# Directional Slope

A Python Tool For Path Design

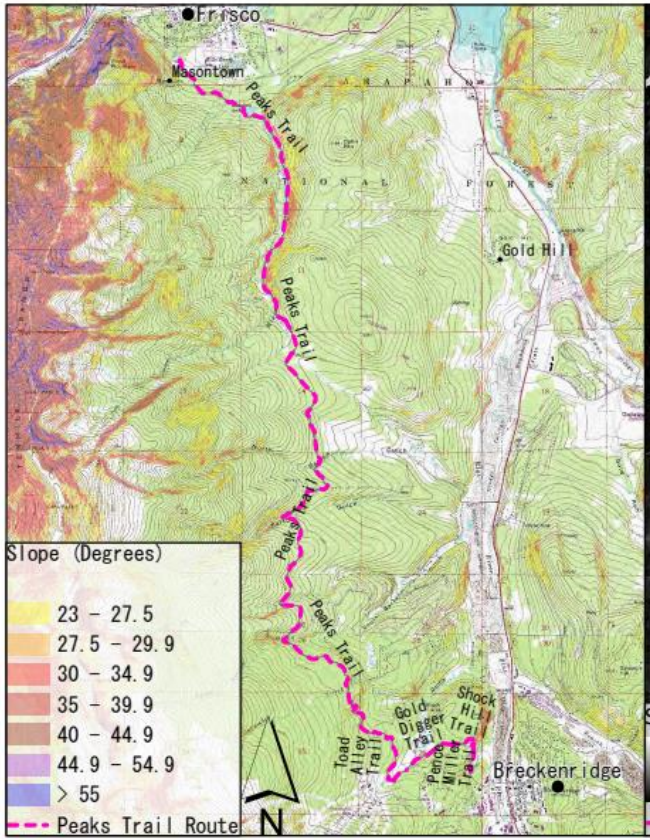
Joe Tayabji

```
def dir_slope(x, y, previous_x, previous_y):
    # transform x, y coordinates to pixel coordinates
    px, py = gdal.ApplyGeoTransform(dem_reverse_gt, x, y)
    px = floor(px)
    py = floor(py)
    # extract the dzdx and dzdy values
    dx, dy = get_horn(px, py)

    # calculate bearing in radians, must be a projected coordinate system
    delta_x = x - previous_x
    delta_y = y - previous_y
    bearing = atan2(delta_x, delta_y)
    bearing = (bearing + 2*pi) % (2*pi)
    tan_alpha = ((dx*sin(bearing)) + (dy * cos(bearing)))
    dir_slope = atan(tan_alpha)*(-180.0)/pi # this is the directional slope value!

    # add line to layer
    line_out_geom = ogr.Geometry(ogr.wkbLineString)
    line_out_geom.AddPoint(previous_x, previous_y)
    line_out_geom.AddPoint(x, y)
    line_out_feat = ogr.Feature(line_out_layer.GetLayerDefn())
    line_out_feat.SetGeometry(line_out_geom)
```

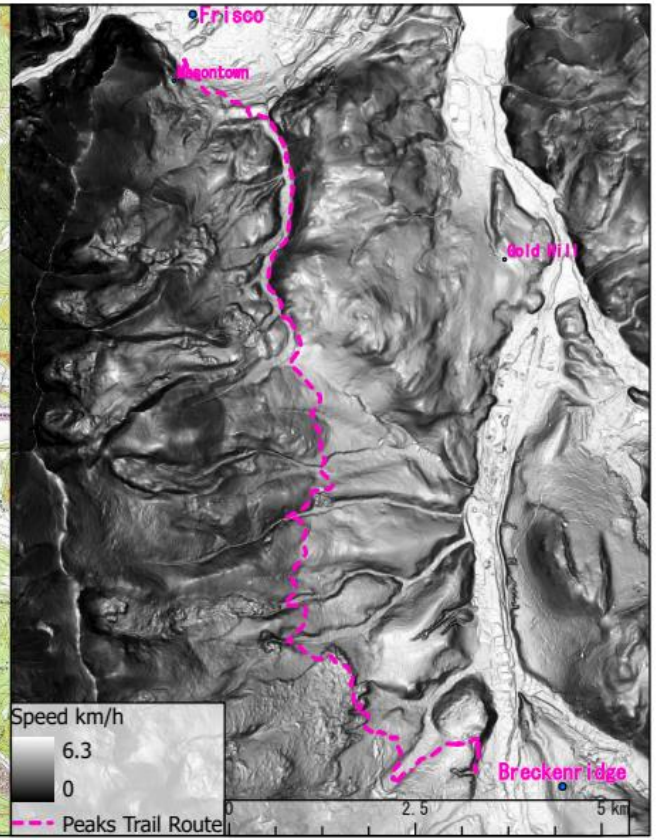
# Tobler's Hiking Function

## Peaks Trail Route, Frisco to Breckenridge

Slope Shading

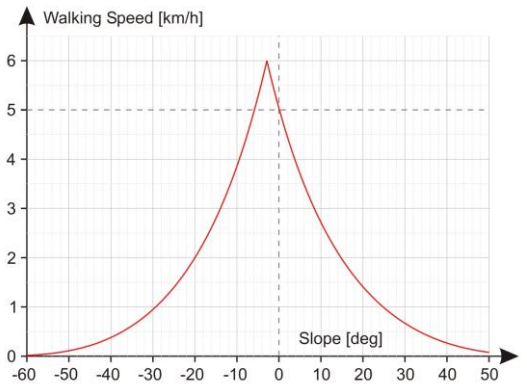



Travel Speed Cell Shading




Inspiration:

A personal project years ago where travel times were estimated using Tobler's Hiking Function.

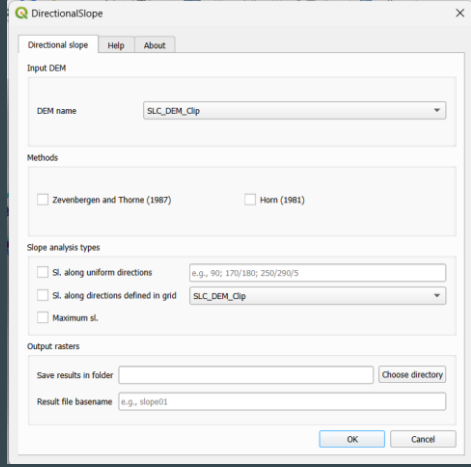


# Modern Day Problem

It many situations, it is useful to have the slope in a specific direction. For example, when traveling on foot or constructing new trails and roads, the slope at a specific point is of interest.

Wouldn't it be nice if there was a tool that took a **line** and a **DEM**, and gave you a **line** output with the slope added as a field? If it could break the line apart into increments of a **specified distance**? If it could be made using FOSS?

# What's out there now?




- Horn (1981): The method used when creating slope rasters
- ArcGIS: Slope tool
  slope_radians = ATAN ( √ ([**dz/dx**]^2 + [**dz/dy**]^2) )
- Neteler and Mitášová (2008): A mathematical formula for directional slope.
  *directional slope (alpha) = arctan [(**dz/dx**) * sin(alpha) + (**dz/dy**) * cos(alpha)]*
- Alberti (2017): A raster to raster QGIS tool using this formula

| | | |
|---|---|---|
| a | b | c |
| d | e | f |
| g | h | i |

# How can this be done?

- Import DEM
- Start by calculating slope as we have, except we need to keep **dz/dx** and **dz/dy** separate for now.
- Store both of these in a two band raster
- Take our input line and break it apart into points at the specified distances (e.g. 0.5 m)
- At each point calculate the bearing and retrieve dz/dx and dz/dy values
- Calculate directional slope at these points
- Reconstruct a new polyline feature class from these points

# What do we need in Python?

numpy

math

os

struct

gdal, ogr

shapely

# IT WORKS! But can it be better?

A 4236 x 5854 DEM with 0.5 m resolution was used. That's 24,797,544 cells of raster calculation. Raster calculations often take a while to complete; this one took 528.84 seconds (tested using time module).

In this example, the Living Room trail was used (right bycampus). The line was broken apart into 2,541 segments, but over 24 million calculations were made. Unlike the Alberti tool, we don't care about the whole raster. We care a line that is inside the raster extent. Surely, there has to be a better way.

# GDAL .ReadAsArray()

```
def get_horn(px, py):
    # Get a 3x3 NumPy array with DEM values centered at location of px, py
    local_dem = dem_raster.ReadAsArray(px-1, py-1, 3, 3)
```

This function returns a NumPy array. It needs the x and y coordinates of the “top left” corner of the data we want, and the width and length of the desired array. In this case, we only care about the eight cells immediately surrounded a point, so a 3x3 array is perfect. This is used to sample data from a raster.

What are px and py?
These are pixel coordinates, using the inverse geotransform of the DEM. More simply, they are the pixel the point is located on the DEM.

# Current Code

- numpy, math, os, gdal, ogr, shapely
- Load the DEM (and metadata) using gdal
- Load the line (and metadata) using ogr
- Set up the output line
- Go along the line, getting a point every 0.5 m (or any specified interval)
- At each point, calculate the bearing from the previous point
- Use the point's coordinates and get the dz/dx and dz/dy values
- Calculate the direction slope using the dz values and the bearing
- Add the slope to the schema, add line to feature class
- Repeat until finished

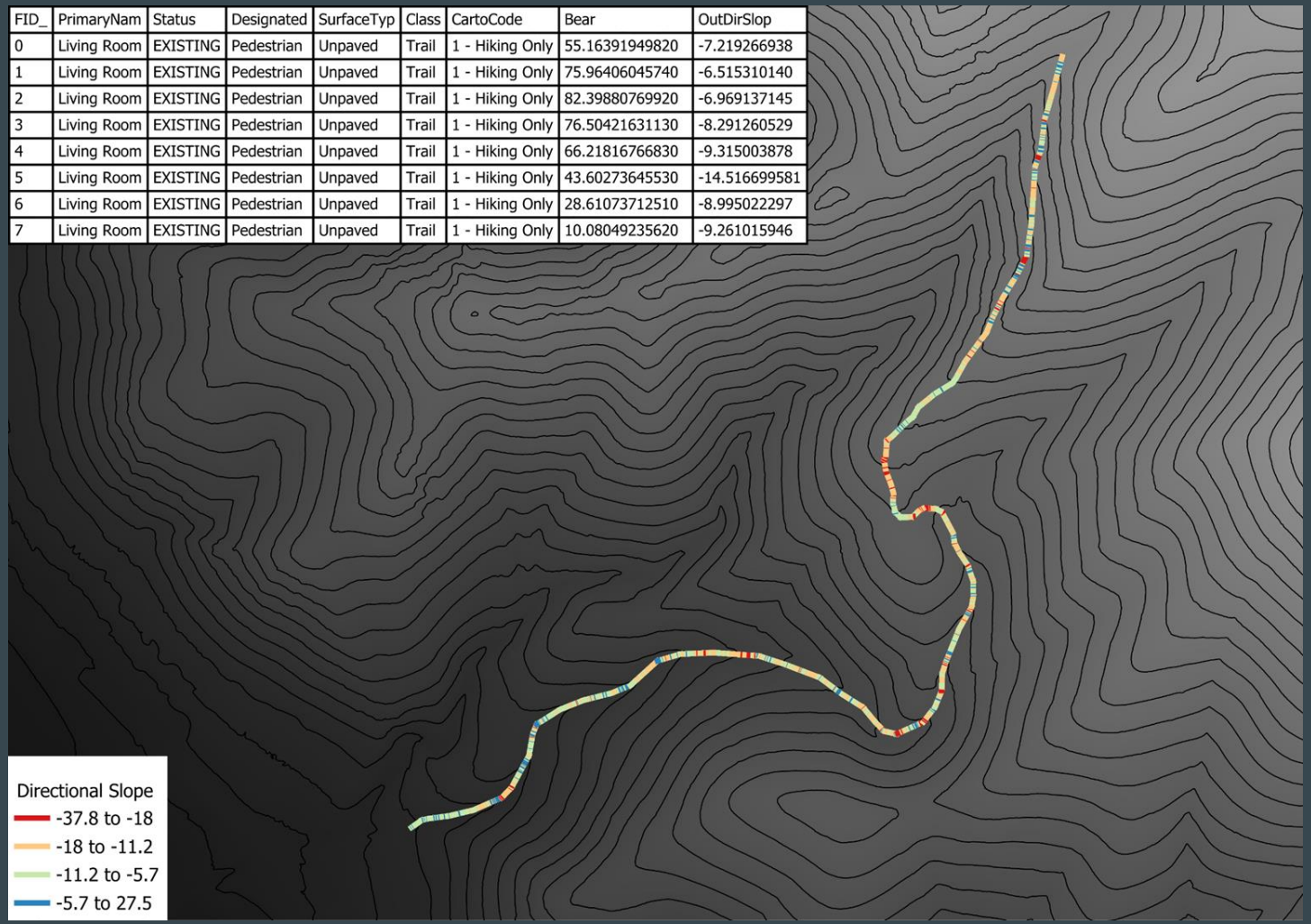
```
sample_interval = 0.5 # resampling distance
for p in line_in_layer:
    geom = p.GetGeometryRef().ExportToWkt()
    shapely_line = wkt.loads(geom)
    # Point at distance 0
    first_point = Point(list(shapely_line.coords)[0])
    previous_x = first_point.xy[0][0]
    previous_y = first_point.xy[1][0]

    # Go through the input line in increments equal to sample_interval
    line_length = shapely_line.length
    distance = sample_interval # initialize distance
    while distance < line_length:
        # get point coordinates for current location on the input line
        point = shapely_line.interpolate(distance)
        x, y = point.xy[0][0], point.xy[1][0]

        dir_slope(x, y, previous_x, previous_y)

        # set up for next iteration
        distance += sample_interval
        previous_x = x
        previous_y = y

    # do the above steps one last time for the last point on the line
    last_point = Point(list(shapely_line.coords)[-1])
    x, y = last_point.xy[0][0], last_point.xy[1][0]
    dir_slope(x, y, previous_x, previous_y)
```

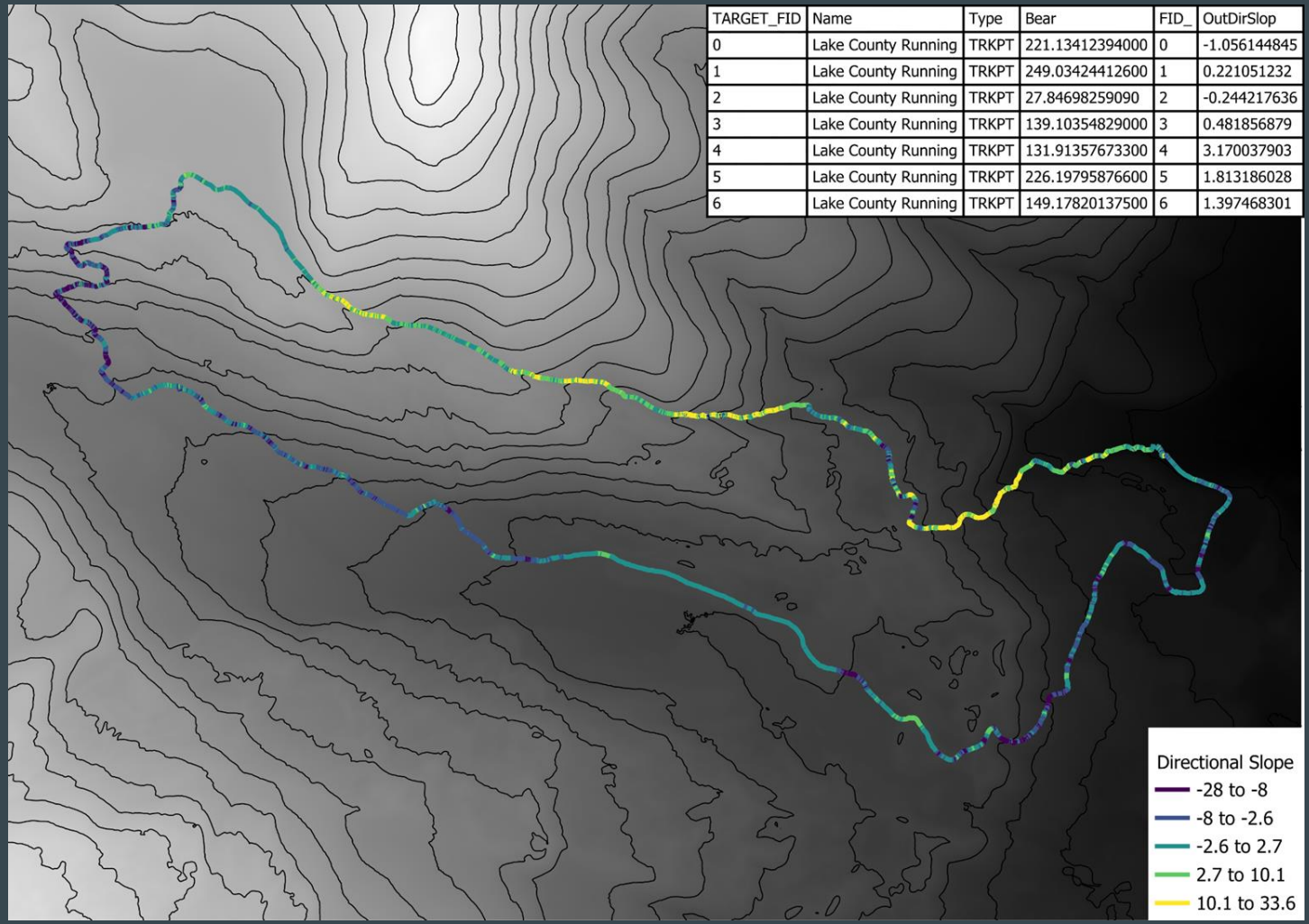
```
def get_horn(px, py):
    # Get a 3x3 NumPy array with DEM values centered at location of px, py
    local_dem = dem_raster.ReadAsArray(px-1, py-1, 3, 3)
    # top left = a
    a = local_dem[0, 0]
    # top center = b
    b = local_dem[0, 1]
    # top right = c
    c = local_dem[0, 2]
    # left = d
    d = local_dem[1, 0]
    # the pixel = e
    e = local_dem[1, 1]
    # right = f
    f = local_dem[1, 2]
    # bottom left = g
    g = local_dem[2, 0]
    # bottom center = h
    h = local_dem[2, 1]
    # bottom right = i
    i = local_dem[2, 2]

    # right column - left column
    dz_dx = ((c+(2*f)+i)-(a+(2*d)+g))/(8*cell_size)
    # top row - bottom row
    dz_dy = ((c+(2*b)+a)-(i+(2*h)+g))/(8*cell_size)

    return dz_dx, dz_dy
```

```
def dir_slope(x, y, previous_x, previous_y):
    # transform x, y coordinates to pixel coordinates
    px, py = gdal.ApplyGeoTransform(dem_reverse_gt, x, y)
    px = floor(px)
    py = floor(py)
    # extract the dzdx and dzdy values
    dx, dy = get_horn(px, py)

    # calculate bearing in radians, must be a projected coordinate system
    delta_x = x - previous_x
    delta_y = y - previous_y
    bearing = atan2(delta_x, delta_y)
    bearing = (bearing + 2*pi) % (2*pi)
    tan_alpha = ((dx*sin(bearing)) + (dy * cos(bearing)))
    dir_slope = atan(tan_alpha)*(-180.0)/pi # this is the directional slope value!

    # add line to layer
    line_out_geom = ogr.Geometry(ogr.wkbLineString)
    line_out_geom.AddPoint(previous_x, previous_y)
    line_out_geom.AddPoint(x, y)
    line_out_feat = ogr.Feature(line_out_layer.GetLayerDefn())
    line_out_feat.SetGeometry(line_out_geom)
```

13.32 seconds

| FID_ | PrimaryNam | Status | Designated | SurfaceTyp | Class | CartoCode | Bear | OutDirSlop |
|---|---|---|---|---|---|---|---|---|
| 0 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 55.16391949820 | -7.219266938 |
| 1 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 75.96406045740 | -6.515310140 |
| 2 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 82.39880769920 | -6.969137145 |
| 3 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 76.50421631130 | -8.291260529 |
| 4 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 66.21816766830 | -9.315003878 |
| 5 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 43.60273645530 | -14.516699581 |
| 6 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 28.61073712510 | -8.995022297 |
| 7 | Living Room | EXISTING | Pedestrian | Unpaved | Trail | 1 - Hiking Only | 10.08049235620 | -9.261015946 |






| TARGET_FID | Name | Type | Bear | FID_ | OutDirSlop |
|---|---|---|---|---|---|
| 0 | Lake County Running | TRKPT | 221.13412394000 | 0 | -1.056144845 |
| 1 | Lake County Running | TRKPT | 249.03424412600 | 1 | 0.221051232 |
| 2 | Lake County Running | TRKPT | 27.84698259090 | 2 | -0.244217636 |
| 3 | Lake County Running | TRKPT | 139.10354829000 | 3 | 0.481856879 |
| 4 | Lake County Running | TRKPT | 131.91357673300 | 4 | 3.170037903 |
| 5 | Lake County Running | TRKPT | 226.19795876600 | 5 | 1.813186028 |
| 6 | Lake County Running | TRKPT | 149.17820137500 | 6 | 1.397468301 |

# Ongoing and Future Work

- Turn this into a QGIS processing tool that can be run in the GUI
- Publish and share
- Update arcpy version for efficiency
- Testing on inputs with multiple trails
- Add other slope methods