Modeling Earth Surface Dynamics with Landlab

Nicole Gasparini, Tulane University

Huge thanks to Katy Barnhart, CU Boulder and Mark Piper, CSDMS

Material in this presentation comes from the Landlab team (in no particular order):

Tulane Team : Jordan Adams(*) & Nathan Lyons[#]

University of Colorado Boulder : Greg Tucker, Eric Hutton, Katy Barnhart[#], & Margaux Mouchene[#]

University of Washington : Erkan Istanbulluoglu, Sai Nudurupati*, and Christina Bandaragoda

Cardiff : Dan Hobley(#)

Goals for this class:

- You will have a better idea of what Landlab is and what it can do.
- You will know where to look to get help if you want to do more with Landlab.
- You will play with some Landlab coupled models.

Assumptions I make in every class I teach:

- If you have a question you will ask it.
- If you are uncomfortable you will tell me.
- If I ask you a question, you will try to answer, but if you cannot, it is fine to say “I pass”.
- You will treat me and your classmates with respect. We are all working towards the same goal of learning about modeling with Landlab. Leave bias outside the classroom *to the best of your ability*.

- Similarly,
  - I will do my best to answer all of your questions.
  - I will treat you with respect and without bias *to the best of my ability*.

Who am I?

- Geomorphologist, interested in how landscapes evolve over time, the details of surface processes, and how different drivers control surface processes (mostly in erosional environments).
- Undergraduate degrees in Math and Physical Geography. Graduate degrees in Environmental Engineering.
- Associate Professor.
- My day to day (research-wise) includes advising students, using spatial data of all sorts (DEMs, PRISM, imagery, etc.), creating and using numerical models, going in the field, writing and reviewing all sorts of things, and answering emails.

LANDLAB

Overview for clinic:

- Introduction to Landlab, http://landlab.github.io/#/
  - Other important websites coming.

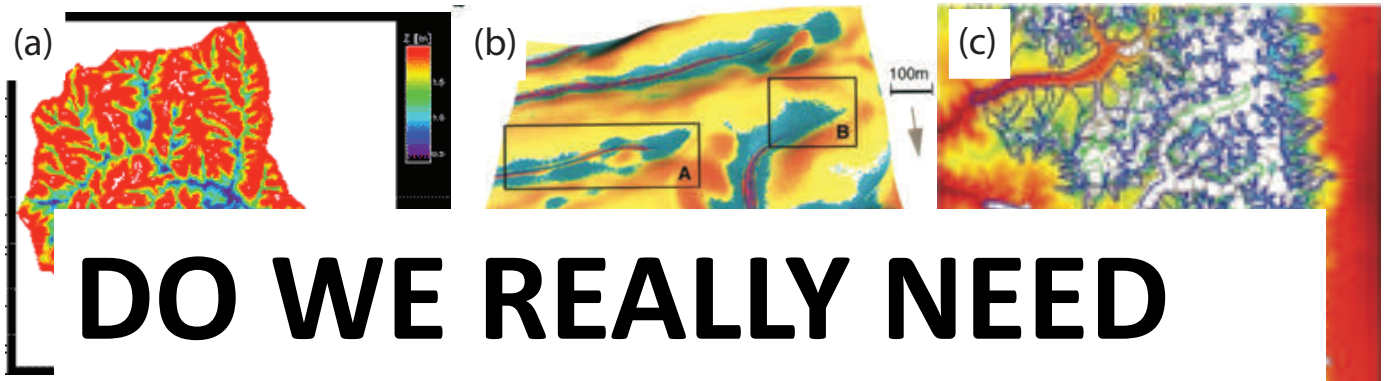
- Running Landlab on CSDMS JupyterHub

Lots of modeling approaches out there...

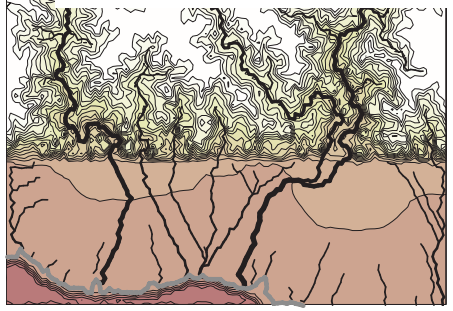
CATCHMENT HYDROLOGY
(Ivanov et al., 2004)

SOIL EROSION
(Mitas and Mitasova, 1998)

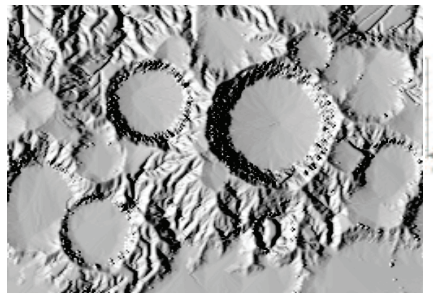
GLACIER DYNAMICS
(Kessler et al., 2006)



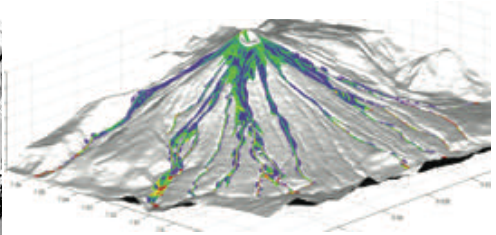
DO WE REALLY NEED ANOTHER MODEL???



LANDSCAPE EVOLUTION
(Tucker and Hancock, 2010)



IMPACT CRATERING AND DEGRADATION
(Howard, 2007)



LAVA FLOWS
(Kelfoun et al., 2009)

Challenges with many academic models:

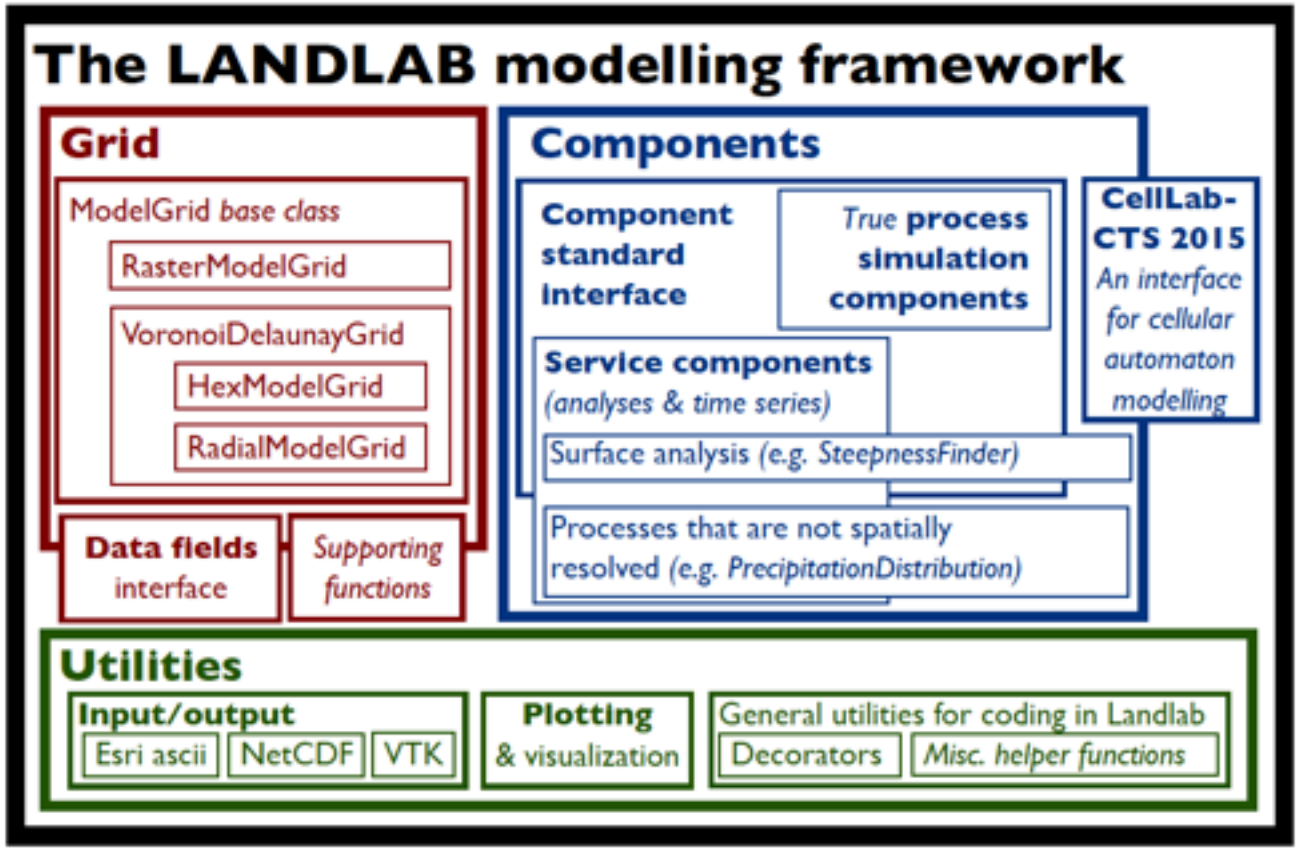
- built to answer a specific research question and not easily adaptable.
- poorly documented.
- not tested.
- platform dependent.
- not open source.

LANDLAB

What is Landlab?

- A toolkit for building models of processes that happen on the earth surface – e.g. overland flow, bedrock river incision, growth and death of vegetation.
- Not a landscape evolution model (LEM), but you can build LEMs with Landlab if you want to.
- Open source.
- **A Python library**.
- Anyone can contribute to the library (following some basic rules).
- Continuously tested.

What is Landlab?



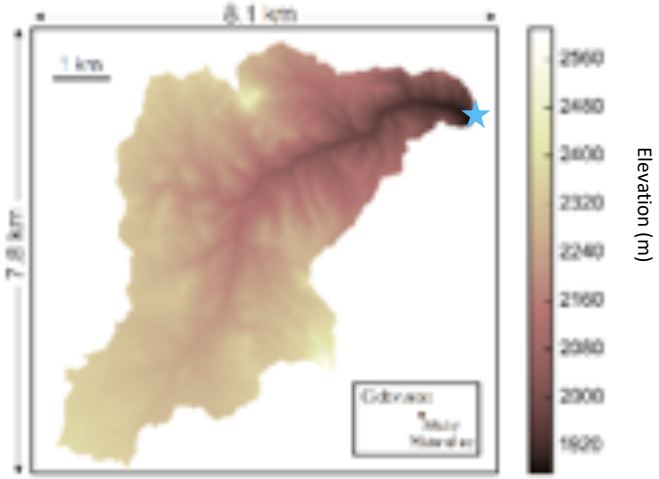
(Hobley et al., 2017, ESURF)

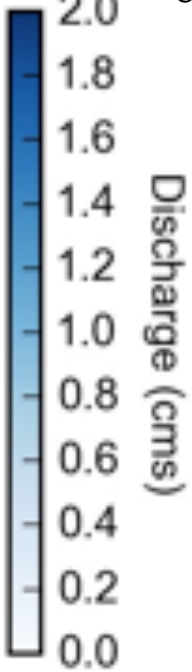
Landlab is also:

- Extensively documented.
- Has supporting materials such as:
  - Tutorials for learning Landlab.
  - Teaching materials for learning concepts in geomorphology and hydrology.
- An issues page where you can search for answers and post questions.
- A community of users, including **YOU**.

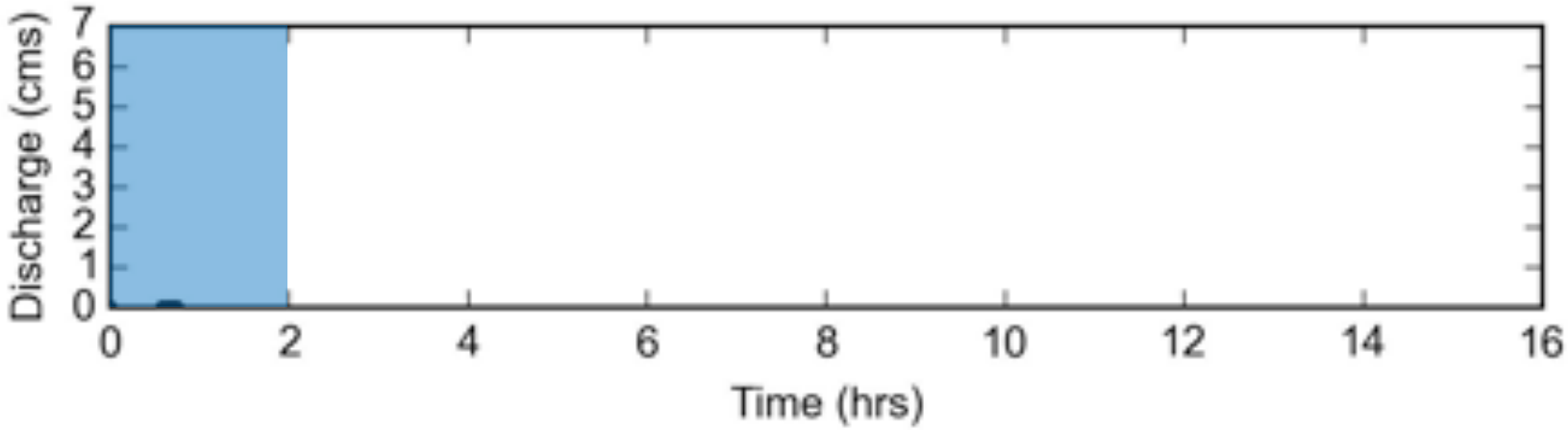
Application in a real world setting: Spring Creek, CO.

*Note scale differences





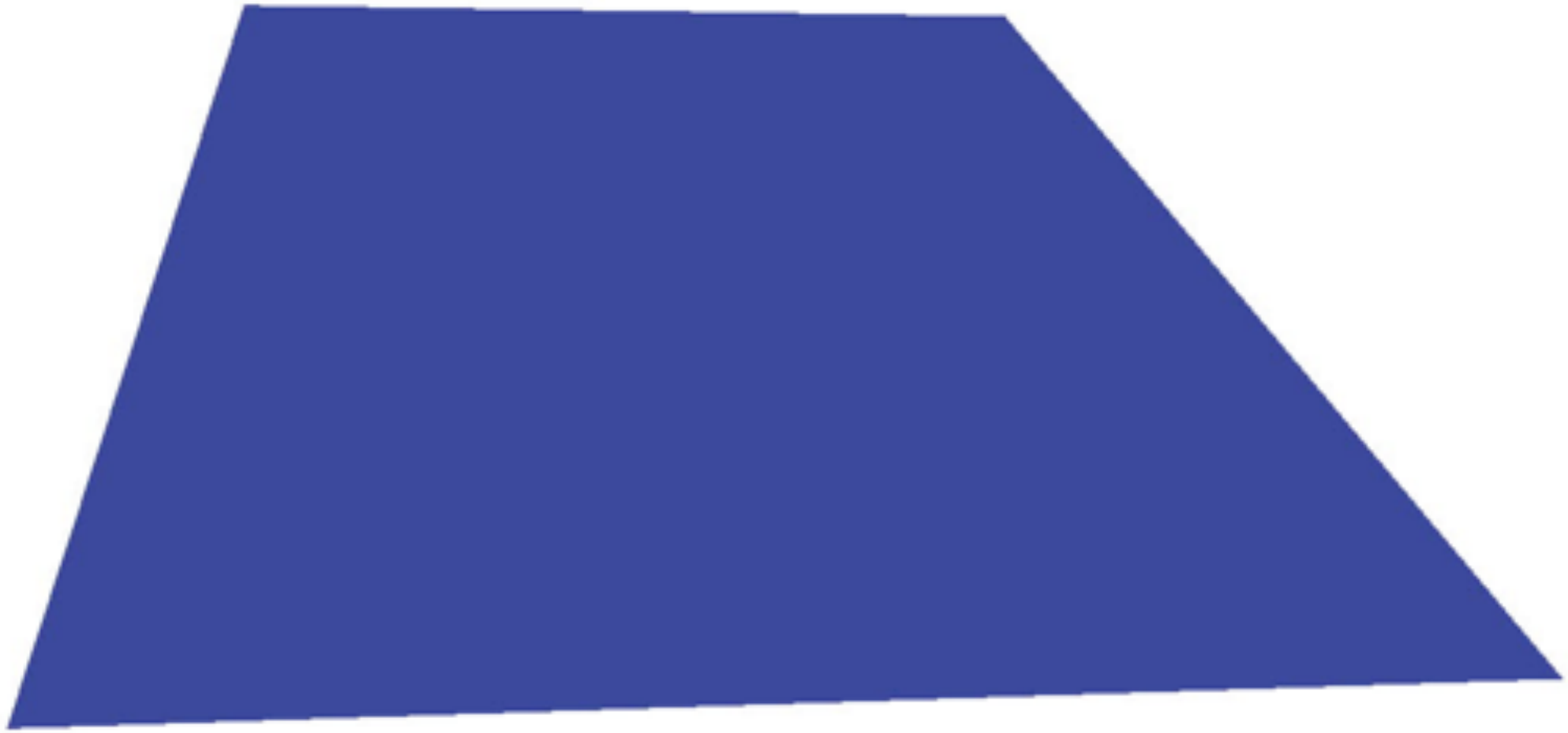
Storm duration



Source:
Jordan Adams,
See Adams et al., 2017

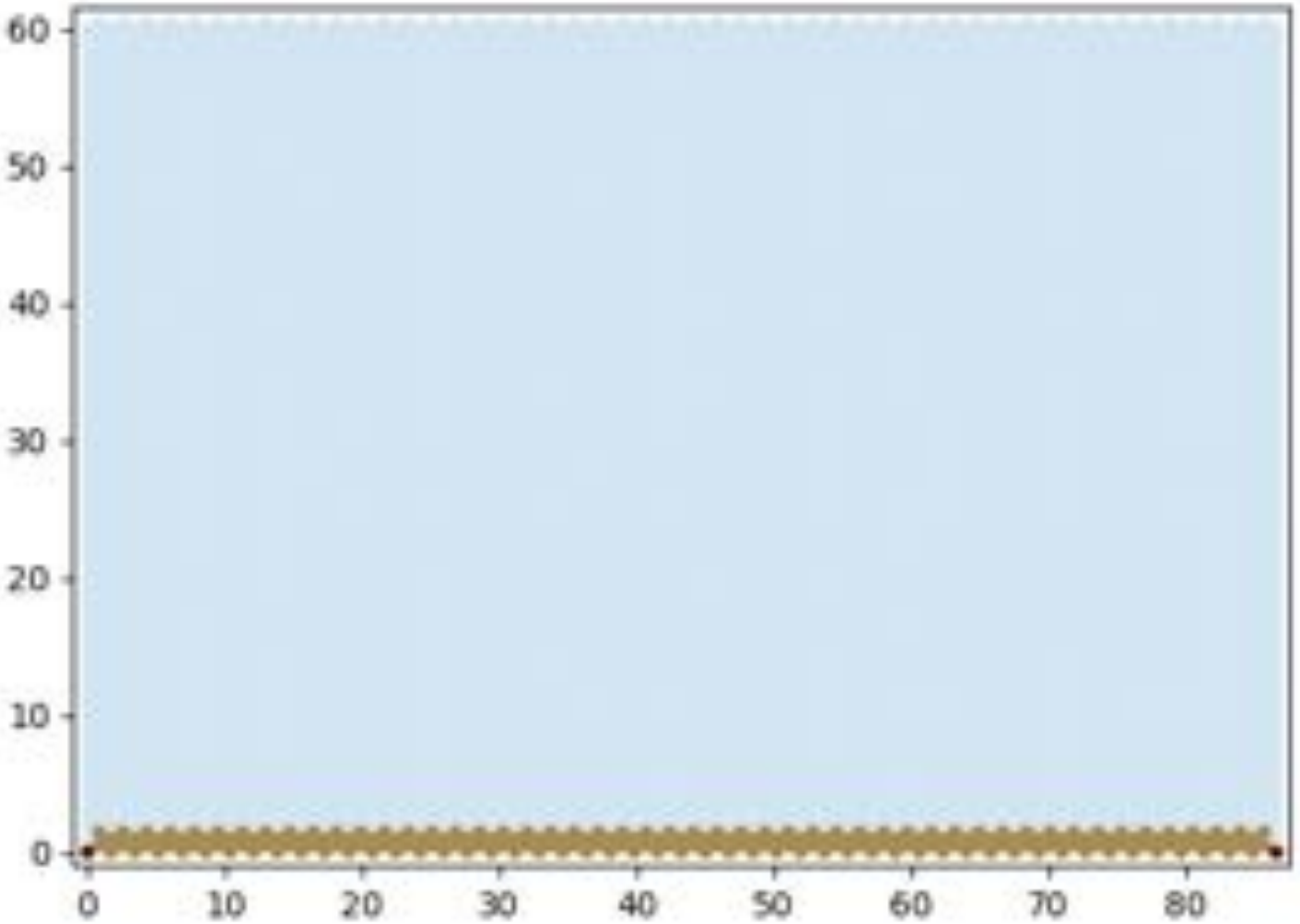
LANDLAB

Diffusion model written with 9 lines of code. From Greg Tucker.



LEM with channels and hillslopes. From Greg Tucker.

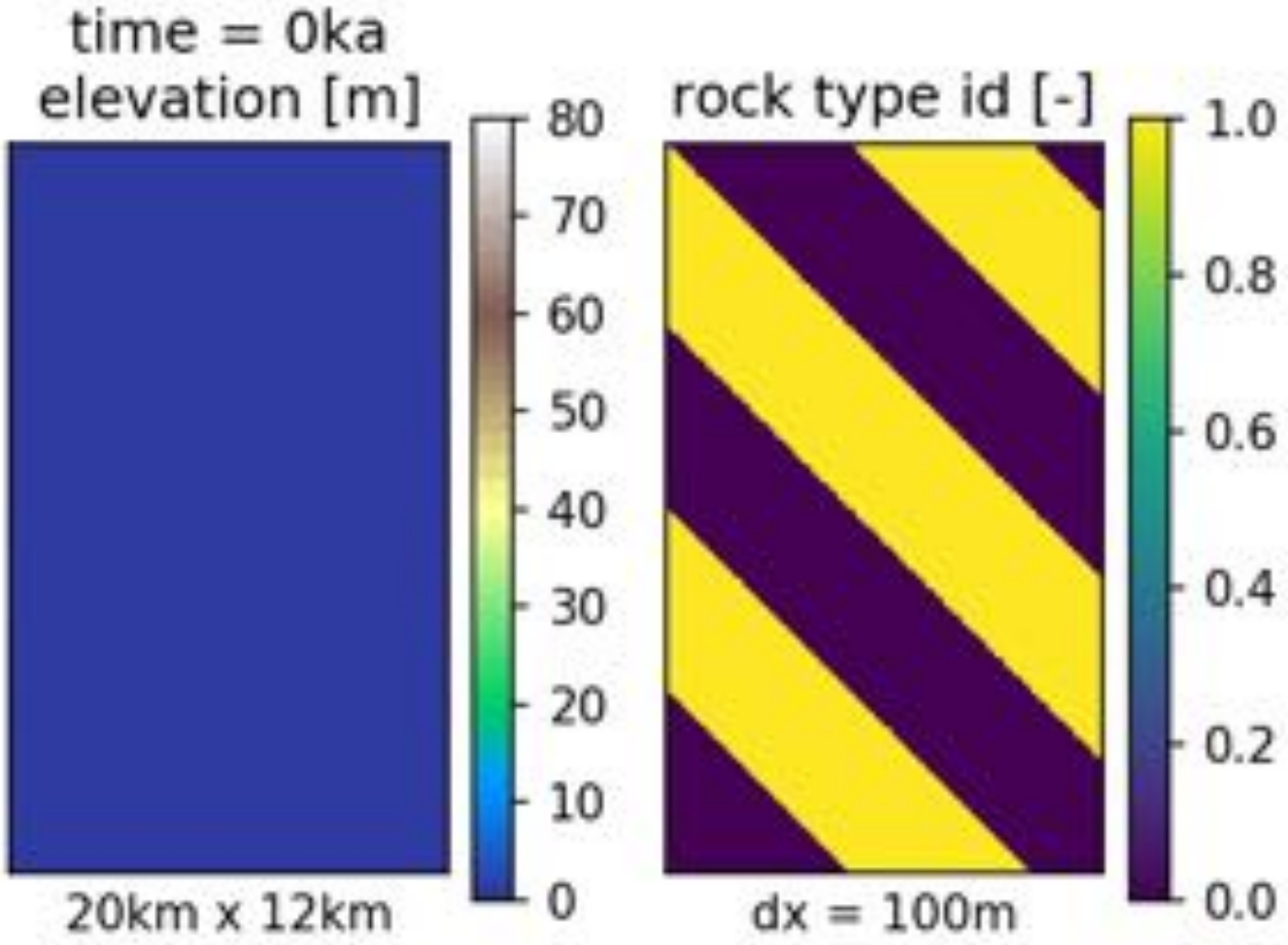
Hogback produced using CellLab, Tucker et al., 2018



Lithology and LithoLayers

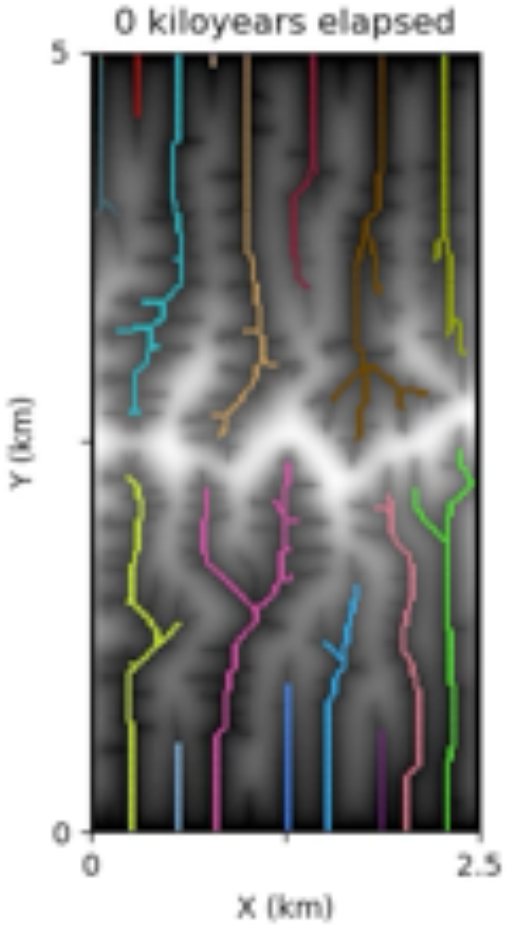
Two new Landlab components for spatially variable rock type.

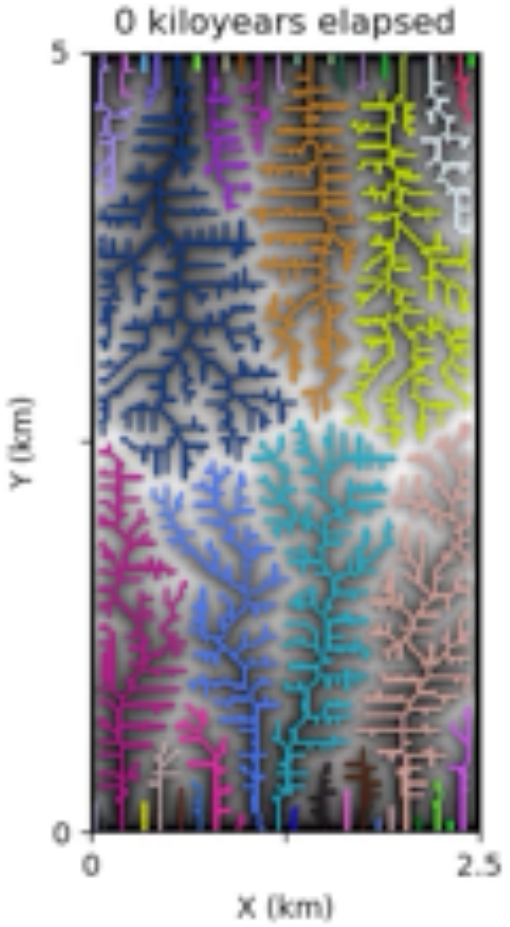
Barnhart et al., (2018). Lithology: A Landlab submodule for spatially variable rock properties. Journal of Open Source Software, 3(30), 979, https://doi.org/10.21105/joss.00979



Animation shows soft (purple) and hard (yellow) rocks dipping to the north east being uplifted and eroded by a simple landscape evolution model. Right panel shows rock type exposed at the surface.

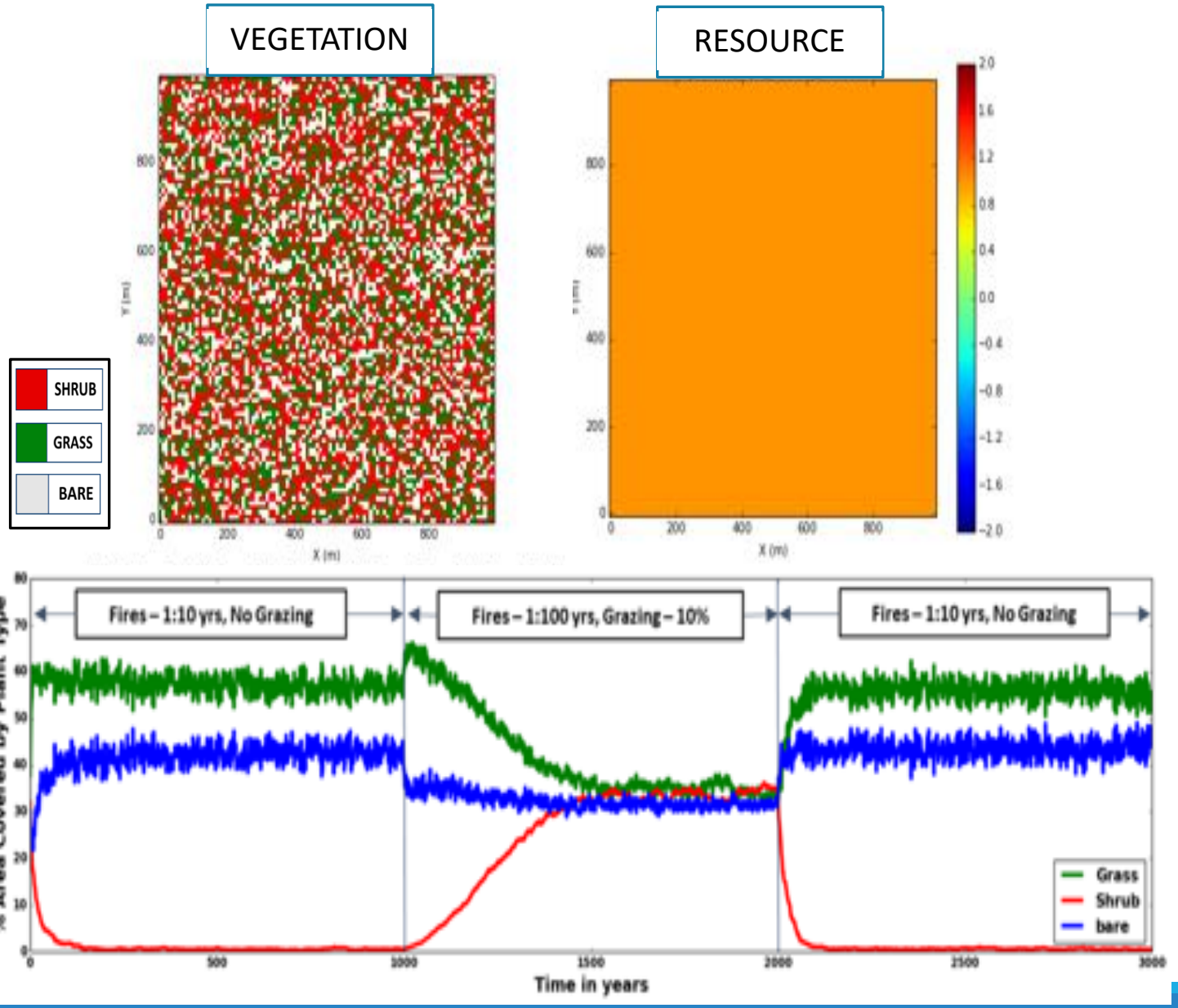
Drainage rearrangement, sensitivity to drainage density.
See Nathan Lyons et al., in review at ESURF.





LANDLAB

Climate Change Experiments #1



Source:
Sai Nudurupati,
U. Washington

LANDLAB

Earth Surf. Dynam., 5, 21–46, 2017
www.earth-surf-dynam.net/5/21/2017/
doi:10.5194/esurf-5-21-2017

Earth Surface Dynamics

Creative computing with Landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of Earth-surface dynamics

Daniel E. J. Hobley[1,2,3], Jordan M. Adams[4], Sai Siddhartha Nudurupati[5], Eric W. H. Hutton[6], Nicole M. Gasparini[4], Erkan Istanbulluoglu[5], and Gregory E. Tucker[1,2]

Earth Surf. Dynam., 6, 49–75, 2018
https://doi.org/10.5194/esurf-6-49-2018

Earth Surface Dynamics

Earth Surf. Dynam., 6, 563–582, 2018
https://doi.org/10.5194/esurf-6-563-2018

A hydroclimatological approach to predicting regional landslide probability using Landlab

Ronda Strauch[1], Erkan Istanbulluoglu[1], Sai Siddhartha Nudurupati[1], Christina Bandaragoda[1], Nicole M. Gasparini[2], and Gregory E. Tucker[3,4]

A lattice grain model of hillslope evolution

Gregory E. Tucker[1], Scott W. McCoy[2], and Daniel E. J. Hobley[3]

THE GEOLOGICAL SOCIETY OF AMERICA®

Off-fault deformation rate along the southern San Andreas fault at Mecca Hills, southern California, inferred from landscape modeling of curved drainages

Harrison J. Gray[1], Charles M. Shobe[1], Daniel E.J. Hobley[2], Gregory E. Tucker[1], Alison R. Duvall[3], Sarah A. Harbert[3], and Lewis A. Owen[4]

Geosci. Model Dev., 9, 823–839, 2016
www.geosci-model-dev.net/9/823/2016/
doi:10.5194/gmd-9-823-2016

Geoscientific Model Development

CellLab-CTS 2015: continuous-time stochastic cellular automaton modeling using Landlab

Gregory E. Tucker[1,2], Daniel E. J. Hobley[1,2], Eric Hutton[3], Nicole M. Gasparini[4], Erkan Istanbulluoglu[5], Jordan M. Adams[4], and Sai Siddartha Nudurupati[5]

Geosci. Model Dev., 10, 1645–1663, 2017
www.geosci-model-dev.net/10/1645/2017/
doi:10.5194/gmd-10-1645-2017

Geoscientific Model Development

The Landlab v1.0 OverlandFlow component: a Python tool for computing shallow-water flow across watersheds

Jordan M. Adams[1], Nicole M. Gasparini[1], Daniel E. J. Hobley[2], Gregory E. Tucker[3,4], Eric W. H. Hutton[5], Sai S. Nudurupati[6], and Erkan Istanbulluoglu[6]

Geosci. Model Dev., 10, 4577–4604, 2017
https://doi.org/10.5194/gmd-10-4577-2017

Geoscientific Model Development

The SPACE 1.0 model: a Landlab component for 2-D calculation of sediment transport, bedrock erosion, and landscape evolution

Charles M. Shobe, Gregory E. Tucker, and Katherine R. Barnhart

LANDLAB

Journal of Geophysical Research: Earth Surface

RESEARCH ARTICLE
10.1029/2017JF004509

Key Points:
- We model expansion of fluvial networks into low-relief landscapes with depressions, representing the postglacial U.S. Central Lowland
- When depressions are connected to

Modeled Postglacial Landscape Evolution at the Southern Margin of the Laurentide Ice Sheet: Hydrological Connection of Uplands Controls the Pace and Style of Fluvial Network Expansion

Jingtao Lai[1] and Alison M. Anders[1]

https://doi.org/10.5194/esurf-2018-13

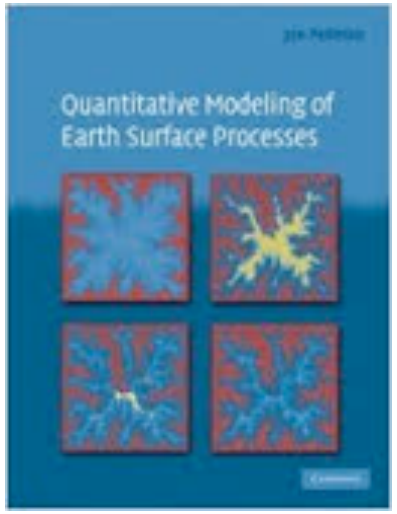
Research article

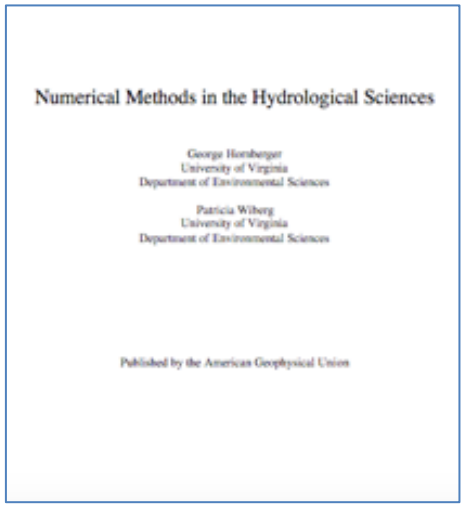
Effect of changing vegetation on denudation (part 2): Landscape response to transient climate and vegetation cover

Manuel Schmid[1], Todd A. Ehlers[1], Christian Werner[2], Thomas Hickler[2,3], and Juan-Pablo Fuentes-Espoz[4]

LANDLAB

Why build numerical models?
What are numerical models?



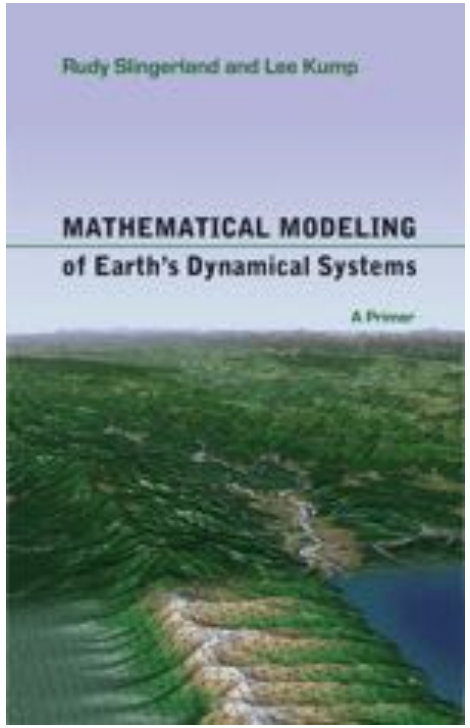


NATURE

DYNAMICAL MODEL

NUMERICAL ALGORITHM

SOFTWARE



LANDLAB

landlab.github.io

a python toolkit for modeling earth surface processes

Install | User Guide | Tutorials | Reference Manual | Support | Cite

What is Landlab?

Landlab is a Python-based modeling environment that allows scientists and students to build numerical landscape models. Designed for disciplines that quantify earth surface dynamics such as geomorphology, hydrology, glaciology, and stratigraphy, it can also be used in related fields.

https://landlab.readthedocs.io/en/latest/user_guide/index.html

User Guide

The User Guide describes Landlab by topic area.

Users brand-new to Landlab should start with 10 minutes to Landlab.

Further information on any specific method can be obtained in the API reference.

The Landlab project creates an environment in which scientists can build a numerical surface process model without having to code all of the individual components. Surface process models compute flows of mass, such as water, sediment, glacial ice, volcanic material, or landslide debris, across a gridded terrain surface. Surface process models have a number of commonalities, such as operating on a grid of points and routing material across the grid. Scientists who want to use a surface process model often build their own unique model from the ground up, re-coding the basic building blocks of their surface process model rather than taking advantage of codes that have already been written.

A list of papers and presentations using Landlab can be found here.

Introduction to Python

- Introduction to Python
  - Why Python?
  - Getting to know Python
  - NumPy, SciPy, and Efficient Coding Style
  - Cython

The Landlab Grid

- Introduction to Landlab's Gridding Library
  - How a Grid is Represented
  - Basic Grid Elements
  - Creating a Grid
  - Adding Data to a Landlab Grid Element using Fields
  - Representing Gradients in a Landlab Grid

https://landlab.readthedocs.io/en/latest/ (Reference Manual)

Landlab: A modular Earth Surface Dynamics modeling library

Landlab is an open-source Python-language package for numerical modeling of Earth surface dynamics. It contains:

- A gridding engine which represents the model domain. Regular and irregular grids are supported.
- A library of process components, each of which represents a physical process (e.g., generation of rain, erosion by flowing water). These components have a common interface and can be combined based on a user's needs.
- Utilities that support general numerical methods, file input/output, and visualization.

In addition Landlab contains a set of Jupyter notebook tutorials providing an introduction to core concepts and examples of use.

Landlab was desiged for disciplines that quantify Earth surface dynamics such as geomorphology, hydrology, glaciology, and stratigraphy. It can also be used in related fields. Scientists who use this type of model often build their own unique model from the ground up, re-coding the basic building blocks of their landscape model rather than taking advantage of codes that have already been written. Landlab saves practitioners from the need for this kind of re-invention by providing standardized components that they can re-use.

Watch the webinar Landlab Toolkit Overview at CSDMS to learn more.

The most current source code is always available from our git repository.

If you are interested in the state of current development, we compile ongoing development. If you are interested in contributing but don't know how to get started (THANK YOU!), we compile desired contributions and have a page all about development.

Many Jupyter notebooks exist describing use of Landlab. Find an overview here. A subset of these notebooks are designed specifically for the classroom. Information about them and how to set them up for classroom use is described on this page.

Landlab 2.0

In late December 2019 Landlab switched to version 2.0-beta. Landlab will be in 2.0-beta until the Landlab 2.0 publication is finalized.

Supported Python Versions

Python 3.6, 3.7, and 3.8

INDEX

Search

Table of Contents

This Page

Documentation Outline

- Installation Instructions
  - Conda instructions
  - Pip Instructions
  - Conda Environment
  - Additional Resources
- User Guide
  - Introduction to Python
  - The Landlab Grid
  - Model with Landlab and Components
  - Landlab Tutorial Library
  - Additional resources
  - Presentations, Clinics, and Classroom Use
  - Overland flow User Guide
  - CellLab-CTS User Guide
  - Major Version Transition Guides
- API reference
  - Basic Model Interface
  - Cellular Automata (CA)
  - Command Interface
  - Components
  - Core Landlab Classes
  - Data Record
  - Landlab Fields
  - Landlab Framework
  - Landlab Graph
  - Landlab Grids
  - Input/Output (IO)
  - Layers
  - Plotting and Visualization
  - Testing Tools
  - Utilities and Decorators
  - Values
  - Full Index

Components

This section contains documentation and API reference information for the following categories of components:

Hillslope geomorphology

- LinearDiffuser: Model soil creep using "linear diffusion" transport law (no depth dependence)
- PerronNLDiffuse: Model soil creep using implicit solution to nonlinear diffusion law
- DepthDependentDiffuser: depth dependent diffusion after Johnstone and Hilley (2014)
- TransportLengthHillslopeDiffuser: Hillslope diffusion component in the style of Carretier et al. (2016), and Davy and Lague (2009)
- TaylorNonLinearDiffuser: Model non-linear soil creep after Ganti et al. (2013)
- DepthDependentTaylorDiffuser: Model depth dependent non-linear soil creep after Ganti et al. (2012) and Johnstone and Hilley (2014)

Fluvial geomorphology

- FastscapeEroder: Compute fluvial erosion using stream power theory ("fastscape" algorithm)
- StreamPower: Compute fluvial erosion using stream power theory (also uses "fastscape" algorithm but provides slightly different formulation and options)
- SedDepEroder: Compute fluvial erosion using using "tools and cover" theory
- StreamPowerSmoothThresholdEroder: Compute fluvial erosion using stream power theory with a numerically smoothed threshold
- DetachmentLtdErosion: Solve stream power equations, but without stability checks
- ErosionDeposition: Fluvial erosion in the style of Davy and Lague (2009)
- Stream Power with Alluvium Conservation and Entrainment

Flow routing

- The Landlab FlowDirectors: Components for Flow Direction
  - FlowDirectorSteepest
  - FlowDirectorD8
  - FlowDirectorMFD
  - FlowDirectorDinf

LinearDiffuser: Model soil creep using “linear diffusion” transport law (no depth dependence)

class LinearDiffuser(*grid*, *linear_diffusivity=0.01*, *method='simple'*, *deposit=True*)[source]

Bases: landlab.core.model_component.Component

This component implements linear diffusion of a Landlab field.

Component assumes grid does not deform. If the boundary conditions on the grid change after component instantiation, be sure to also call updated_boundary_conditions to ensure these are reflected in the component (especially if fixed_links are present).

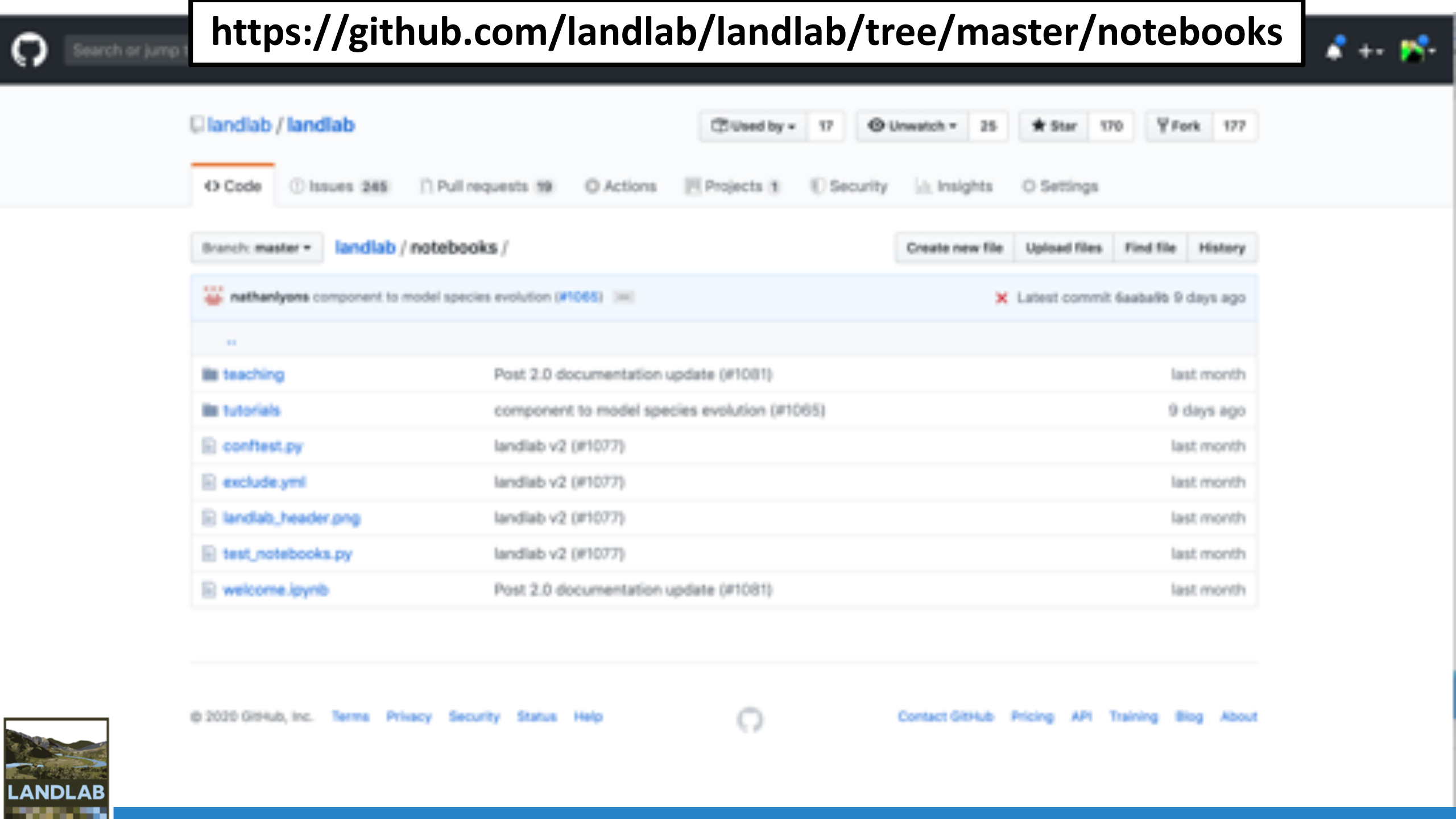
The method keyword allows control of the way the solver works. Options other than 'simple' will make the component run slower, but give second order solutions that incorporate information from more distal nodes (i.e., the diagonals). Note that the option 'resolve_on_patches' can result in somewhat counterintuitive behaviour - this option tells the component to treat the diffusivity as a field **with directionality to it** (i.e., that the diffusivites are defined on links). Thus if all links have the same diffusivity value, with this flag active "effective" diffusivities at the nodes will be *higher* than this value (by a factor of root 2) as the diffusivity at each patch will be the mean vector sum of that at the bounding links.

The primary method of this class is run_one_step.

Examples

```
>>> from landlab import RasterModelGrid
>>> import numpy as np
>>> mg = RasterModelGrid((9, 9))
>>> z = mg.add_zeros("topographic__elevation", at="node")
>>> z.reshape((9, 9))[4, 4] = 1.
>>> mg.set_closed_boundaries_at_grid_edges(True, True, True, True)
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.)
>>> for i in range(1):
...     ld.run_one_step(1.)
```

https://github.com/landlab/landlab/tree/master/notebooks



LANDLAB

The first example uses set_watershed_boundary_condition which finds the outlet for the user.

- First import what we need.

In [ ]:
```
from landlab import RasterModelGrid
import numpy as np
from landlab.plot.imshow import imshow_grid
%matplotlib inline
```

- Now we create a 5 by 5 grid with a spacing (dx and dy) of 1.
- We also create an elevation field with value of 1. everywhere, except at the outlet, where the elevation is 0. In this case the outlet is in the middle of the bottom row, or at at location (0,2) and has a node id of 2.

In [ ]:
```
mg1 = RasterModelGrid((5,5), 1.)
z1 = mg1.add_ones('node','topographic__elevation')
mg1.at_node['topographic__elevation'][2] = 0.
mg1.at_node['topographic__elevation']
```

- The set_watershed_boundary_condition in RasterModelGrid will find the outlet of the watershed.
- This method takes the node data, in this case z, and, optionally the no_data value.
- This method sets all nodes that have no_data values to closed boundaries.
- This example does not have any no_data values, which is fine.
- In this case, the code will set all of the perimeter nodes as CLOSED_BOUNDARY (boundary status 4) in order to create this boundary around the core nodes.
- The exception on the perimeter is node 2 (with elevation of 0). Although it is on the perimeter, it has a value and it has the lowest value. So in this case node 2 will be set as FIXED_VALUE_BOUNDARY (boundary status 1).
- The rest of the nodes are set as a CORE_NODE (boundary status 0)

In [ ]:
```
mg1.set_watershed_boundary_condition(z1)
```

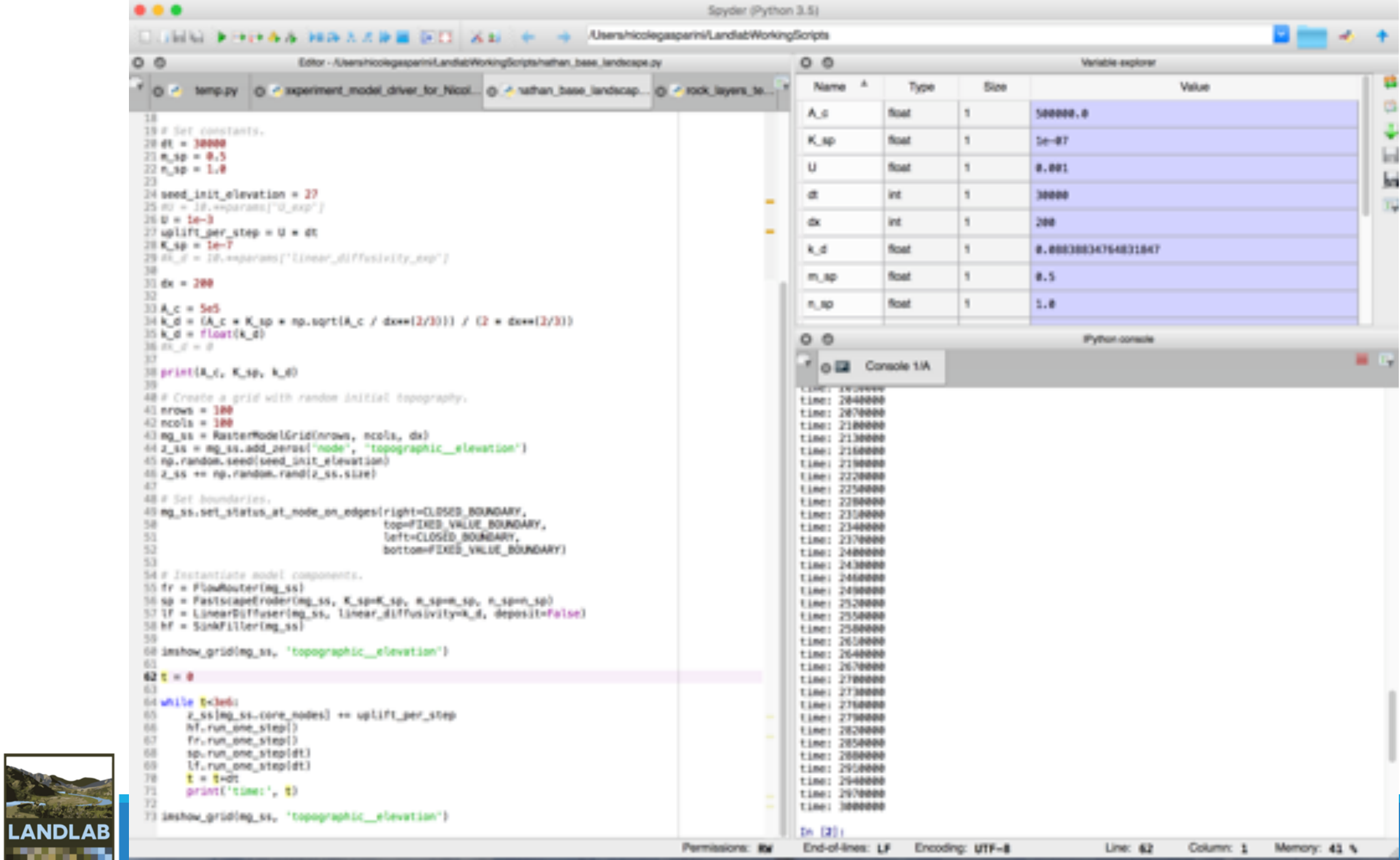
- Check to see that node status were set correctly.
- imshow_grid will default to not plot the value of CLOSED_BOUNDARY nodes, which is why we override this below with the option color_for_closed

In [ ]:
```
imshow_grid(mg1, mg1.status_at_node, color_for_closed='blue')
```

The second example uses set_watershed_boundary_condition_outlet_coords

- In this case the user knows the coordinates of the outlet node.
- First instantiate a new grid, with new data values.

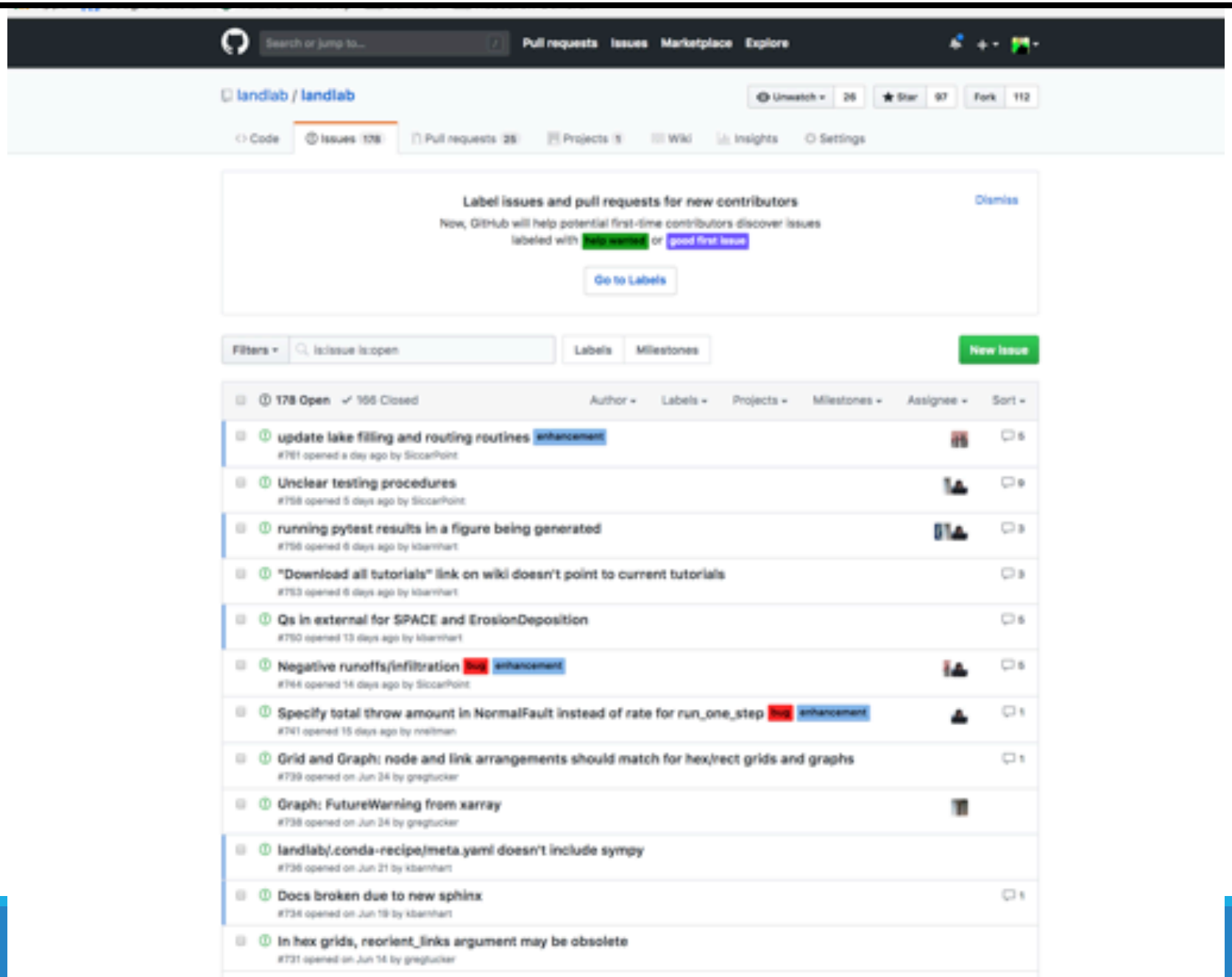
In [ ]:
```
mg2 = RasterModelGrid((5,5), 10.)
z2 = mg2.add_ones('node','topographic__elevation')
mg2.at_node['topographic__elevation'][1] = 0.
mg2.at_node['topographic__elevation']
```


Spyder (Python 3.5)
/Users/nicolegasparini/Landlab/WorkingScripts
Editor - /Users/nicolegasparini/Landlab/WorkingScripts/nathan_base_landscape.py
temp.py
experiment_model_driver_for_Nicol...
nathan_base_landscap...
rock_layers_te...
# Set constants.
dt = 30000
m_sp = 0.5
n_sp = 1.0
seed_init_elevation = 27
#U = 10.**eparams['U_exp']
U = 1e-3
uplift_per_step = U * dt
K_sp = 1e-7
#K_d = 10.**eparams['linear_diffusivity_exp']
dx = 200
A_c = 5e5
k_d = (A_c * K_sp * np.sqrt(A_c / dx**(2/3))) / (2 * dx**(2/3))
k_d = float(k_d)
#K_d = 0
print(A_c, K_sp, k_d)
# Create a grid with random initial topography.
nrows = 100
ncols = 100
mg_ss = RasterModelGrid(nrows, ncols, dx)
z_ss = mg_ss.add_zeros('node', 'topographic__elevation')
np.random.seed(seed_init_elevation)
z_ss += np.random.rand(z_ss.size)
Set boundaries.
mg_ss.set_status_at_node_on_edges(right=CLOSED_BOUNDARY,
top=FIXED_VALUE_BOUNDARY,
left=CLOSED_BOUNDARY,
bottom=FIXED_VALUE_BOUNDARY)
# Instantiate model components.
fr = FlowRouter(mg_ss)
sp = FastscapeEroder(mg_ss, K_sp=K_sp, m_sp=m_sp, n_sp=n_sp)
lf = LinearDiffuser(mg_ss, linear_diffusivity=k_d, deposit=False)
hf = SinkFiller(mg_ss)
imshow_grid(mg_ss, 'topographic__elevation')
t = 0
while t<1e6:
z_ss[mg_ss.core_nodes] += uplift_per_step
hf.run_one_step()
fr.run_one_step()
sp.run_one_step(dt)
lf.run_one_step(dt)
t = t+dt
print('time:', t)
imshow_grid(mg_ss, 'topographic__elevation')
Variable explorer
Name
Type
Size
Value
A_c
float
1
500000.0
K_sp
float
1
1e-07
U
float
1
0.001
dt
int
1
30000
dx
int
1
200
k_d
float
1
0.0003883476483831847
m_sp
float
1
0.5
n_sp
float
1
1.0
IPython console
Console 1/A
time: 2040000
time: 2070000
time: 2100000
time: 2130000
time: 2160000
time: 2190000
time: 2220000
time: 2250000
time: 2280000
time: 2310000
time: 2340000
time: 2370000
time: 2400000
time: 2430000
time: 2460000
time: 2490000
time: 2520000
time: 2550000
time: 2580000
time: 2610000
time: 2640000
time: 2670000
time: 2700000
time: 2730000
time: 2760000
time: 2790000
time: 2820000
time: 2850000
time: 2880000
time: 2910000
time: 2940000
time: 2970000
time: 3000000
In [2]:
Permissions: RW
End-of-lines: LF
Encoding: UTF-8
Line: 62
Column: 1
Memory: 43 %

LANDLAB

https://github.com/landlab

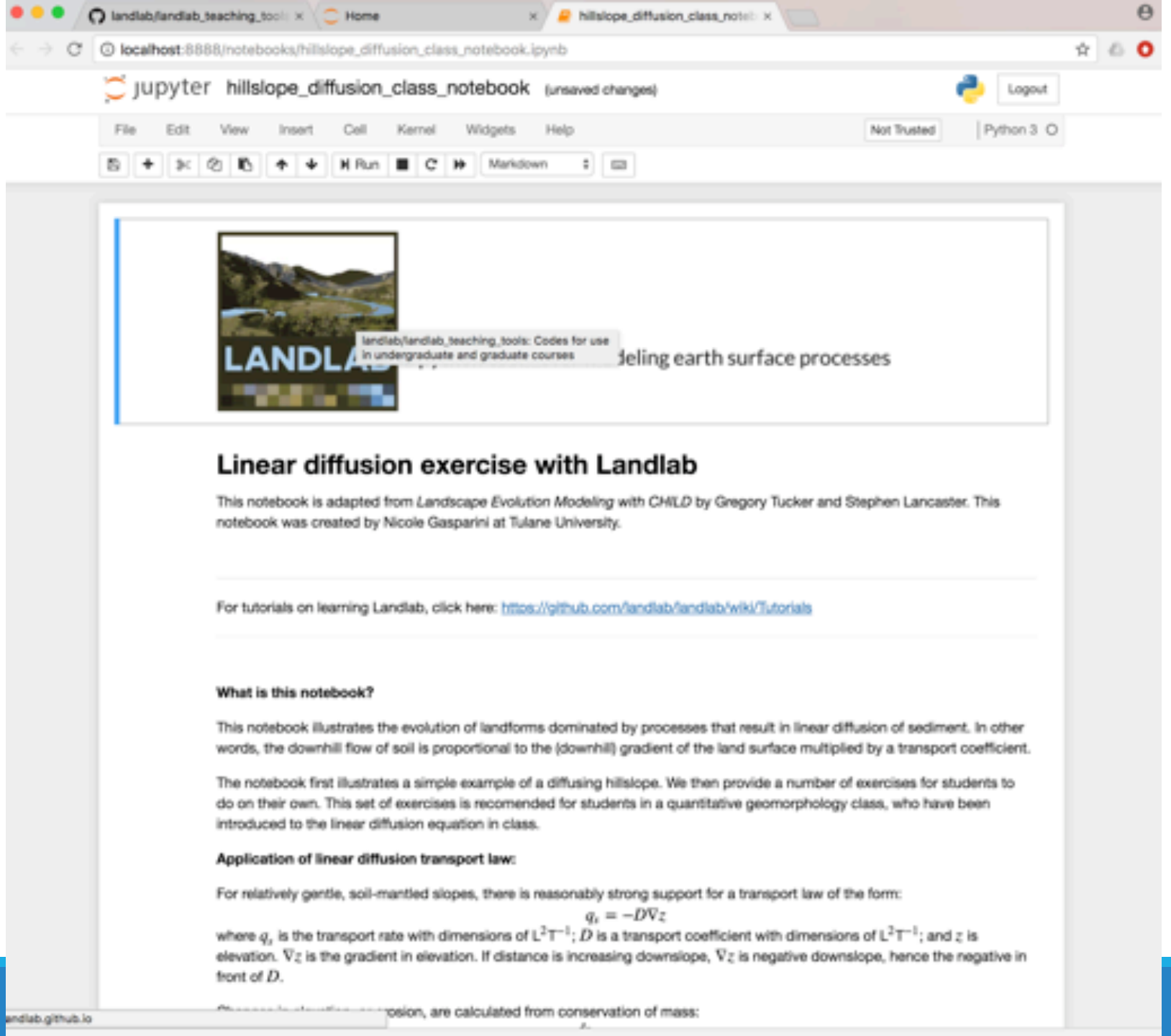
https://github.com/landlab/landlab/issues



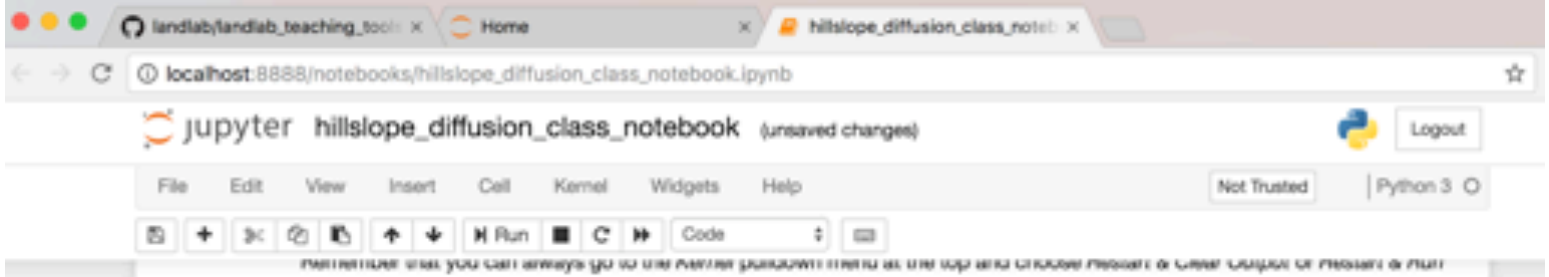
LANDLAB

Your average undergrad knows nothing about Landlab and numerical modeling. Maybe even nothing about coding.

- Step in their shoes!
- They can still learn from models.


landlab/landlab_teaching_tools
Home
hillslope_diffusion_class_notebook
localhost:8888/notebooks/hillslope_diffusion_class_notebook.ipynb
jupyter hillslope_diffusion_class_notebook (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
LANDLAB
landlab/landlab_teaching_tools: Codes for use in undergraduate and graduate courses
deling earth surface processes
Linear diffusion exercise with Landlab
This notebook is adapted from Landscape Evolution Modeling with CHILD by Gregory Tucker and Stephen Lancaster. This notebook was created by Nicole Gasparini at Tulane University.
For tutorials on learning Landlab, click here: https://github.com/landlab/landlab/wiki/Tutorials
What is this notebook?
This notebook illustrates the evolution of landforms dominated by processes that result in linear diffusion of sediment. In other words, the downhill flow of soil is proportional to the (downhill) gradient of the land surface multiplied by a transport coefficient.
The notebook first illustrates a simple example of a diffusing hillslope. We then provide a number of exercises for students to do on their own. This set of exercises is recomended for students in a quantitative geomorphology class, who have been introduced to the linear diffusion equation in class.
Application of linear diffusion transport law:
For relatively gentle, soil-mantled slopes, there is reasonably strong support for a transport law of the form:
$q_s = -D\nabla z$
where q_s is the transport rate with dimensions of L^2T^{-1}; D is a transport coefficient with dimensions of L^2T^{-1}; and z is elevation. ∇z is the gradient in elevation. If distance is increasing downslope, ∇z is negative downslope, hence the negative in front of D.
osion, are calculated from conservation of mass:
landlab.github.io


LANDLAB



All if you change things and want to start afresh. If you just change one code block and rerun only that code block, only the parts of the code in that code block will be updated. (E.g. if you change parameters but don't reset the code blocks that initialize run time or topography, then these values will not be reset.)

Now on to the code example

Import statements. You should not need to edit this.

In [1]:
```
# Code Block 1

import numpy as np
from landlab.plot.imshow import imshow_grid
#below is to make plots show up in the notebook
%matplotlib inline
from matplotlib.pyplot import figure, show, plot, \
                              xlabel, ylabel, title, legend, ylim
```

We will create a grid with 41 rows and 5 columns, and dx is 5 m (a long, narrow, hillslope). The initial elevation is 0 at all nodes.

We set-up boundary conditions so that material can leave the hillslope at the two short ends.

In [ ]:
```
# Code Block 2

# setup grid
from landlab import RasterModelGrid
mg = RasterModelGrid((41, 5), 5.)
z_vals = mg.add_zeros('topographic__elevation', at='node')

# initialize some values for plotting
ycoord_rast = mg.node_vector_to_raster(mg.node_y)
ys_grid = ycoord_rast[:, 2]

# set boundary condition.
mg.set_closed_boundaries_at_grid_edges(True, False, True, False)
```

Now we import and initialize the *LinearDiffuser* component.

In [ ]:
```
# Code Block 3

from landlab.components import LinearDiffuser
D = 0.01 # initial value of 0.01 m^2/yr
lin_diffuse = LinearDiffuser(mg, linear_diffusivity=D)
```

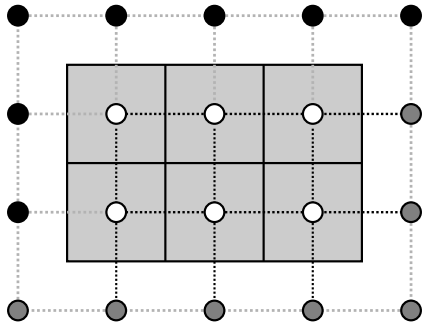
LANDLAB

Take ~10 minutes to play with this notebook.

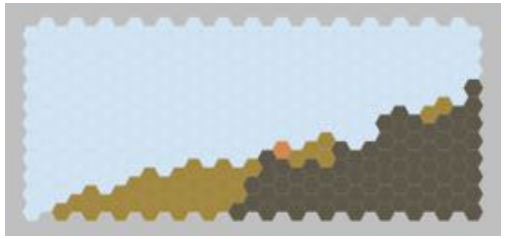
What Landlab provides

1. **Grid creation and management**
   - **Create** a structured or unstructured **grid** in one or a few lines of code
   - **Attach data** to grid elements
     - Facilitates staggered-grid schemes
     - Passing the grid = passing the data

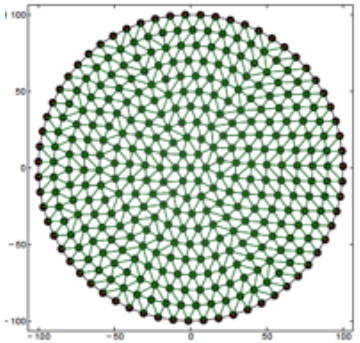
RASTER



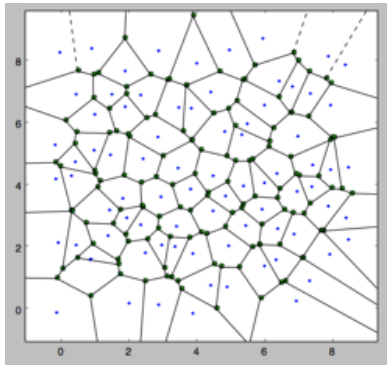
HEXAGONAL



RADIAL



VORONOI / DELAUNAY



Using Landlab grid

Aim: make it easier to set up a 2D numerical model grid

Grid data and functions contained in a single Python object

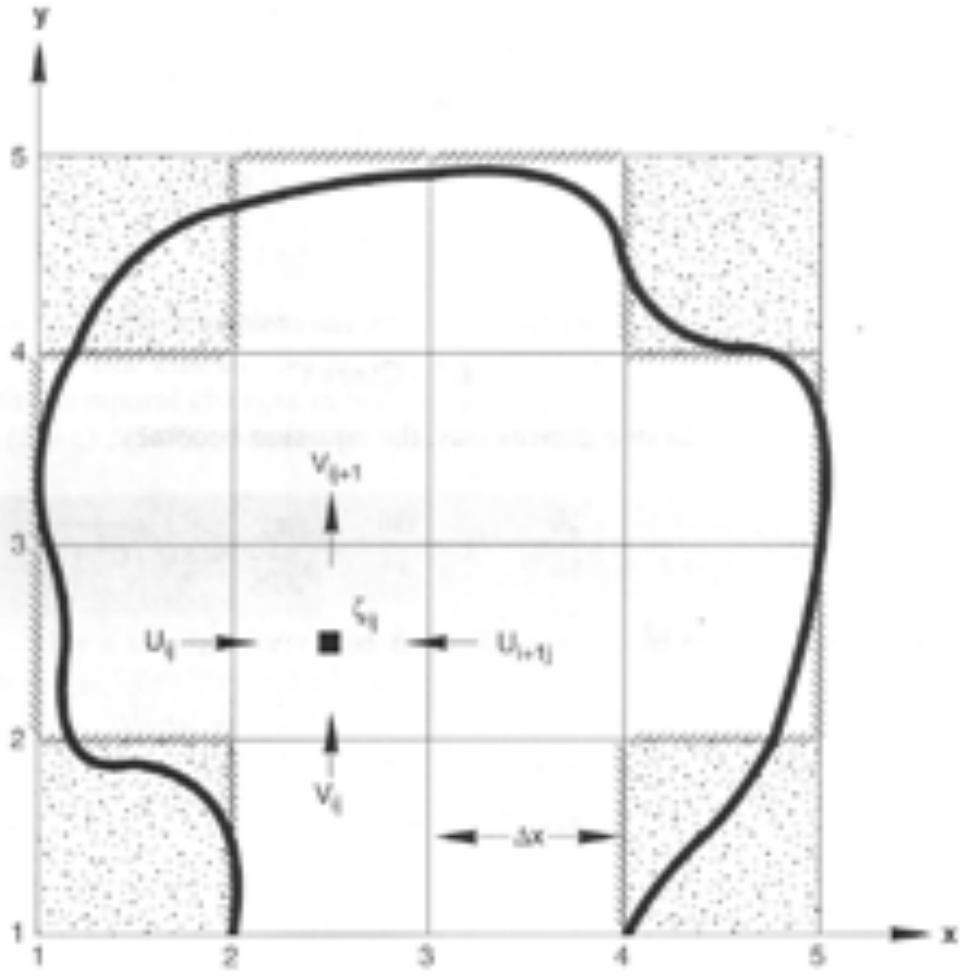
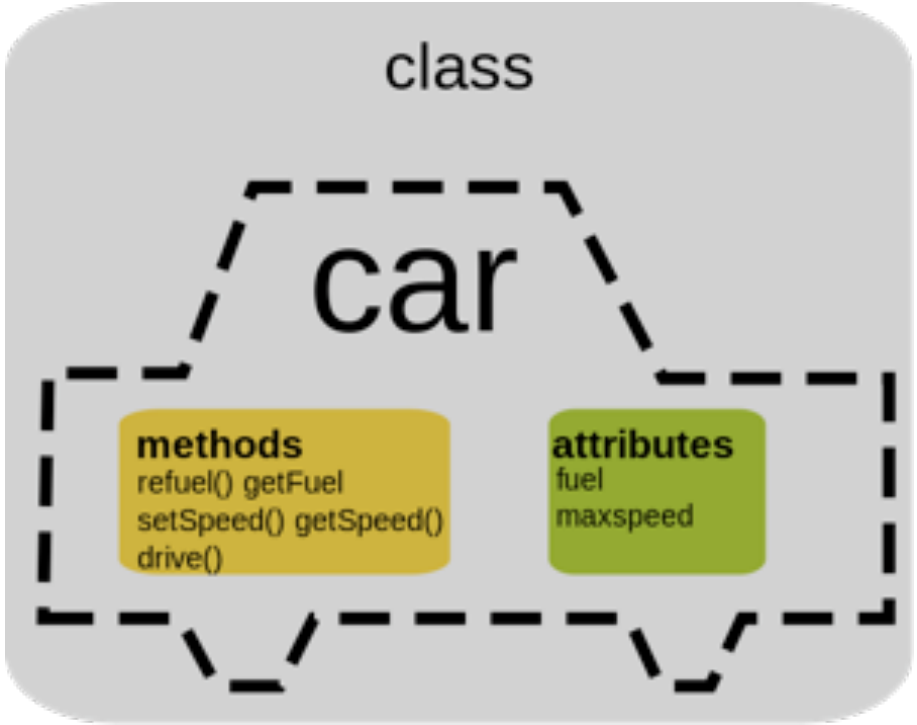


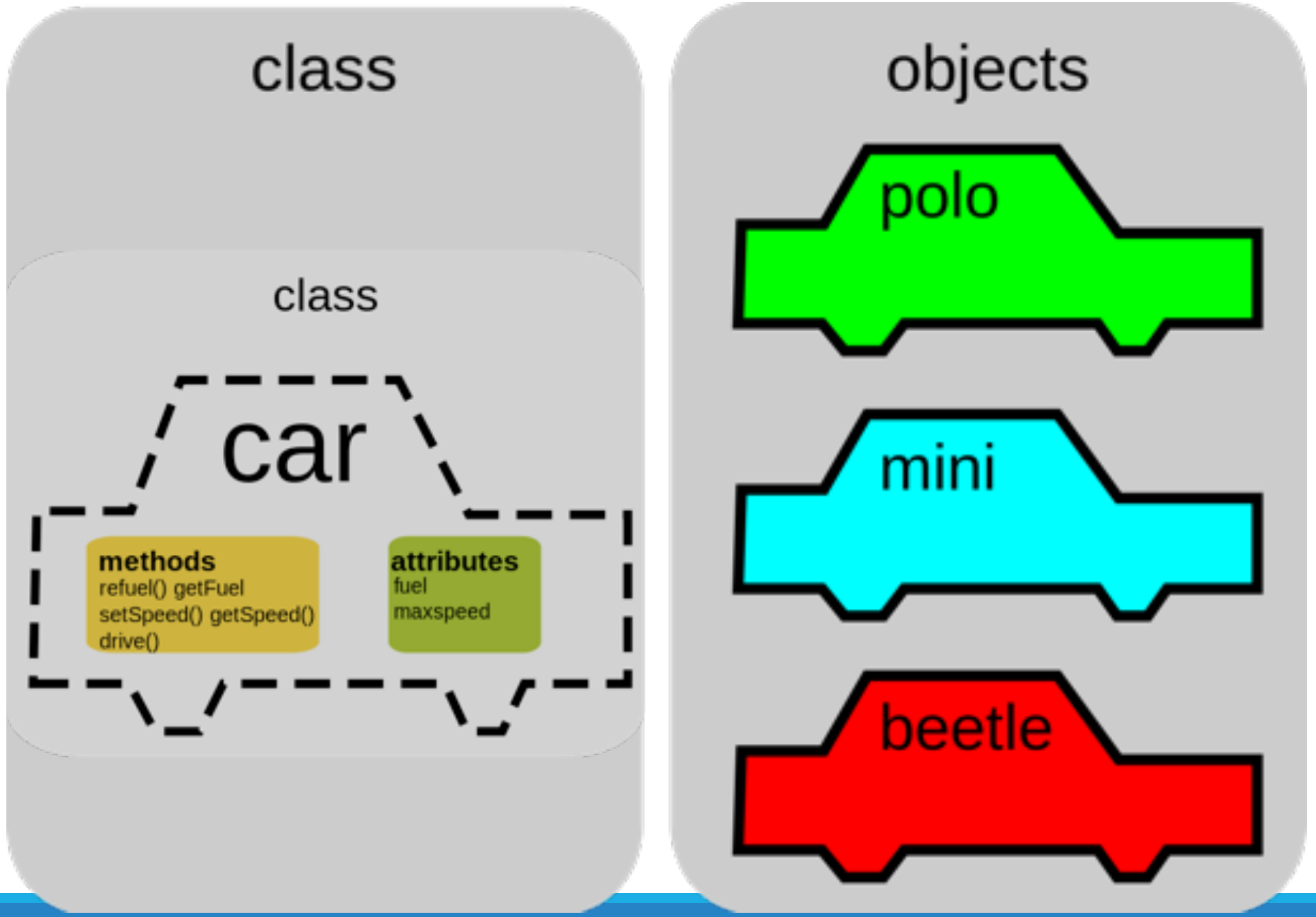
Figure 5-19 Discretization grid for 2-D circulation model.

Slingerland, Harbaugh, and Furlong (1994)

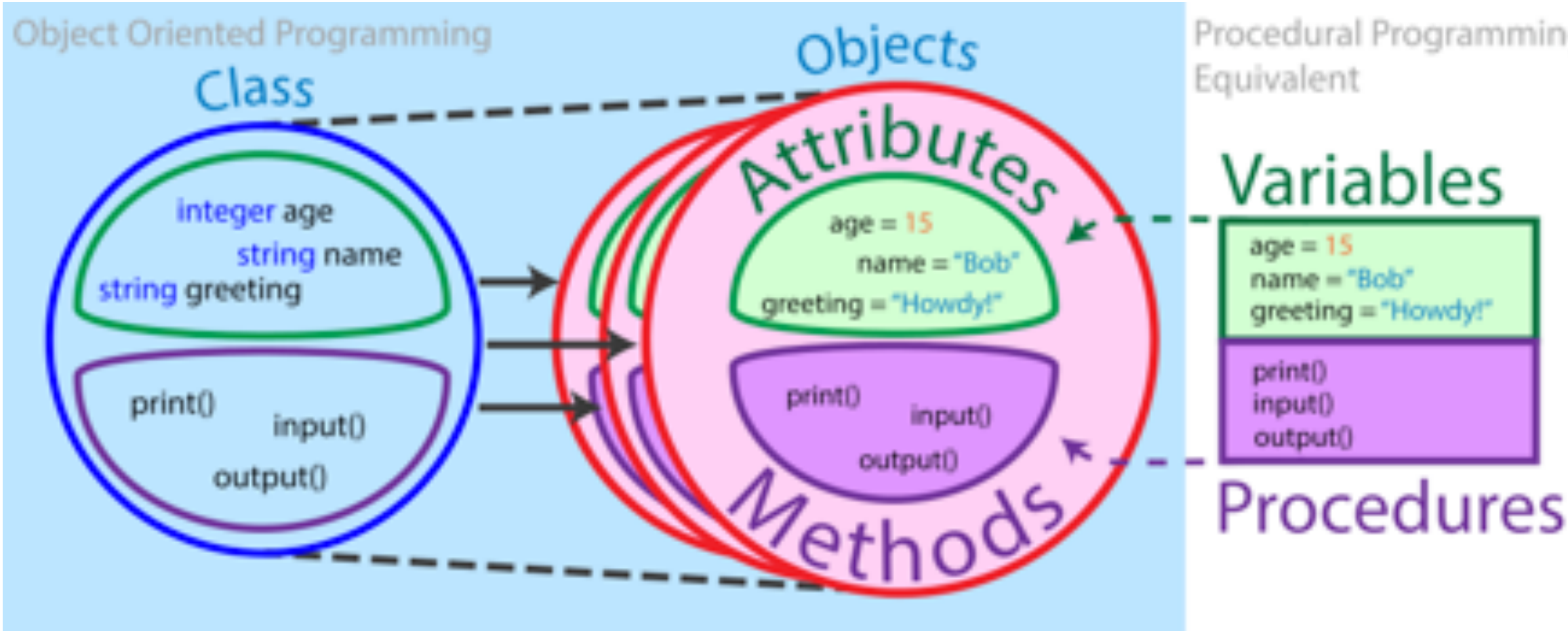
Quick side bar on Object Oriented Programing:

From Wikipedia: “**Object-oriented programming** (**OOP**) is a programming paradigm based on the concept of "objects", which may contain data, ... often known as *attributes;* and code, in the form of procedures, often known as *methods*. ” (https://en.wikipedia.org/wiki/Object-oriented_programming)



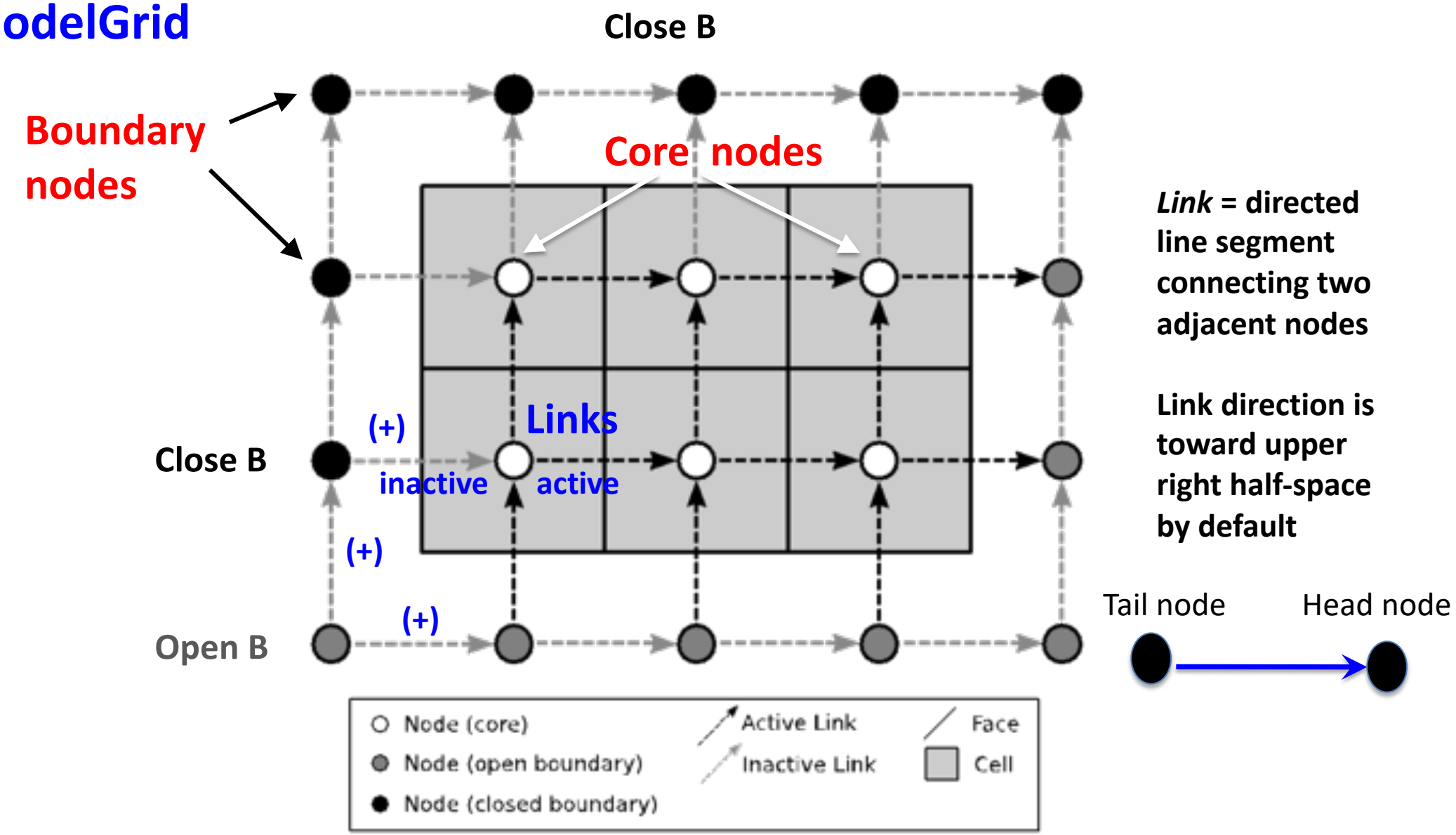


https://brilliant.org/wiki/classes-oop/



https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_programming/Elements_of_Object-Oriented_Programming

RasterModelGrid



Link **= directed line segment connecting two adjacent nodes**

Link direction is toward upper right half-space by default

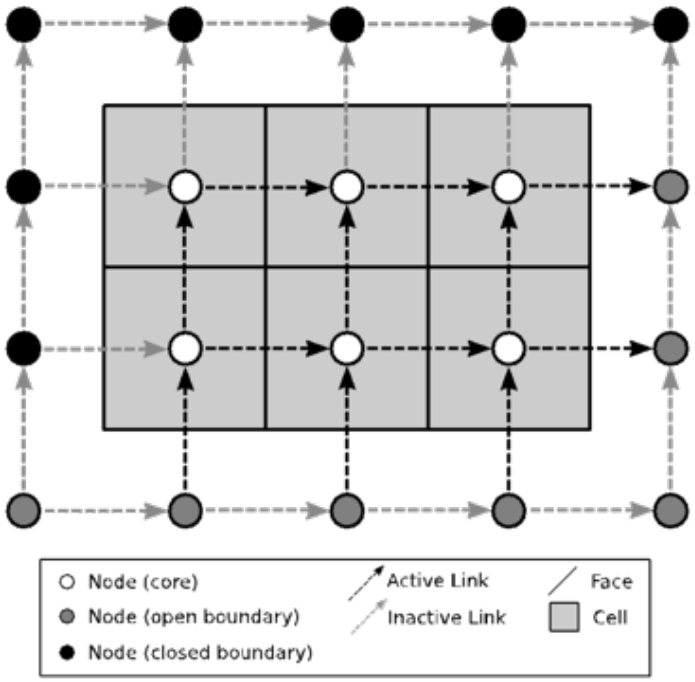
Tail node Head node

CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)
TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4

LANDLAB

RasterModelGrid example: Create a 4 by 5 grid

```
>> from landlab import RasterModelGrid
>> rg = RasterModelGrid((4,5),10.0)
```




```
>> rg.number_of_nodes
Out: 20
>> rg.number_of_links
Out: 31
>> rg.number_of_node_rows
Out: 4
>> rg.number_of_node_columns
Out: 5
>> rg.number_of_core_nodes
Out: 6
>> rg.number_of_cells
Out: 6
```

Fields: Attaching data to the grid

- A **field** is a NumPy array containing data that are associated with a particular type of grid element (typically nodes or links)
- Fields are **1D arrays**
- Values correspond to the element with the same ID. Example: value 5 of a node field belongs to node #5.
- Fields are "**attached**" to the grid (the grid object includes dictionaries listing all the fields)
- Fields have names (as strings)
- Create fields with grid functions `add_zeros`, `add_ones`, or `add_empty`

Example: attaching data to the grid

```
>> from landlab import RasterModelGrid
>> rg = RasterModelGrid((4,5),10.0)
```

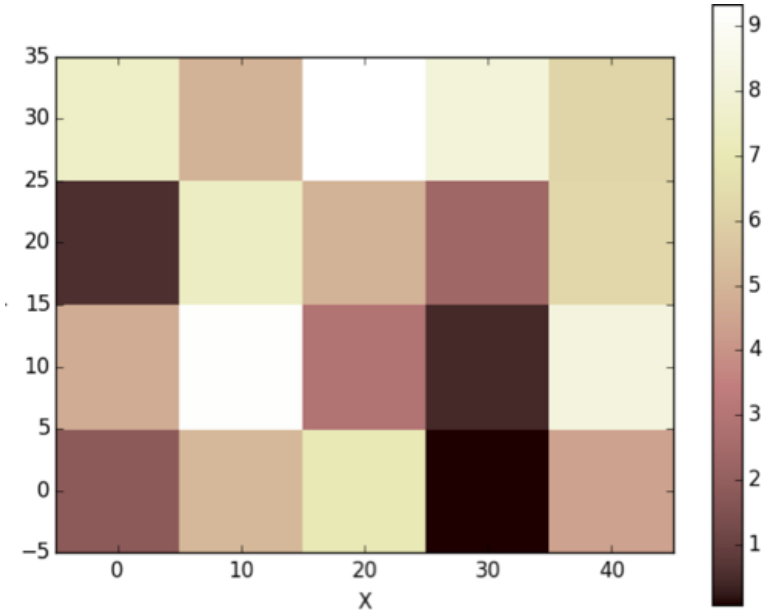
Create a random field and “attach” it to the grid as ‘elevation’

```
>> import numpy as np
>> z=10*np.random.rand(rg.number_of_nodes)
>> rg.add_field('node','topographic__elevation', z)
```

Plot the elevation field

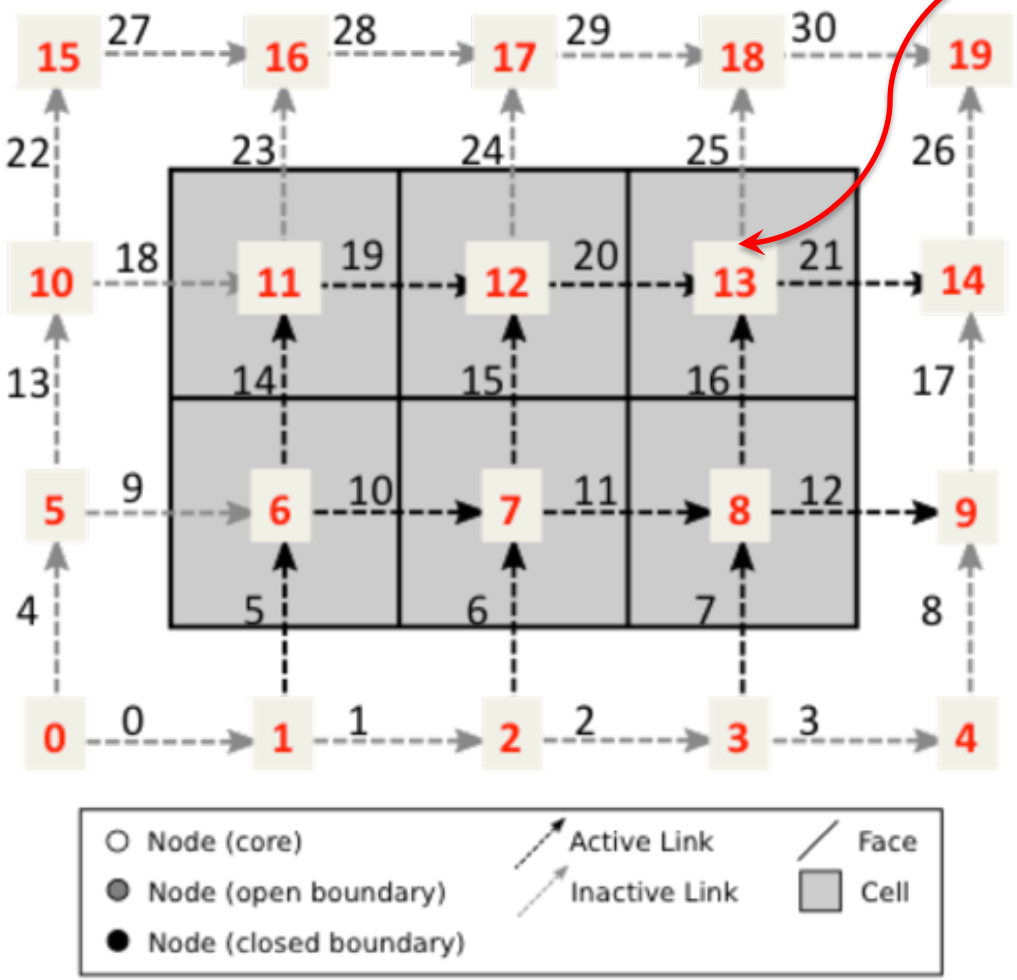
```
>> from landlab.plot import imshow_grid
>> import matplotlib.pyplot as plt
>> imshow_grid(rg, 'topographic__elevation')
>> plt.show()
```

```
>> rg.at_node.keys()
```



Explore node and link structure and “attached” data

Starting from the Southwest corner nodes are numbered to the East in each row.
Links are numbered using the number of their tail nodes.




```
>> rg.status_at_node[13]
Out: 0
>> rg.links_at_node[13]
Out: array([21, 25, 20, 16])
>> rg.link_dirs_at_node[13]
Out: array([-1, -1,  1,  1])
>> rg.node_at_link_tail[20]
Out: 12
>> rg.node_at_link_head[20]
Out: 13
```

```
>> rg.at_node['topographic__elevation'][13]
Out: 2.36
```

Now set ‘elevation’ to 100

```
>> rg.at_node[' 'topographic__elevation'][13]=100
```

OR: z[13]=100

```
>> rg.at_node[' 'topographic__elevation '][13]
Out: 100
```

CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)
TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4

LANDLAB

What Landlab provides

2. **Components and coupling framework for components**
   - A ***component*** models a **single process** (e.g., lithosphere flexure, linear diffusion, flow routing across terrain)
   - Components have a **standard interface** and can be combined by writing a **short Python script** (model, driver)
   - **Initialized** using input grid and parameters
   - **Update** relevant **'fields'** on the grid
   - Components also include analytical tools for analyzing landscapes (e.g. channel steepness, hillslope length, …)
   - Most components have a method "***run_one_step***" carries out the process.

Components

This section contains documentation and API reference information for the following categories of components:

Hillslope geomorphology

- LinearDiffuser: Model soil creep using "linear diffusion" transport law (no depth dependence)
- PerronNLDiffuse: Model soil creep using implicit solution to nonlinear diffusion law
- DepthDependentDiffuser: depth dependent diffusion after Johnstone and Hilley (2014)
- TransportLengthHillslopeDiffuser: Hillslope diffusion component in the style of Carretier et al. (2016), and Davy and Lague (2009)
- TaylorNonLinearDiffuser: Model non-linear soil creep after Ganti et al. (2013)
- DepthDependentTaylorDiffuser: Model depth dependent non-linear soil creep after Ganti et al. (2012) and Johnstone and Hilley (2014)

Fluvial geomorphology

- FastscapeEroder: Compute fluvial erosion using stream power theory ("fastscape" algorithm)
- StreamPower: Compute fluvial erosion using stream power theory (also uses "fastscape" algorithm but provides slightly different formulation and options)
- SedDepEroder: Compute fluvial erosion using using "tools and cover" theory
- StreamPowerSmoothThresholdEroder: Compute fluvial erosion using stream power theory with a numerically smoothed threshold
- DetachmentLtdErosion: Solve stream power equations, but without stability checks
- ErosionDeposition: Fluvial erosion in the style of Davy and Lague (2009)
- Stream Power with Alluvium Conservation and Entrainment

Flow routing

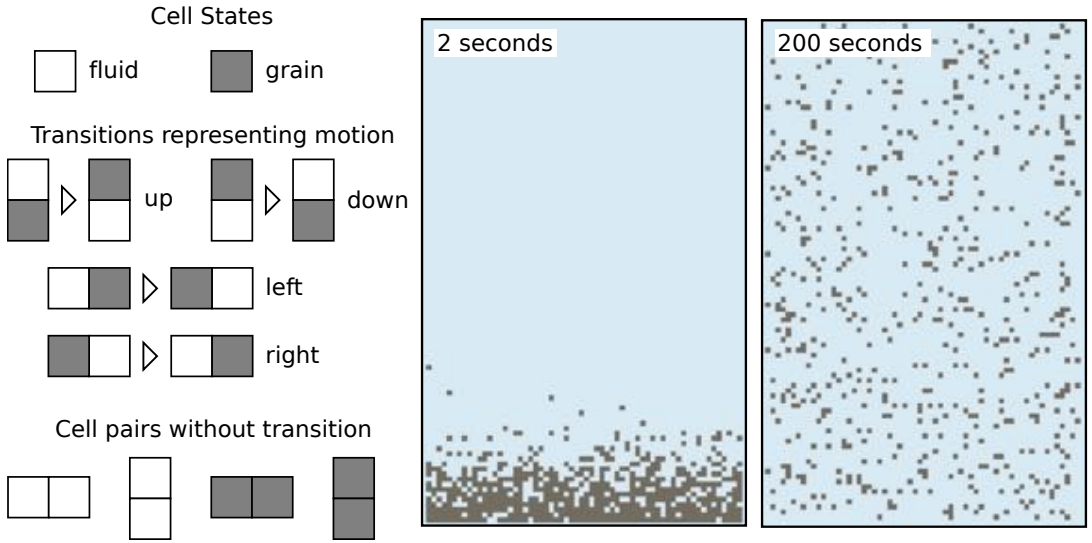
- The Landlab FlowDirectors: Components for Flow Direction
  - FlowDirectorSteepest
  - FlowDirectorD8
  - FlowDirectorMFD
  - FlowDirectorDinf

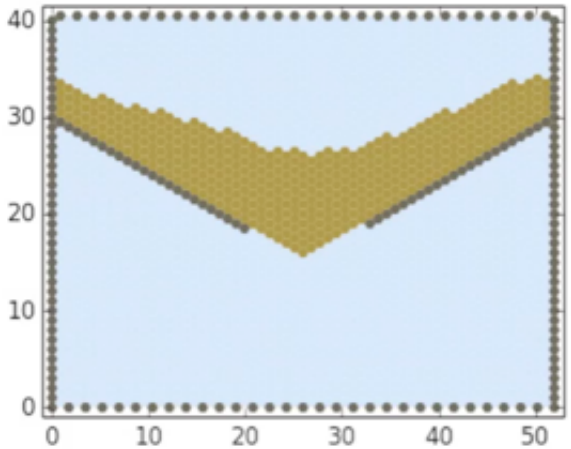
What Landlab provides

3. Input, output, visualization
    - **Read model parameters** from a formatted text file (optional)
    - **Read in** digital terrain data (e.g., **DEM**s) ➔ grid
    - **Write gridded output** to files (netCDF format)
    - **Plot** data using Matplotlib graphics library

What Landlab provides

4. Support for cellular-automaton modeling
   - CellLab-CTS: Continuous-time stochastic CA model “engine”

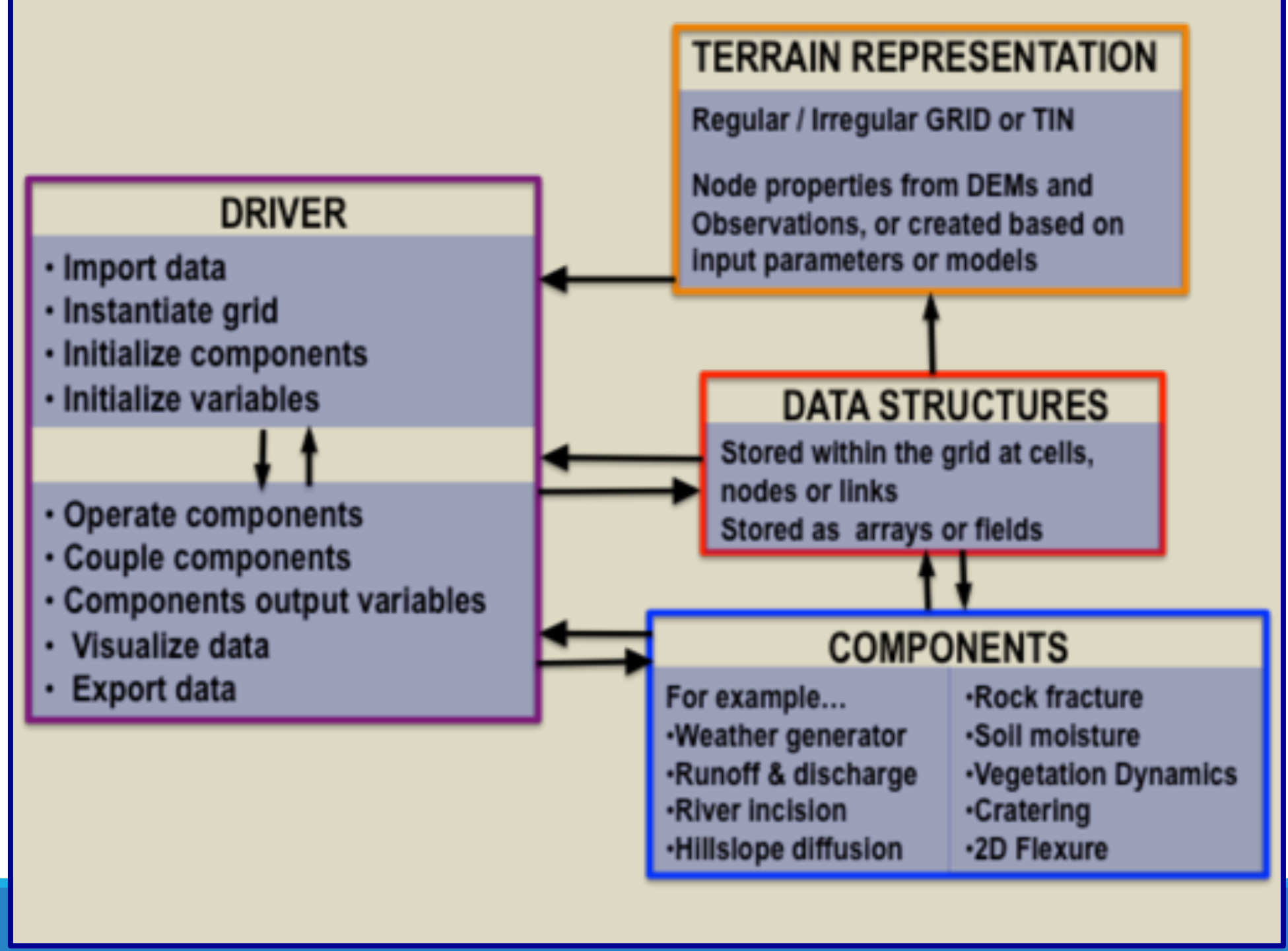




(Tucker et al., 2016 Geoscientific Model Development)

How do you create a Landlab model?

- Your **unique** creation.
- ***Usually*** a model will:
  - Operate on a **grid**;
  - **Create data**;
  - Use components to **update data values through time**;
  - **Visualize data**.
- but it **does not have to do *all* of these things**.

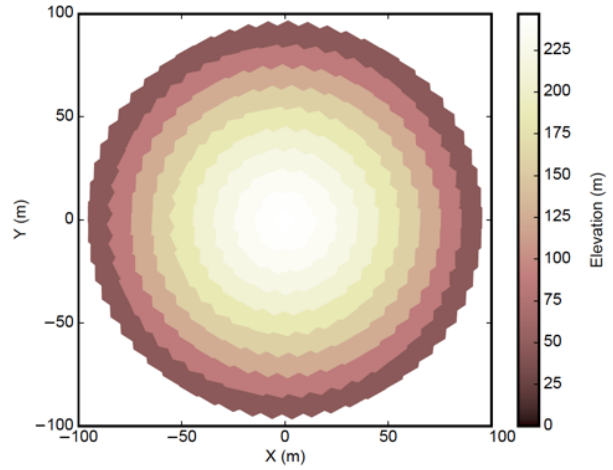

TERRAIN REPRESENTATION
Regular / Irregular GRID or TIN
Node properties from DEMs and Observations, or created based on input parameters or models
DRIVER
• Import data
• Instantiate grid
• Initialize components
• Initialize variables
• Operate components
• Couple components
• Components output variables
• Visualize data
• Export data
DATA STRUCTURES
Stored within the grid at cells, nodes or links
Stored as arrays or fields
COMPONENTS
For example...
•Weather generator
•Runoff & discharge
•River incision
•Hillslope diffusion
•Rock fracture
•Soil moisture
•Vegetation Dynamics
•Cratering
•2D Flexure

LANDLAB

Example single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> z = mg.add_zeros('node', 'topographic__elevation')
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



(Hobley et al., 2017, ESURF)

Linear Diffusion Equation

$$q_s = -K_d \nabla z$$

$$q_s = -K_d \frac{dz}{dx}$$

$$\frac{dz}{dt} = U - \frac{dq_s}{dx}$$

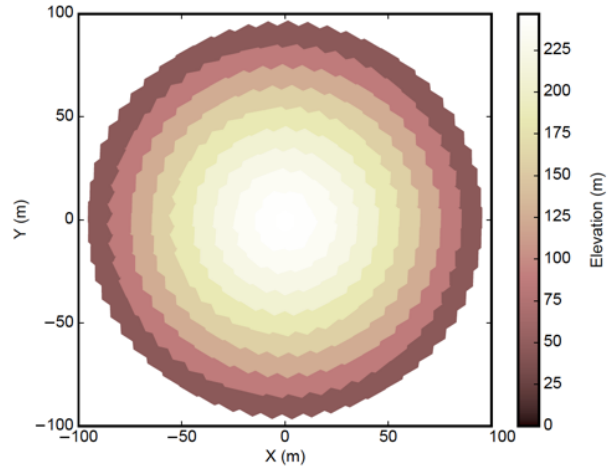
$z =$ elevation
$t =$ time
$U =$ rock uplift
$q_s =$ sediment transport rate
$x =$ distance downslope
$K_d =$ linear diffusivity

Single component diffusion model

Import Statements

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>> from matplotlib.pyplot import show
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic__elevation')
6. >>> dt = 2000.  # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```



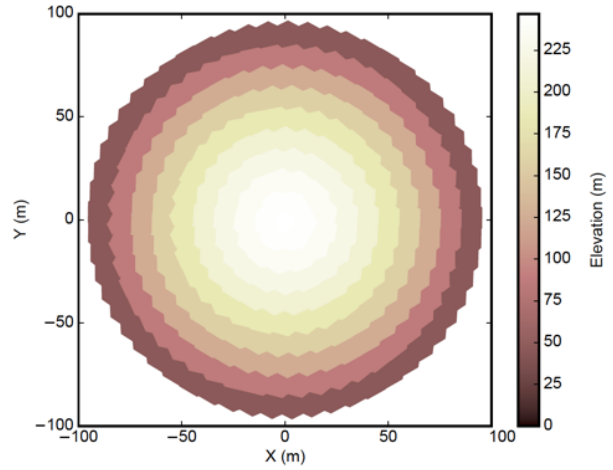
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

Instantiate model grid; Add data/create field

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic__elevation')
6. >>> dt = 2000.  # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```



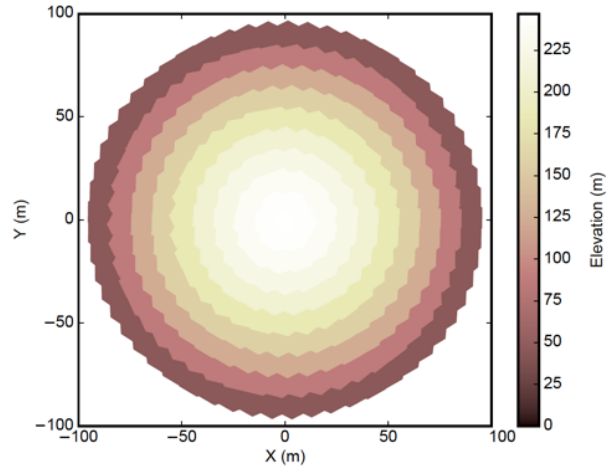
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```

Initialize parameter; Instantiate diffusion object



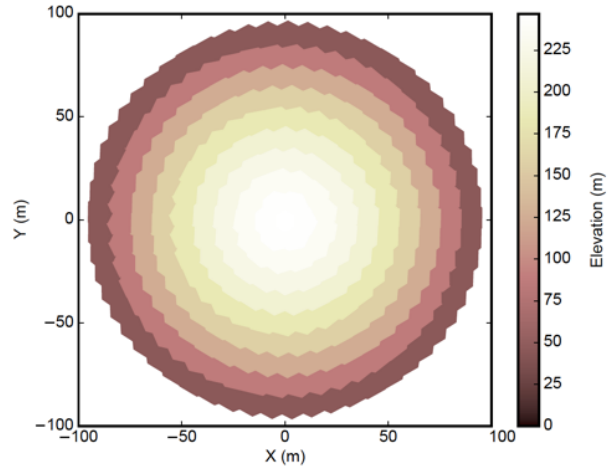
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>> from matplotlib.pyplot import show
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic__elevation')
6. >>> dt = 2000.  # no longer the stable timestep
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
0. ...     ld.run_one_step(dt)
Elevation (m)')
12. >>> show()
```

Time loop; Do the calculations!



(Hobley et al., 2017, ESURF)

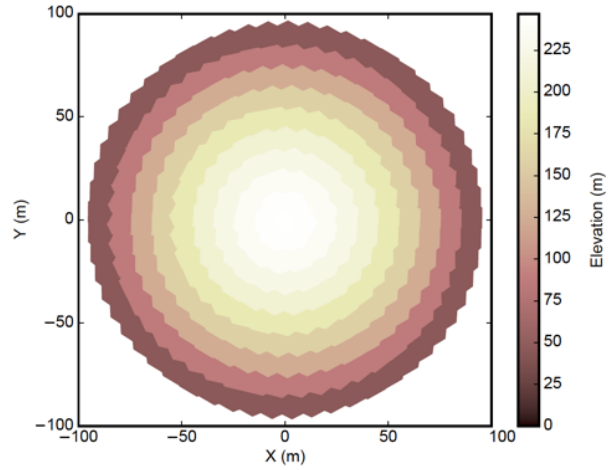
Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>> from matplotlib.pyplot import show
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic__elevation')
6. >>> dt = 2000.  # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
```

Visualize data

```
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```



(Hobley et al., 2017, ESURF)

Every model is some variant of the previous example

- Can be on the **command line** (as last example).
- Can be a **Python (.py) file** that is run from the command line.
- Can be an **Jupyter Notebook (.ipynb)**, which is great for teaching and learning.

Run some more models on CSDMS JypterHub

- Overland Flow
- Simple Landscape Evolution Model
- Landscape and life evolution