Determining the Optimal DEM Resolution from Lidar Data

Taylor Smith (tasmith@uni-potsdam.de)
Aljoscha Rheinwalt (aljoscha.rheinwalt@uni-potsdam.de)
Bodo Bookhagen (bookhage@uni-potsdam.de)

Institute for Geosciences, Universität Potsdam, Germany

Working example manual with reference to the paper:

T Smith, A Rheinwalt, and B Bookhagen. "Determining the Optimal Grid Resolution for Topographic Analysis on an Airborne Lidar Dataset", Earth Surface Dynamics 7 (2019): 475-489
https://doi.org/10.5194/esurf-7-475-2019.

Contents

1 Background and Concept **3**

2 Synthetic Example **4**
2.1 Uncertainty in DEMs 9
2.2 Calculating Uncertainty and Error Directly on the Grid 10

3 Real World Data – Santa Cruz Island **11**
3.1 Calculating the Slope and Aspect Errors and Uncertainties 11
3.2 Choosing a DEM Resolution 13

4 Further Steps – Subsetting the Data **15**

List of Figures

1 Exponentially curved hill showing two possible slope values – one that only considers the top and bottom points on the 'hill', and one that splits the hill into two sections. . 3
2 Exponentially curved hills with slightly different scaling – the solid black line runs from 0 to 1, and the dashed line runs from -0.1 to 1.1. Blue solid and line shows the slope estimates from the ends to the middle for the [0,1] line and red shows the [-0.1, 1.1] line. 4
3 Gaussian Hill with a max elevation of one. 4
4 Gaussian Hill slope and aspect biases. 8
5 Fine-gridded curve. 8
6 Interplay between grid resolution and spatially uncorrelated white noise, taken from Smith et al. (2019 9
7 Quality ratio magnitudes based on a set of grid spacings and noise levels, taken from Smith et al. (2019 10
8 Slope and aspect quality ratios for the Pozo catchment at 10m spatial resolution. . . . 12
9 Slope and aspect quality ratios for the Pozo catchment across resolutions from 2m to 30m. 14
10 Optimal slope grid resolution across SCI. Taken from Smith et al. (2019). 15
11 Optimal grid resolution for minimizing error bounds on slope and aspect calculations across SCI compared to catchment median slope. Point sizes are scaled to catchment size, from 0.1 to 34 km^2. Dashed lines show the 25th percentile to 75th percentile of optimal grid resolution, colored by the standard deviation of that point. Stars show the whole-island optimal resolution and median slope. There exists a clear trend where lower median slopes lead to lower optimal grid resolutions. Standard deviations also tend to be smaller in these catchments, due to higher DEM fidelity over flat terrain. Catchments with low (>10 meter) optimal grid resolutions are small and have complex topography. Taken from Smith et al. (2019) 16

1 Background and Concept

Many geographic analyses require gridded data – for example, a raster showing elevation, rainfall, etc. Treating geographic data as an image or matrix makes a lot of calculations simpler and faster, and most tools are geared towards this use case. However, when dealing with lidar data, we do not have a pre-defined grid; data is available at pseudorandom points across our study area.

This then invites the question of what resolution is the 'best' for analysis? Unfortunately, there is no true measure of 'best' in this context – the ideal resolution is highly analysis specific. For example, to analyzing microtopography is not really possible with a 10m DEM. Likewise, that same 10m DEM might contain small-scale topography that hinders statistical analysis of features that occur at the kilometer scale.

Slope and aspect are commonly used terrain derivatives in hydrology, ecology, geomorphology, and a range of other disciplines. They are usually defined and calculated on a grid (e.g., slope is a unit of elevation change over a fixed distance), and are sensitive to the choice of grid resolution. Imagine, for example, a curved hill, as in Figure 1.

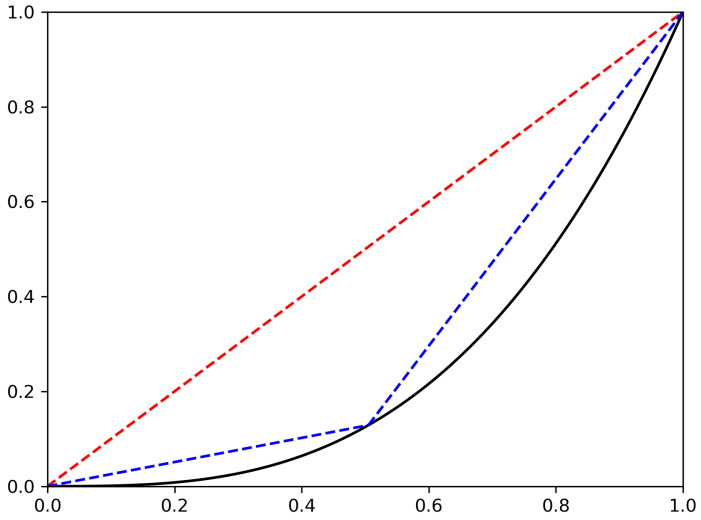

Figure 1: Exponentially curved hill showing two possible slope values – one that only considers the top and bottom points on the 'hill', and one that splits the hill into two sections.

It is clear that choosing a grid size of one unit (red line) and one half unit (blue lines) would result in very different slope values. Furthermore, it is clear that neither choice of grid resolution really captures the true slope of the hill – they both overestimate the slope in some areas, and underestimate it in others.

The picture becomes even more complicated when we introduce noise or uncertainty into our elevation estimates. How can we be sure that the slope value we capture from a grid actually measures a difference between the highest and lowest points in a given area? A simplified version of this effect can be seen in Figure 2.

These two sources of uncertainty – grid spacing and elevation uncertainty – are propagated into any terrain derivative (e.g., slope, aspect, curvature, etc). This can be shown with synthetic data, and also applied to real-world data.

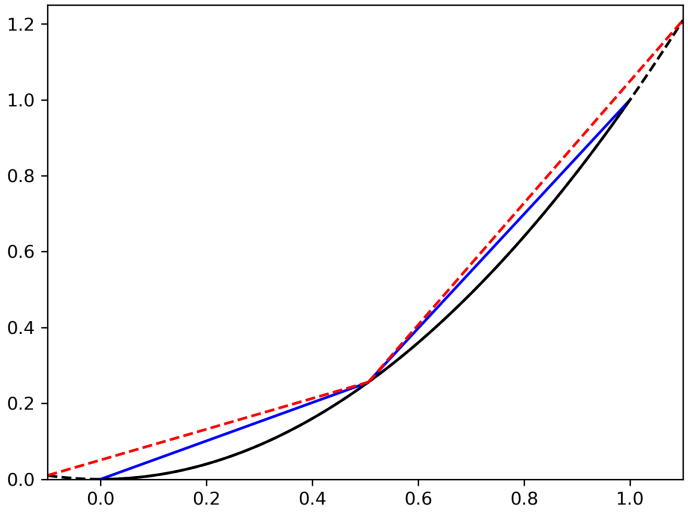

Figure 2: Exponentially curved hills with slightly different scaling – the solid black line runs from 0 to 1, and the dashed line runs from -0.1 to 1.1. Blue solid and line shows the slope estimates from the ends to the middle for the [0,1] line and red shows the [-0.1, 1.1] line.

It is important to note that this method does not find the 'best' DEM resolution for every application, but rather seeks to minimize the errors in slope and aspect estimates for a given elevation dataset.

2 Synthetic Example

Before diving into real-world data, it is instructive to look at a simplified case. We will use a Gaussian Hill, with elevations defined on a regular square grid as $z = e^{(-x^2-y^2)}$ for each (x, y) point. The desired output is something like Figure 3.

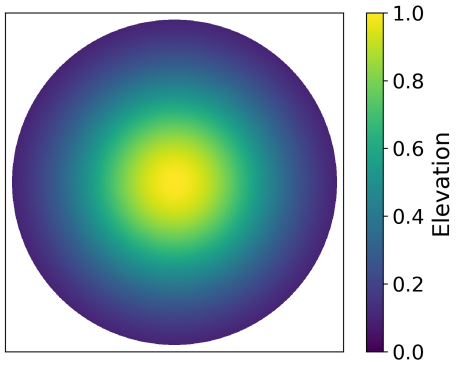

Figure 3: Gaussian Hill with a max elevation of one.

Let's do this in python:

```
import numpy as np
import matplotlib.pyplot as plt

def make_hill(n):
   # sym grid limits for area 10
   r0 = np.sqrt(10) / 2
   xmin, xmax = -r0, r0
   ymin, ymax = -r0, r0

   # cell boundaries
   xb = np.linspace(xmin, xmax, n+1)
   yb = np.linspace(ymin, ymax, n+1)
   #xx, yy = np.meshgrid(xb, yb)
   spacing = abs(xb[1]-xb[0])

   # cell center
   x = xb[:-1] + 0.5 *abs(xb[1]-xb[0])
   y = yb[:-1] + 0.5 *abs(yb[1]-yb[0])

   x, y = np.meshgrid(x, y)
   z = np.exp(-x*x-y*y)

   return x, y, z, spacing

x, y, z, spacing = make_hill(1001)
mask = np.where(z < 0.1)
z[mask] = np.nan

plt.close('all')
f, ax = plt.subplots(1)

cb = ax.imshow(z, vmin=0, vmax=1)
cb = plt.colorbar(cb, ax=ax)
cb.ax.tick_params(labelsize=16)
cb.set_label('Elevation', fontsize=18)
ax.set_xticks([])
ax.set_yticks([])
```

Since the Gaussian Hill is a differentiable function, we know what the slope at each point is simply by taking the first derivative of the surface. Again, let's do that in python:

```
def slope_hill(x, y):
   mag = 2 *np.sqrt(x*x+y*y) *np.exp(-x*x-y*y)
   return np.arctan(mag) *180 / np.pi

def aspects_hill(x, y):
   return np.arctan2(y, -1*x) *180 / np.pi + 180

x, y, z, spacing = make_hill(1001)
mask = np.where(z < 0.1)
z[mask] = np.nan
slp = slope_hill(x, y)
asp = aspects_hill(x, y)
slp[mask] = np.nan
asp[mask] = np.nan

f, (ax, ax2) = plt.subplots(1,2)

cb = ax.imshow(slp, vmin=0, vmax=40, cmap=plt.cm.RdBu)
cb = plt.colorbar(cb, ax=ax)
cb.ax.tick_params(labelsize=12)
cb.set_label('Slope', fontsize=14)
ax.set_xticks([])
ax.set_yticks([])

cb = ax2.imshow(asp, vmin=0, vmax=360, cmap=plt.cm.inferno)
cb = plt.colorbar(cb, ax=ax2)
cb.ax.tick_params(labelsize=12)
cb.set_label('Aspect', fontsize=14)
ax2.set_xticks([])
ax2.set_yticks([])
```

On a grid, slope $S(x, y)$ and aspect $A(x, y)$ are defined as ,

$$S(x,y) = \frac{180^\circ}{\pi} \times \arctan \sqrt{\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2} \tag{1}$$

and

$$A(x,y) = \frac{180^\circ}{\pi} \times \arctan\left(\frac{\partial z}{\partial y} \times \frac{\partial x}{\partial z}\right) + 180^\circ \tag{2}$$

with x and y being the spatial coordinates, and $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ being the directional derivatives.

Fortunately, taking those directional derivatives in python is simple. We can define two functions, one for slope and one for aspect using the `numpy` package.

```
from mpl_toolkits.axes_grid1 import make_axes_locatable

def colorbar(mappable):
   ax = mappable.axes
   fig = ax.figure
   divider = make_axes_locatable(ax)
   cax = divider.append_axes("right", size="5%", pad=0.05)
   return fig.colorbar(mappable, cax=cax, extend='both')

def aspects_np(z, spacing):
   y, x = np.gradient(z, spacing)
   return np.arctan2(-1*y, x) *180 / np.pi + 180

def slopes_np(z, spacing):
   y, x = np.gradient(z, spacing)
   mag = np.sqrt(x*x + y*y)
   return np.arctan(mag) *180 / np.pi

x, y, z, spacing = make_hill(1001)
mask = np.where(z < 0.1)
z[mask] = np.nan
slp = slope_hill(x, y)
asp = aspects_hill(x, y)
slp[mask] = np.nan
asp[mask] = np.nan

slp_np, asp_np = slopes_np(z, spacing), aspects_np(z, spacing)
slp_dif = slp_np -slp
asp_dif = asp_np -asp

plt.close('all')
f, (ax, ax2) = plt.subplots(1,2)

vmin, vmax = np.nanpercentile(slp_dif, [5, 95])
cb = ax.imshow(slp_dif, vmin=vmin, vmax=vmax, cmap=plt.cm.seismic)
cb = colorbar(cb)
cb.formatter.set_powerlimits((-1, 4))
cb.update_ticks()
cb.set_label('Slope Difference (degree)', fontsize=18)
cb.ax.tick_params(labelsize=16)
cb.ax.yaxis.offsetText.set_fontsize(16)
ax.set_xticks([])
ax.set_yticks([])

vmin, vmax = np.nanpercentile(asp_dif, [5, 95])
cb = ax2.imshow(asp_dif, vmin=vmin, vmax=vmax)
cb = colorbar(cb)
cb.formatter.set_powerlimits((-1, 4))
cb.update_ticks()
cb.set_label('Aspect Difference (degree)', fontsize=18)
cb.ax.tick_params(labelsize=16)
cb.ax.yaxis.offsetText.set_fontsize(16)
ax2.set_xticks([])
ax2.set_yticks([])
```

This gives us a difference grid, like the one below:

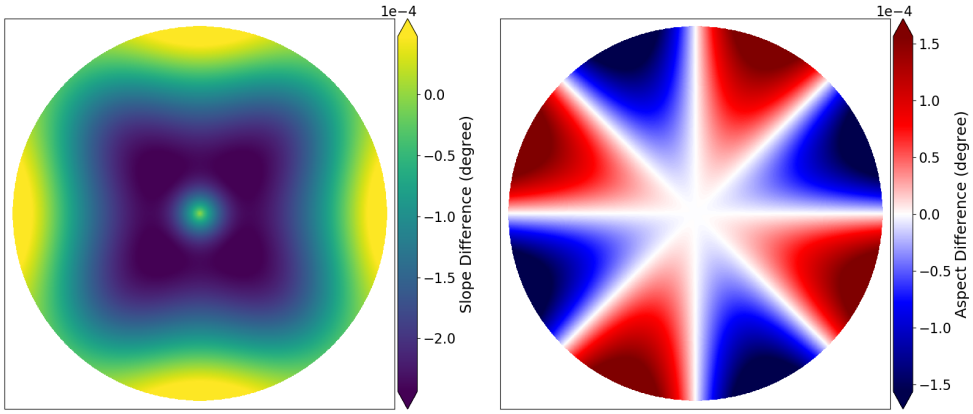

Figure 4: Gaussian Hill slope and aspect biases.

The magnitude of the errors in slope and aspect is directly related to the gridding – we can never represent the perfect, mathematically defined, curve of a Gaussian Hill with a series of discrete steps. Look again at Figure 1 – the blue lines better represent the curve of the hill, but still do not capture the true shape of things. If we break things down further, for example by gridding every 0.1 degrees, we start to get a more accurate curve:

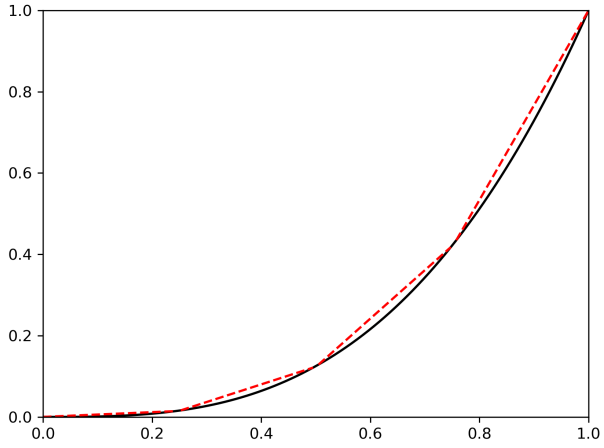

Figure 5: Fine-gridded curve.

In essence, the finer our grid is, the closer we get to the 'perfect' mathematical definition of slope and aspect. If there were no noise in our elevation dataset, a smaller grid size would result in more accurate slope and aspect estimations.

2.1 Uncertainty in DEMs

Unfortunately, real-world elevation data will have some uncertainty due to sensor limitations and measurement biases. Even with incredibly dense lidar data, we will still have extremely fine-scale topography which is not perfectly captured. With 100 points/sq m, we could still have cracks in the soil, slight differences in roughness, etc which would impact our estimates of slope and aspect.

Given a known uncertainty for a grid, we can propagate the elevation uncertainty forward into our slope and aspect calculations. **A key point to make here is that very fine grid sizes see larger impacts from noise.** For example, one meter of vertical noise may not be visible if you are looking at a 90m DEM, but would render a 10cm DEM completely useless. If we consider a range of grid sizes and a range of noise values for our synthetic example, we get two curves – one for the gridding error, termed here the truncation error (TE), and one for the propagated uncertainty, which we refer to as propagated elevation uncertainty (PEU).

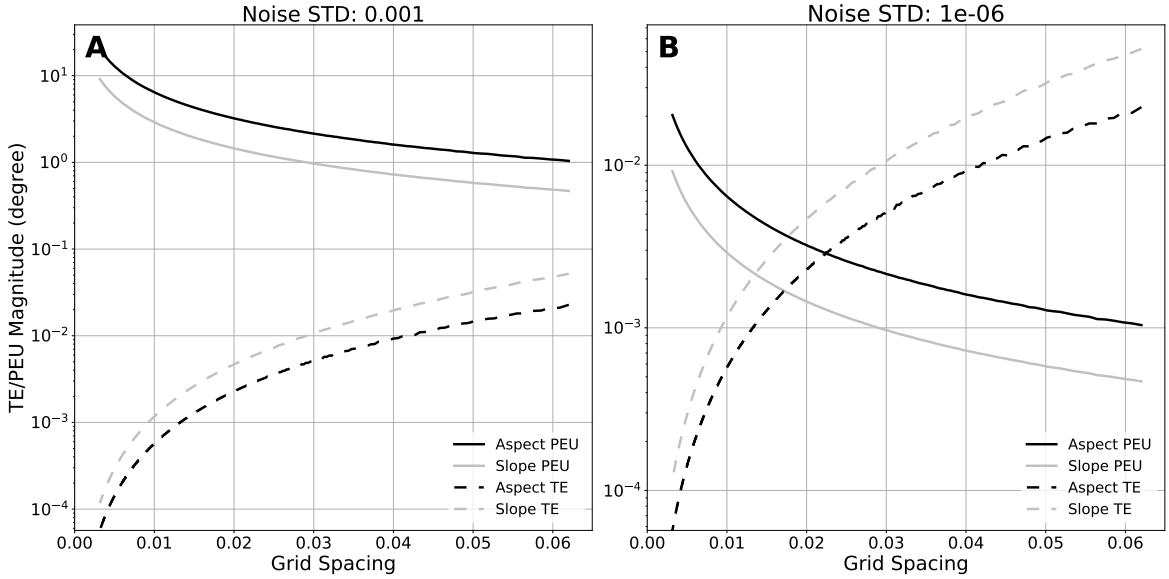

Figure 6: Interplay between grid resolution and spatially uncorrelated white noise, taken from Smith et al. (2019

As can be seen in Figure 6, for very low noise levels (right panel), grid spacing will create larger uncertainties than the noise. However, for larger noise levels (left panel), gridding error is relatively less important. Which source of error is dominant is determined by the magnitude of elevation uncertainty relative to the size of the grid.

2.2 Calculating Uncertainty and Error Directly on the Grid

Both TE and PEU can be derived directly, given a paired elevation and elevation uncertainty dataset. For more details on the derivation of these functions, refer to the Smith et al. (2019) publication listed on the title page.

The function definitions for TE and PEU on a grid can be found here: https://github.com/UP-RS-ESP/TopoMetricUncertainty/blob/master/uncertainty.py

Since both error sources contribute to the overall 'goodness' of slope and aspect estimations, we can define a Quality Ratio (QR) that is maximized when the combined magnitude of TE and PEU is minimized. We define this QR as:

$$QR = \left(\frac{1}{1 + (m \times |TE|)}\right) \times \left(\frac{1}{1 + (m \times |PEU|)}\right) \tag{3}$$

where m is a normalization factor which accounts for the four-fold difference in the range of possible values between slope and aspect.

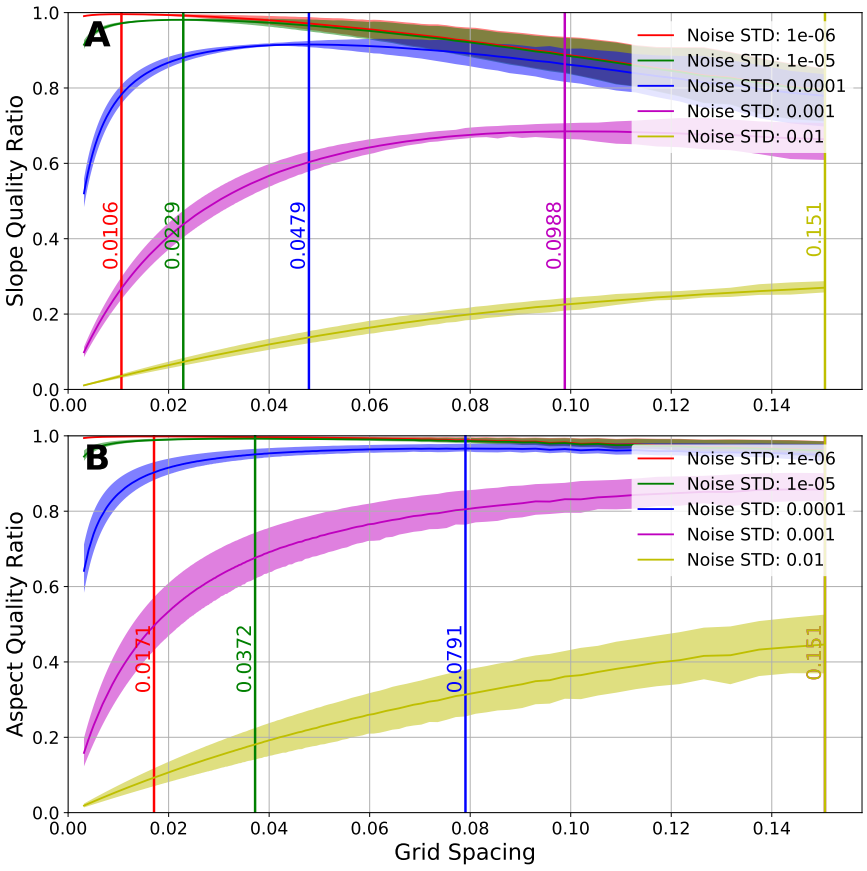

Figure 7: Quality ratio magnitudes based on a set of grid spacings and noise levels, taken from Smith et al. (2019

3 Real World Data – Santa Cruz Island

Now that we have a handle on the theory, let's actually implement this with real world data. We have provided a set of DEMs and DEM uncertainty grids for the Pozo catchment on Santa Cruz Island. They can be found via github in the 'example' subfolder:
https://github.com/UP-RS-ESP/TopoMetricUncertainty

For details on the gridding and uncertainty estimation methodology please refer to the Smith et al. (2019) publication, and the more detailed gridding manual here:
https://github.com/BodoBookhagen/Lidar_PC_interpolation

3.1 Calculating the Slope and Aspect Errors and Uncertainties

We can implement the QR calculation in python by taking the definitions for TE and PEU from github https://github.com/UP-RS-ESP/TopoMetricUncertainty/blob/master/uncertainty.py:

```
def QR(z, spacing, std, mask):
    '''
    Function that takes an elevation, grid spacing, elevation std, and a mask
    to return the Quality Ratio for both slope and aspect calculations
    '''
    trunc_slp = trunc_err_slope(z, spacing)
    trunc_asp = trunc_err_aspect(z, spacing) / 4.
    trunc_slp[mask] = np.nan
    trunc_asp[mask] = np.nan
    peu_slp = peu_slope_field(z, spacing, std)
    peu_asp = peu_aspect_field(z, spacing, std) / 4.
    peu_slp[mask] = np.nan
    peu_asp[mask] = np.nan

    QRslp = (1 / (1 + np.abs(trunc_slp))) *(1 / (1 + np.abs(peu_slp)))
    QRasp = (1 / (1 + np.abs(trunc_asp))) *(1 / (1 + np.abs(peu_asp)))
    QRslp[mask] = np.nan
    QRasp[mask] = np.nan

    return QRslp, QRasp
```

This function takes an elevation dataset z, the grid spacing, the elevation uncertainty *std* and a mask. In our case, we use a mask of 0 to remove water and nodata areas. For a single grid resolution, it would look like this:

```
import gdalnumeric
base_dir =
spacing = 10
dem = base_dir + 'DEM/Pozo_UTM11_NAD83_g_' + str(spacing) + 'm.tif'
uncertainty = base_dir + 'STD/Pozo_UTM11_NAD83_g_' + str(spacing) + 'm_std.tif'

#Load the data as arrays
elev = gdalnumeric.LoadFile(dem).astype(float)
std = gdalnumeric.LoadFile(uncertainty).astype(float)

#Mask out water areas
mask = np.where(elev < 0)
elev[mask] = np.nan
std[mask] = np.nan
```

```
#Calculate the Quality Ratios
QRslp, QRasp = QR(elev, i, std, mask)
QRslp[mask] = np.nan
QRasp[mask] = np.nan

plt.close('all')
f, (ax, ax2) = plt.subplots(1,2)

vmin, vmax = np.nanpercentile(QRslp, [5, 95])
cb = ax.imshow(QRslp, vmin=vmin, vmax=vmax)
cb = colorbar(cb)
cb.formatter.set_powerlimits((-1, 4))
cb.update_ticks()
cb.set_label('Slope QR', fontsize=18)
cb.ax.tick_params(labelsize=16)
cb.ax.yaxis.offsetText.set_fontsize(16)
ax.set_xticks([])
ax.set_yticks([])

vmin, vmax = np.nanpercentile(QRasp, [5, 95])
cb = ax2.imshow(QRasp, vmin=vmin, vmax=vmax, cmap=plt.cm.seismic)
cb = colorbar(cb)
cb.formatter.set_powerlimits((-1, 4))
cb.update_ticks()
cb.set_label('Aspect QR', fontsize=18)
cb.ax.tick_params(labelsize=16)
cb.ax.yaxis.offsetText.set_fontsize(16)
ax2.set_xticks([])
ax2.set_yticks([])
```

Which gives us Figure 8.

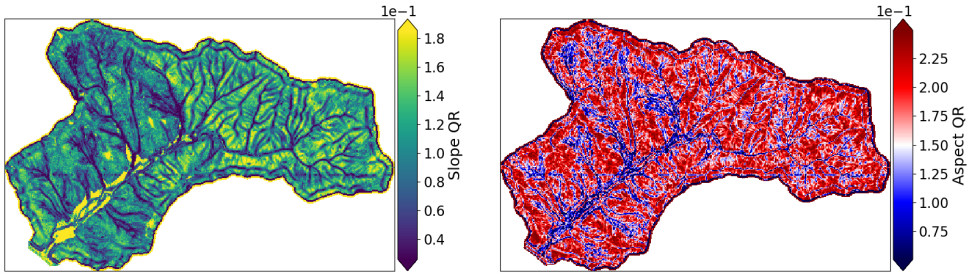

Figure 8: Slope and aspect quality ratios for the Pozo catchment at 10m spatial resolution.

3.2 Choosing a DEM Resolution

We can calculate the QR for any grid resolution – we have provided data from 2m to 30m in 1m steps. Please note that we take the median QR as the overall QR for each grid resolution. As can be seen in Figure 8, the QR is highly spatially variable. Furthermore, the slope and aspect grids behave differently – aspect is better defined on steep slopes, and slope is better defined on flat areas. For example, if you are most interested in maximizing the quality of your DEM in flat areas, it might be better to take the median QR of only low-slope areas.

We can loop through all DEM resolutions and plot the results as a graph:

```
spacings = range(2, 31)[::-1]
qr_s, qr_a = [], []
for i in spacings:
   dem = base_dir + 'DEM/Pozo_UTM11_NAD83_g_' + str(i) + 'm.tif'
   uncertainty = base_dir + 'STD/Pozo_UTM11_NAD83_g_' + str(i) + 'm_std.tif'

   #Load the data as arrays
   elev = gdalnumeric.LoadFile(dem).astype(float)
   std = gdalnumeric.LoadFile(uncertainty).astype(float)

   #Mask out water areas
   mask = np.where(elev < 0)
   elev[mask] = np.nan
   std[mask] = np.nan

   #Calculate the Quality Ratios
   QRslp, QRasp = QR(elev, i, std, mask)
   QRslp[mask] = np.nan
   QRasp[mask] = np.nan

   #Append the QR mean values to a list
   qr_s.append(np.nanmean(QRslp))
   qr_a.append(np.nanmean(QRasp))
   print(i)

#Plot the QR curves
plt.close('all')
f, (ax, ax2) = plt.subplots(2)
ax.plot(spacings, qr_s, 'k-')
ax2.plot(spacings, qr_a, 'k-')
ax2.set_xlabel('Grid Resolution (m)', fontsize=16)
ax.set_ylabel('QR -Slope', fontsize=16)
ax2.set_ylabel('QR -Aspect', fontsize=16)
ax.set_xlim(0, 30)
ax2.set_xlim(0, 30)

#Get the max
qr_s, qr_a = np.array(qr_s), np.array(qr_a)
ideal_space_slp = spacings[np.where(qr_s == np.nanmax(qr_s))[0][0]]
ideal_space_asp = spacings[np.where(qr_a == np.nanmax(qr_a))[0][0]]

#Add the max to the plot
ym, ya = ax.get_ylim()
ax.plot((ideal_space_slp, ideal_space_slp), (ym, ya), 'r--')
ax.text(ideal_space_slp, (ym + ya)/2, str(ideal_space_slp) + 'm', va='top',
    rotation='vertical', fontsize=16, color='r')
ax.set_ylim(ym, ya)
```

```
ym, ya = ax2.get_ylim()
ax2.plot((ideal_space_asp, ideal_space_asp), (ym, ya), 'r--')
ax2.text(ideal_space_asp, (ym + ya)/2, str(ideal_space_slp) + 'm', va='top',
    rotation='vertical', fontsize=16, color='r')
ax2.set_ylim(ym, ya)

plt.tight_layout()
plt.show()
f.savefig(base + 'optimal_res.png', dpi=300)
```

The QR curve for Pozo looks like this:

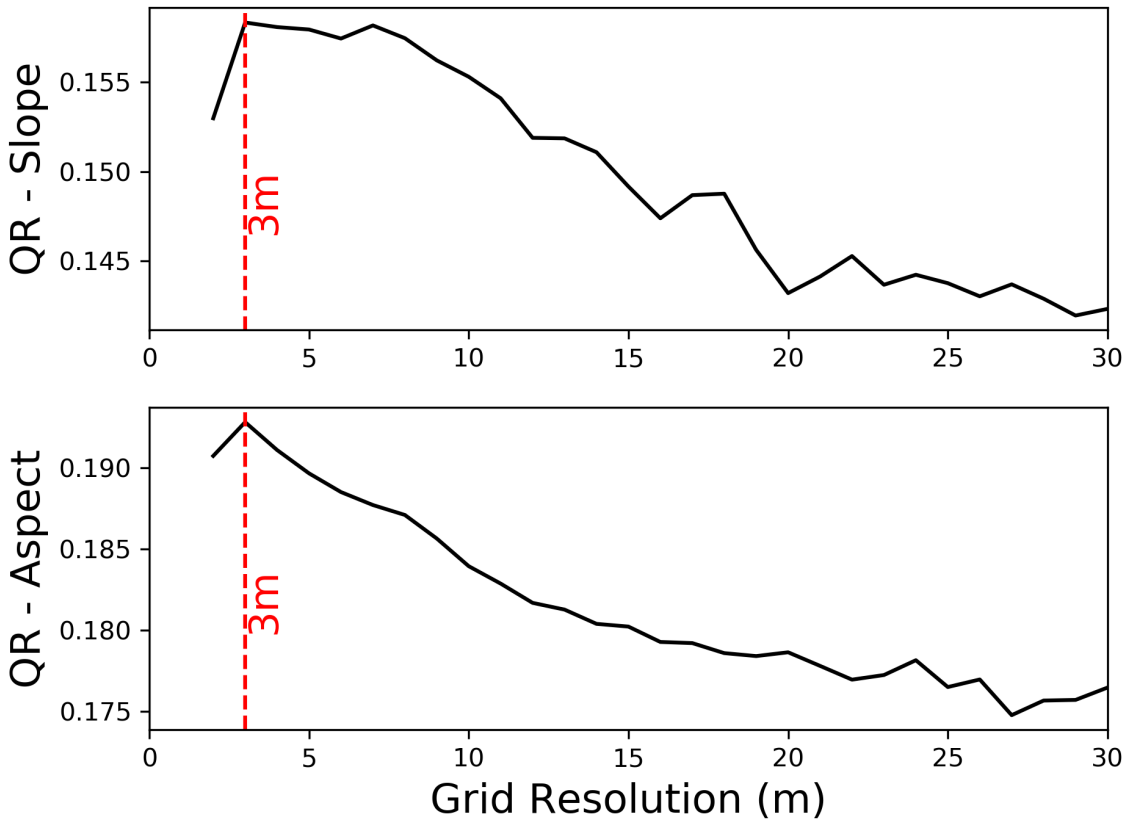

Figure 9: Slope and aspect quality ratios for the Pozo catchment across resolutions from 2m to 30m.

For the entire Pozo catchment, we find an optimal resolution of 3m, given the topography and our calculated uncertainty. **It is important to note that not all lidar datasets are created equal! If a lidar dataset has much less uncertainty (e.g., due to a high point density), the optimal DEM resolution that minimizes the combined error from TE and PEU might be much smaller.**

4 Further Steps – Subsetting the Data

As is clear from Figure 8, not all areas have the same TE, PEU or QR. Our approach minimizes the combined TE and PEU over a defined area. Depending on your application, splitting your study area into smaller pieces may result in better results. For example, when we considered the entire SCI watershed by watershed, we found large differences in optimal resolution (Figures 10-11).

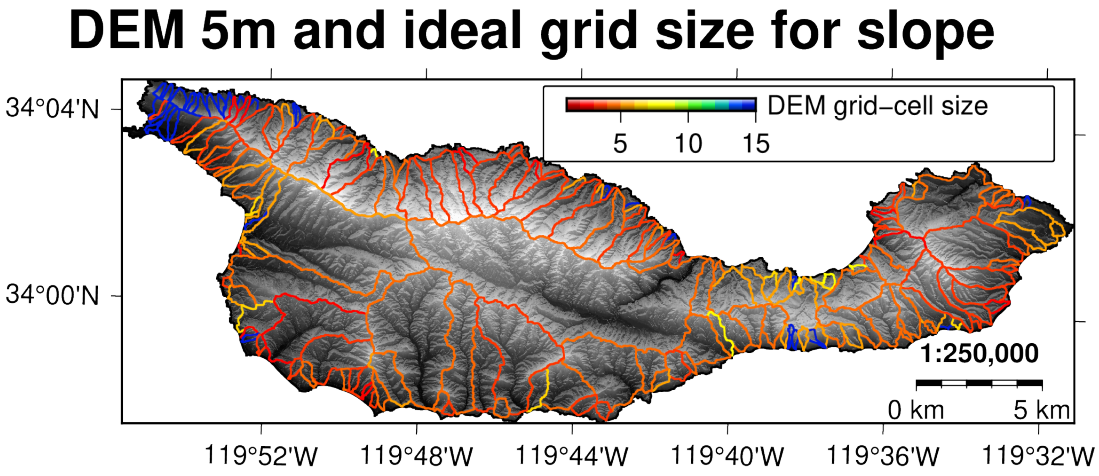

Figure 10: Optimal slope grid resolution across SCI. Taken from Smith et al. (2019).

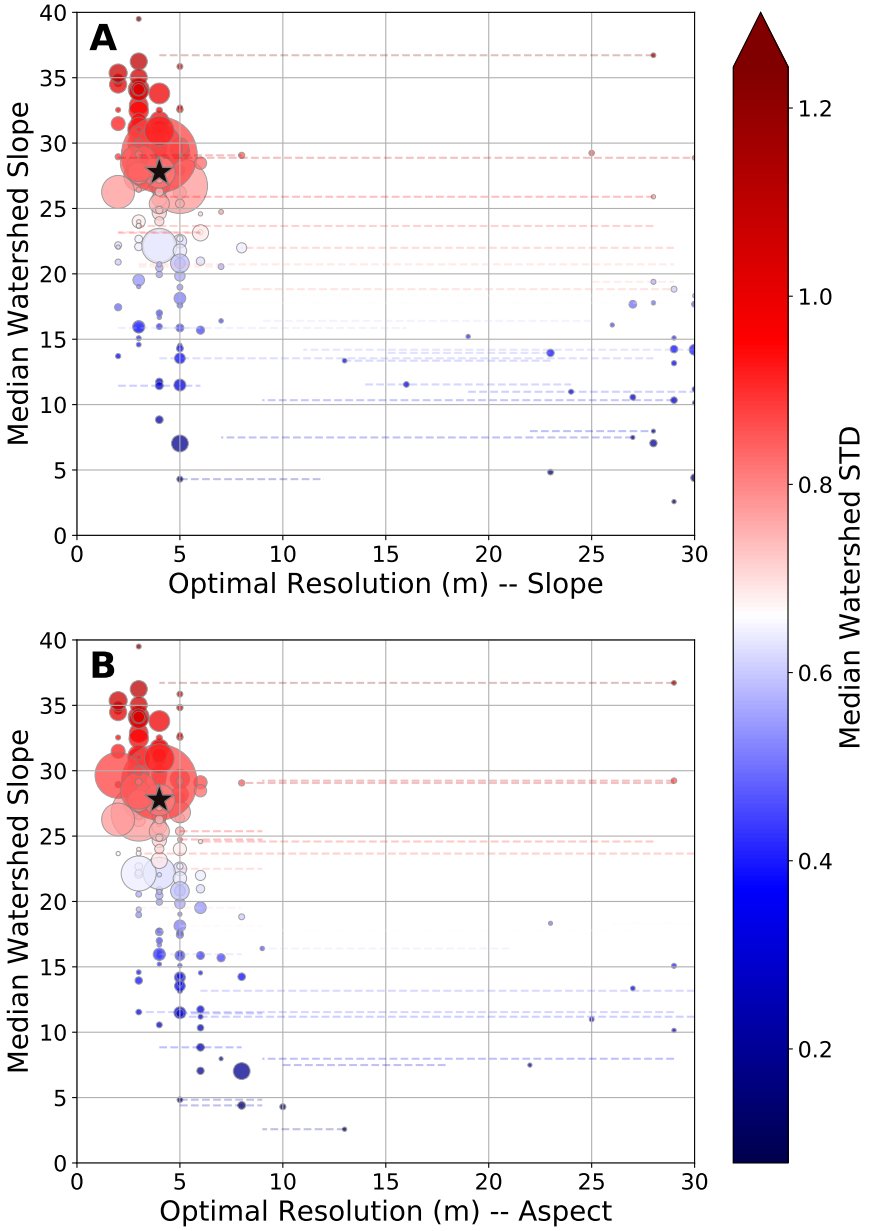

Figure 11: Optimal grid resolution for minimizing error bounds on slope and aspect calculations across SCI compared to catchment median slope. Point sizes are scaled to catchment size, from 0.1 to 34 km^2. Dashed lines show the 25th percentile to 75th percentile of optimal grid resolution, colored by the standard deviation of that point. Stars show the whole-island optimal resolution and median slope. There exists a clear trend where lower median slopes lead to lower optimal grid resolutions. Standard deviations also tend to be smaller in these catchments, due to higher DEM fidelity over flat terrain. Catchments with low (>10 meter) optimal grid resolutions are small and have complex topography. Taken from Smith et al. (2019)