,div_init,mg.dx)
x_top = np.arange(div_init,end_nodes,mg.dx)
z_base = fx_init[0](x_base)
z_top = fx_init[2](x_top)
z_1d = np.append(z_base,z_top)
#reshape elevation values to 2D array
z2d = np.tile(z_1d,(mg.shape[0],1)).flatten()
#link variable to field names
z = mg.add_zeros("topographic__elevation",at="node",clobber=True)
H = mg.add_zeros("soil__depth",at="node",clobber=True)
BRz = mg.add_zeros("bedrock__elevation",at="node",clobber=True)
```

```
#assign values to fields
z[:] = z2d
H[:] += 0
BRz[:] = z-H

#plot initial elevations
plot3d(mg,z,BRz,True,cmap="Blues_r",cmap2="Reds_r")
```



Set up geomorphic processes (soil production and soil transport)

```
#Exponential Weatherer
def weatherer_mode(mg,w0,Hp=0.5,ef=1):
    weatherer = ExponentialWeathererIntegrated(mg,soil_production__ma
ximum_rate=w0,soil_production__decay_depth=Hp,soil_production__expans
ion_factor=ef)
    return weatherer

#Diffuser mode
def diffuser_mode(mg,k,Ht = 0.12,Sc = 1):
    ld = LinearDiffuser(mg, linear_diffusivity=k)
    td = TaylorNonLinearDiffuser(mg, linear_diffusivity=k,slope_crit=
Sc,dynamic_dt=True)
    ddld = DepthDependentDiffuser(mg,linear_diffusivity=k,soil_transp
```

```
ort_decay_depth=Ht)
    ddtd = DepthDependentTaylorDiffuser(mg,soil_transport_velocity=k,
slope_crit=Sc,soil_transport_decay_depth=Ht,dynamic_dt=True)
    return [ld,td,ddld,ddtd]
```

Set up tectonic displacements

```
#Horizontal Displacement Setup
def shift_elevation(grid, layers_1d, start_distance, end_distance, sh
ift_distance):
    layers = layers_1d.reshape(grid.shape)

    start_column = int(start_distance / grid.dx)
    end_column = int(end_distance / grid.dx)
    shift_amount = int(shift_distance / grid.dx)

    wrapped_columns = layers[:, end_column - shift_amount:end_column]
.copy()
    layers[:, start_column:end_column - shift_amount] = layers[:, sta
rt_column + shift_amount:end_column]
    layers[:, end_column - shift_amount:end_column] = wrapped_columns

    return layers.flatten()
```

```
#Vertical Displacement Setup
dhdt = Sf*mg.dx
x = x_top[0]
y = mg.shape[0]*mg.dx
nf = NormalFault(
    mg,
    faulted_surface=['topographic__elevation'],
    fault_throw_rate_through_time=(('time', [0]), ('rate', [-dhdt])),
    fault_dip_angle=90.0,
    fault_trace=(('x1', x), ('y1', 0), ('x2', x), ('y2', y)),
    include_boundaries=True,)
```

Initiate model based on displacement mode

```
t_range = [50000,80000,90000,100000,110000,120000,150000,200000]
k_range = [0.0005,0.001,0.0015,0.002,0.0025,0.003,0.005,0.0065,0.0075
,0.0085,0.01,0.011,0.015]
print("k_range: ",k_range)
print("t_range: ",t_range)
```

```
k_range:  [0.0005, 0.001, 0.0015, 0.002, 0.0025, 0.003, 0.005, 0.0065,
0.0075, 0.0085, 0.01, 0.011, 0.015]
t_range:  [50000, 80000, 90000, 100000, 110000, 120000, 150000, 200000]
```

```
#Vertical Displacement Setup
dhdt = Sf*mg.dx
x = x_top[0]
y = mg.shape[0]*mg.dx
nf = NormalFault(
    mg,
    faulted_surface=['topographic__elevation'],
    fault_throw_rate_through_time=(('time', [0]), ('rate', [-dhdt])),
    fault_dip_angle=90.0,
    fault_trace=(('x1', x), ('y1', 0), ('x2', x), ('y2', y)),
    include_boundaries=True,)

for t1 in range(int(n_faults)):
    #do tectonic displacement
    H[:]=z-BRz
    H[H<0] = 0
    shift_elevation(mg,z,0,x+mg.dx,mg.dx)
    nf.run_one_step(1)
    BRz[:] = z-H
```

```
def single_event(grid,k,t,w0):
    #grid setup
    model = copy.deepcopy(grid)
    z_model = model.at_node["topographic__elevation"]
    BRz_model = model.at_node["bedrock__elevation"]
    H_model = model.at_node["soil__depth"]

    time = t
    nt = 100 #number of time steps
    dt = time/nt #time step size for geomorphological simulation
    nt_f = int(n_faults) #number of tectonic events

    #setup geomorphologic process
    #weatherer setup
    weatherer = weatherer_mode(model,w0)
    #diffuser setup
    diffuser = diffuser_mode(model,k) #[0]LinearDiffuser,[1]TaylorNon
linearDiffuser,[2]DepthDependentDiffuser, [3]DepthDependentTaylorDiff
user
    eroder = diffuser[3] #pick diffuser mode that you want to use

        #do geomorphic displacement
    for t1 in range(nt):
```

```
        weatherer.run_one_step(dt)
        eroder.run_one_step(dt)
    fname = './output/ddtd/single/'+'z_'+str(int(k*1000))+'m2pkyr_'+s
tr(int(time/1000))+'kyr'+'.asc'
    write_esri_ascii(fname, model,clobber=True)
    return z_model
```

```
def evaluate(k, t):
    # Run the model with the given parameters
    y_pred = single_event(mg,k,t,w0=0.0002) #z model
    y_true = z_dem #z dem
    # Calculate the RMSE between the model output and the target outp
ut
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    return rmse

# Load existing grid search results
filename = "grid_search_results.csv"
results = {}
try:
    df = pd.read_csv(filename)
    for index, row in df.iterrows():
        results[(row['k'], row['t'])] = row['rmse']
except FileNotFoundError:
    pass

# Run the grid search
for i, k in tqdm(enumerate(k_range)):
    for j, t in tqdm(enumerate(t_range), leave=False):
        # Check if the current parameters have already been evaluated
        if (k, t) not in results:
            rmse = evaluate(k, t)
            results[(k, t)] = rmse
            print(k, t, rmse)

# Save the updated grid search results
df = pd.DataFrame([(k, t, rmse) for (k, t), rmse in results.items()],
columns=['k', 't', 'rmse'])
df.to_csv(filename, index=False)
```

```
13it [00:00, 489.44it/s]
```

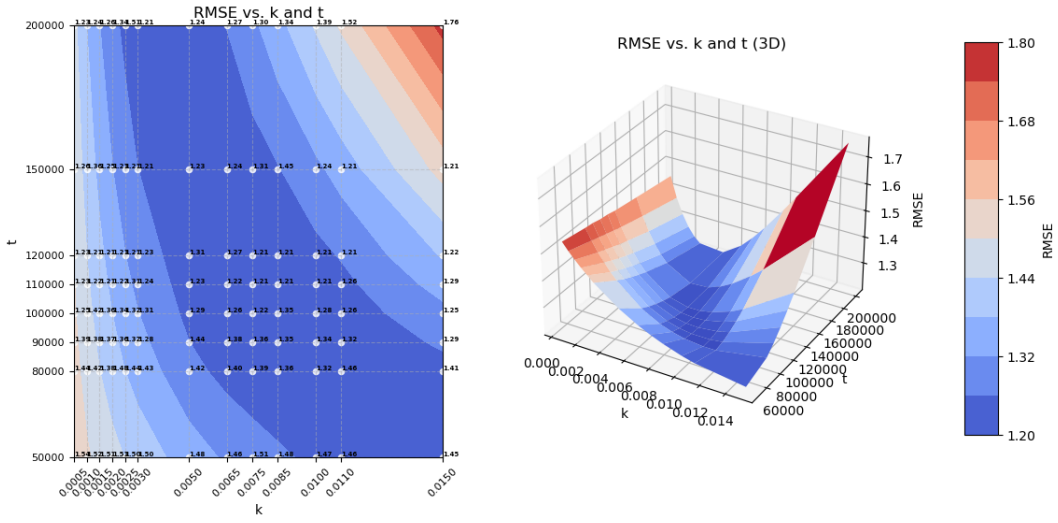
```
# Find the index of the minimum rmse value
min_rmse_index = df['rmse'].idxmin()

# Get the optimal k and t values corresponding to the minimum rmse va
lue
```

```
optimal_k = df.loc[min_rmse_index, 'k']
optimal_t = df.loc[min_rmse_index, 't']

print("Optimal k value:", optimal_k)
print("Optimal t value:", optimal_t)
```

```
Optimal k value: 0.0075
Optimal t value: 110000.0
```

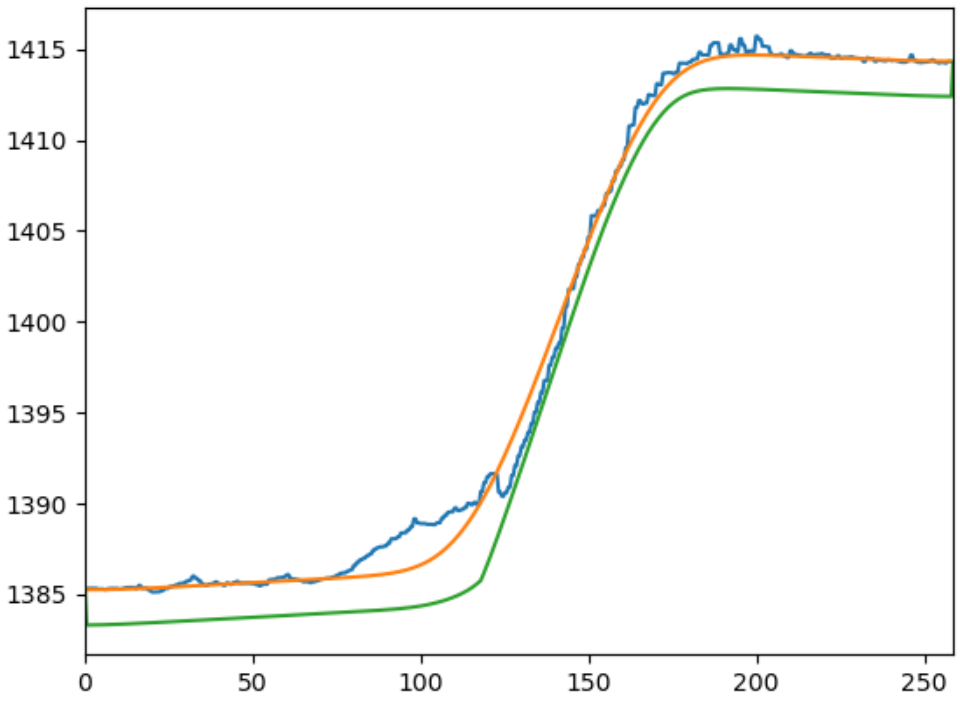
```
# Assuming your DataFrame is named 'df'
sorted_df = df.sort_values(['k', 't'])

# Save the sorted DataFrame to a CSV file
sorted_df.to_csv('sorted_grid_search_results.csv', index=False)

# Read the sorted data from the CSV file
sorted_df = pd.read_csv('sorted_grid_search_results.csv')

# Get the unique k and t values and the sorted results as a 2D array
unique_k = sorted_df['k'].unique()
unique_t = sorted_df['t'].unique()
sorted_results = sorted_df['rmse'].values.reshape(len(unique_k), len(
unique_t))

# Create the contour plot and 3D plot side by side
K, T = np.meshgrid(unique_k, unique_t)
fig = plt.figure(figsize=(12, 6))

# Contour plot
ax1 = fig.add_subplot(121)
c = ax1.contourf(K, T, sorted_results.T, levels=10, cmap='coolwarm' )

# Add scatter plot on top of the contour plot
ax1.scatter(K, T, c='white', s=20, marker='o', alpha=0.8)

# Display the value of each point on the contour plot
for k, t, sorted_result in zip(K.flatten(), T.flatten(), sorted_resul
ts.flatten()):
    ax1.text(k, t, f'{sorted_result:.2f}', fontsize=5, ha='left', va=
'bottom', color='black', fontweight='bold')

ax1.grid(True, linestyle='--', alpha=0.5)
ax1.set_xlabel('k')
ax1.set_ylabel('t')
ax1.set_title('RMSE vs. k and t')
ax1.set_xticks(k_range)
ax1.set_yticks(t_range)
ax1.tick_params(axis='both', which='major', labelsize=8)
ax1.xaxis.set_tick_params(rotation=45)
```

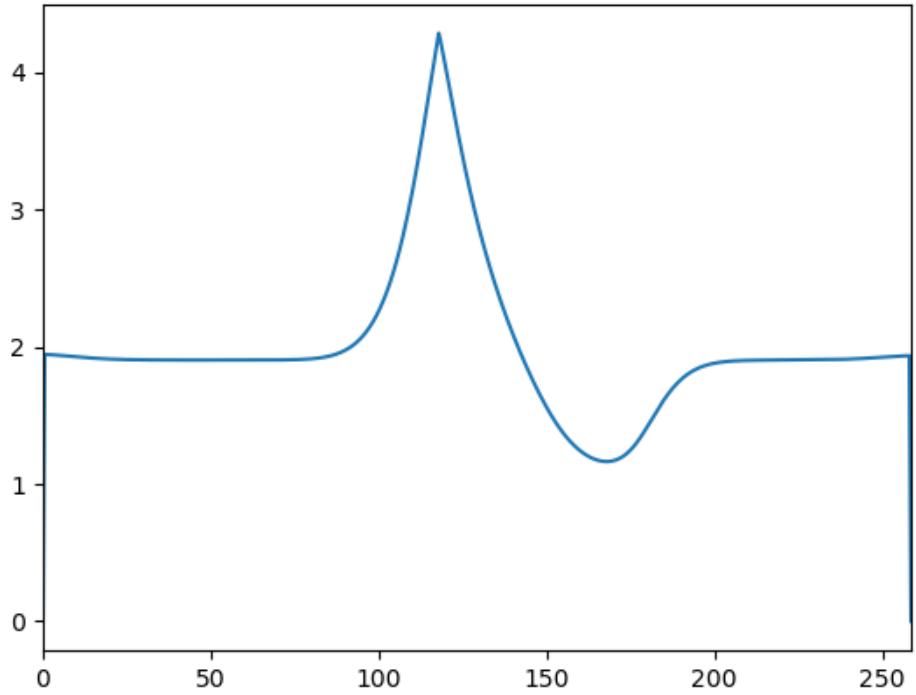
```
# 3D plot
ax2 = fig.add_subplot(122, projection='3d')
surf = ax2.plot_surface(K, T, sorted_results.T, cmap='coolwarm')

ax2.set_xlabel('k')
ax2.set_ylabel('t')
ax2.set_zlabel('RMSE',rotation=90)
ax2.set_title('RMSE vs. k and t (3D)')

fig.subplots_adjust(right=0.85)
# Create a colorbar axes and draw the colorbar
spacing = 0.05  # Adjust this value to change the spacing between the colorbar and the plots
cbar_ax = fig.add_axes([0.92, 0.15, 0.03, 0.7])  # Adjust the left, bottom, width, and height values
cb = Colorbar(ax=cbar_ax, mappable=c, orientation='vertical')
cb.set_label('RMSE')


# Save and show the plot
plt.savefig('./output/ddtd/single/rmse_vs_k_and_t.png')

plt.show()
```

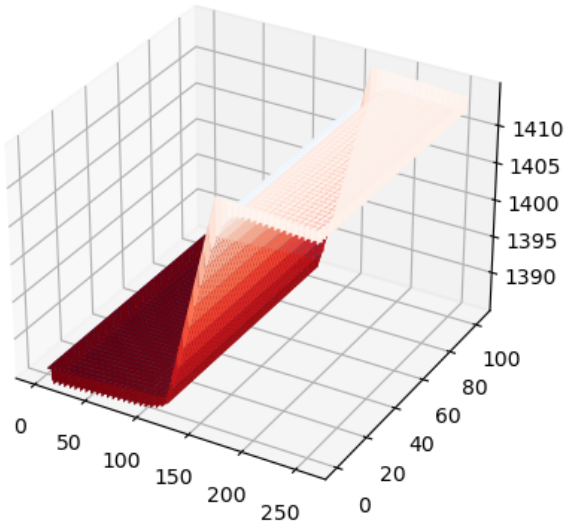



```
(grid,model110k) = read_esri_ascii('./output/ddtd/single/z_7m2pkyr_110kyr_topographic__elevation.asc')
(grid,model110k_br) = read_esri_ascii('./output/ddtd/single/z_7m2pkyr_110kyr_bedrock__elevation.asc')
(grid,model110k_h) = read_esri_ascii('./output/ddtd/single/z_7m2pkyr_
```

```
110kyr_soil__depth.asc')
grid.at_node["topographic__elevation"] = model110k
grid.at_node["bedrock__elevation"] = model110k_br
grid.at_node["soil__depth"] = model110k_h

mid_y = (grid.shape[0]/2)*grid.dy
x_nodes = grid.node_x[grid.node_y == mid_y]
y_dem = z_dem[grid.node_y == mid_y]
y_model110k = model110k[grid.node_y == mid_y]
y_model110k_br = model110k_br[grid.node_y == mid_y]
y_model110k_h = model110k_h[grid.node_y == mid_y]
plt.plot(x_nodes,y_dem)
plt.plot(x_nodes,y_model110k)
plt.plot(x_nodes,y_model110k_br)
plt.xlim(x_nodes[0],x_nodes[-1])
plt.show()
plt.plot(x_nodes,y_model110k-y_model110k_br)
plt.xlim(x_nodes[0],x_nodes[-1])
plt.show()
plot3d(grid,model110k,model110k_br,True,cmap='Blues_r',cmap2='Reds_r'
)
```




4
3
2
1
0
0
50
100
150
200
250


1410
1405
1400
1395
1390
100
80
60
40
20
0
0
50
100
150
200
250