Marcel Rodriguez-Riccelli
2/4/2026
Satellite Image Analysis (GEOG 581)
LAB 03 - TASSELED CAP GENERATION AND TERRAIN CORRECTION

LAB DAY: THURSDAYS
LINK TO CODE: GOOGLE COLLAB, GITHUB

QUESTION 1. What is the result of using first() on the landsat5 ImageCollection?

The *.first()* function selects the first image of our filtered ImageCollection object. In this case, by following the function chain and arguments in the code:

```
langtang_landsat = landsat5
.filterBounds(langtang)
.filterDate('2006-07-01', '2006-09-30'
).sort('CLOUD_COVER').first()
```

we can expect that the first filter will filter the Landsat 5 Top of Atmosphere collection for only images that include our point of interest and the second filter will pass only images captured between July 1 and September 30th, 2006. Then, the *.sort()* function sorts the filtered collection by cloud coverage from lowest to highest, and the final function in the chain *.first()* returns the first image in that collection. The result is that the *langtang_landsat* variable is a type image, and that image is the image with the lowest cloud coverage between our specified dates that includes our point of interest.

QUESTION 2. In your own words, how does calling sort() affect the contents of langtang_landsat?

The *.sort()* function reorganizes the ImageCollection sequentially based on the variable whose key is input as the first argument to the function, while the second argument allows for the Images to be sorted in either ascending or descending order, but if no second argument is passed in the default is ascending order.

QUESTION 3. Is 'CLOUD_COVER' an ImageCollection property, an Image property, a band name, or a pixel value?

CLOUD_COVER is a key in the metadata dictionary associated with an Image type object that has a single value associated with it for each Image. For example, from the image selected by our earlier *.first()* function, we can reference the metadata dictionary to find the *CLOUD_COVER* value for that image by the following code:

```
langtang_landsat_info = langtang_landsat.getInfo()
cloud_cover_info = langtang_landsat_info["properties"]["CLOUD_COVER"]
```

print("Cloud coverage value for langtang-landsat:", cloud_cover_info)

which returns:

Cloud coverage value for langtang-landsat: 5

meaning that the cloud coverage value for our selected image is 5, and proving that CLOUD_COVER is an Image property.

QUESTION 4. Does it matter at what location in the chain of functions (filterBounds(), filterDate(), etc.) that we call sort()? Why or why not?

It matters computationally but not in the end result; for example if we were to .sort() before .filterDate or .filterBounds, the .sort() function would be sorting through a significantly larger Image Collection, but we would ultimately arrive at the same collection sorted in the same way.

QUESTION 5. Describe the image you have -- what land covers do you see, what does the landscape look like, and what are the general atmospheric conditions?

The landscape appears, and the basemap reinforces, that this is an area of extremely drastic relief. There appears to be mixed land cover throughout—including snow, water, healthy, and dry vegetation. This image appears to be quite clear of atmospheric interference; I'm not able to detect any haze or clouds by manually inspecting the image.

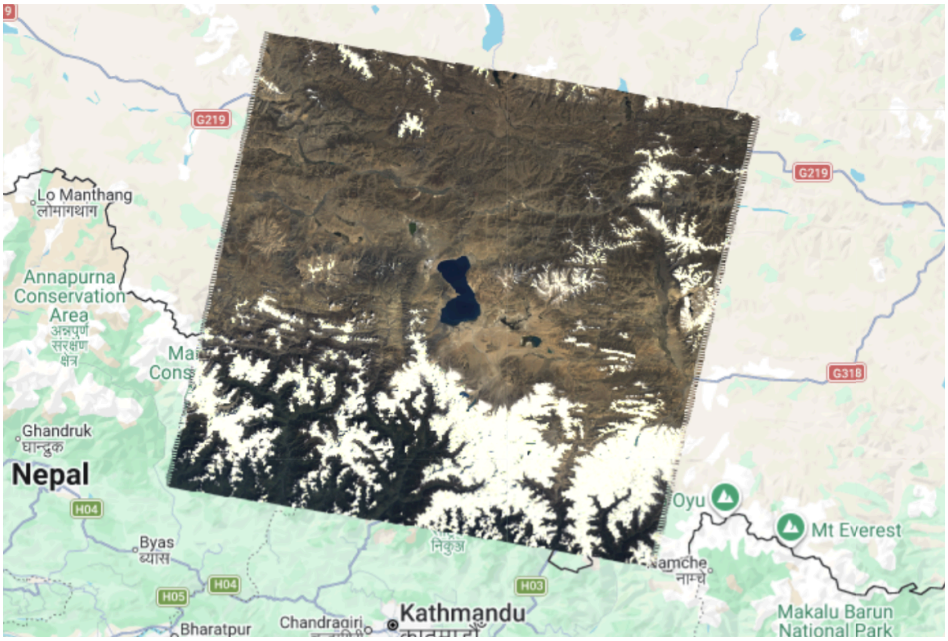

Fig. 1. A true-color image of the "langtang" area of interest on 2006-09-28, selected from the Landsat 5 Top of Atmosphere dataset.

QUESTION 6. In the script above, what is the dimension of tc_coefficients? What do the six columns in each row relate to? What do the six rows of tc_coefficients each correspond to?

The *tc_coefficients* array has a dimensionality of 6 x 6. This array is a linear transform matrix, presumably for Landsat 5 Thematic Mapper Top of Atmosphere dataset. Each column relates to a particular band in the Landsat 5 Top of Atmosphere dataset, excluding Band 6 (Thermal Infrared):

1st Column - Band 1 (Blue)
2nd Column - Band 2 (Green)
3rd Column - Band 3 (Red)
4th Column - Band 4 (NIR)
5th Column - Band 5 (SWIR 1)
7th Column - Band 7 (SWIR 2)

Each row relates to a Tasseled Cap index, that after the transformation, will correspond to one layer in the transformed image:

Row 1 - Brightness
Row 2 - Greenness
Row 3 - Wetness
Row 4 - Fourth
Row 5 - Fifth
Row 6 - Sixth

The first row represents brightness, the second greenness, and the third wetness, while the fourth, fifth, and sixth rows are higher order residual components. Each value in each row corresponds to the weight given to each band when calculating the weighted sum to yield each new layer.

QUESTION 7. Consider the Tasseled Cap transform coefficients in the third row of tc_coefficients (eg, 0.1509, 0.1973, etc.). Why are some bands' coefficients relatively large and positive while others may be very small (close to 0) or very low negative values? When taken together, why are they effective at producing a Wetness image?

The third row in tc_coefficients:

[0.1509, 0.1973, 0.3279, 0.3406, -0.7112, -0.4572]

contains large positive values for Band 3 (Red) and Band 4 (NIR), lower positive values for Band 1 (Blue) and Band 2 (Green), and large negative values for Bands 6 (TIR) and Band 7 (SWIR). Using the values from the dataset, we can test how this may affect a pixel that represents a moist canopy versus one that represents a dry canopy.

Wet Vegetation Pixel

WET VEG: {'B1': 0.08888793736696243, 'B2': 0.08158010989427567, 'B3': 0.06066643446683884, 'B4': 0.3331824243068695, 'B5': 0.1884404420852661, 'B7': 0.07795027643442154}

Dry Vegetation Pixel

DRY VEG: {'B1': 0.17151549458503723, 'B2': 0.22493833303451538, 'B3': 0.2686265110969543, 'B4': 0.3331824243068695, 'B5': 0.4338604211807251, 'B7': 0.4091622531414032}

For each pixel, we can apply our weight coefficients to each band and get the resulting "wetness" by summing the resulting values:

Wet Vegetation Pixel

$$0.01341 + 0.01609 + 0.01989 + 0.11349 - 0.13401 - 0.03563 = \mathbf{-0.00676}$$

Dry Vegetation Pixel

$$0.02589 + 0.04438 + 0.08807 + 0.11349 - 0.30863 - 0.18708 = \mathbf{-0.22388}$$

From this operation, we can see that our dry pixel resulted in a higher wetness value than the land pixel.

We know that generally, the transformation should promote variance between wet versus dry surfaces. We can refer to the spectral signature of wet and dry vegetation to understand why each coefficient was chosen to produce an effective wetness image *(Fig. 2)*—we see that the visible band is weighted gently where dry and wet surfaces have similar reflectance values and exhibit little variance; the red and near infrared are weighted moderately and positively where there is slightly more variance and wet surfaces are more reflective; and the shortwave infrared bands are weighted very strongly and negatively where there is very high variance and dry surfaces are much more reflective.

QUESTION 8. Please add langtang_landsat and langtang_array1D to your map and inspect values at a location of your choice using the Inspector tool. Note the format of these values. How are the two images different or alike? How are the image values within langtang_landsat and langtang_array1D organized? For this question, don't focus on the array values so much as how the array is structured.

By using the reducer to reduce the image to a single pixel for each langtang_landsat and langtang_array1D, and printing out the information, we can see the difference in structure for each:

{'B1': 0.17151549458503723, 'B2': 0.22493833303451538, 'B3': 0.26862651109695435, 'B4': 0.3331824243068695, 'B5': 0.4338604211807251, 'B6': 301.9180603027344, 'B7': 0.4091622531414032, 'QA_PIXEL': 5440, 'QA_RADSAT': 0, 'SAA': 14240, 'SZA': 3700, 'VAA': -9165, 'VZA': 60}
{'array': [0.17151549458503723, 0.22493833303451538, 0.26862651109695435, 0.3331824243068695, 0.4338604211807251, 0.4091622531414032]}

We can see that the original *langtang_landsat* image stores data in a dictionary, where each key is the band and each associated value is the reflectance value in that band for the pixel being inspected, while *langtang_array1D* stores the reflectance value of each band for each pixel as a 1x6 array, with one value for each of the six bands we had selected for.

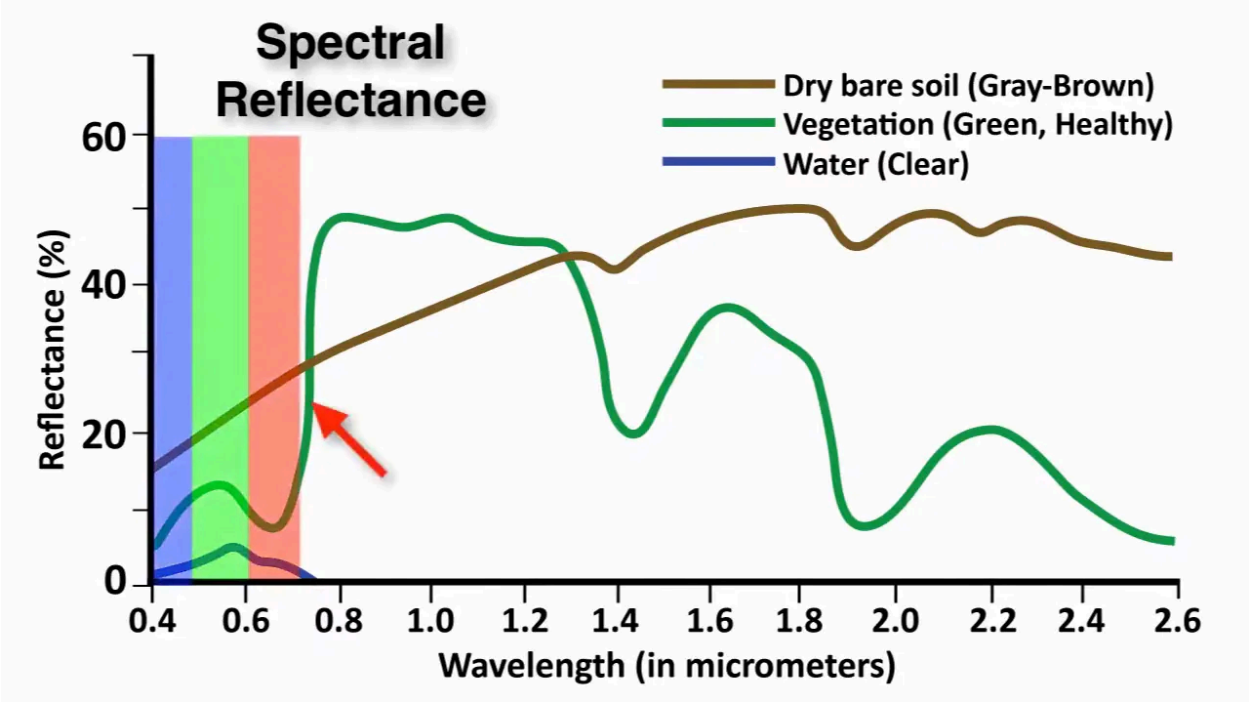

Fig 2. Spectral reflectance curves for dry bare soil (brown), healthy vegetation (green), and water (blue).

QUESTION 9. As above, please add langtang_array2D to your map and inspect values at a location of your choice. How are the three images different or alike? How are the image values within langtang_landsat, langtang_array1D, and langtang_array2D organized?

After adding each to our map, we can see that they're all being visualized differently—the original *langtang_landsat* image shows as a true color image, the

langtang_array1D false color, and *langtang_array2D* greyscale. Looking at the difference in the way each value is organized, we can see that *langtang_landsat* and *langtang_array1D* pixel data is stored similarly to how they were in the previous question, while pixel information for the *langtang_array2D* is stored in the following format:

{'array': [[0.17151549458503723], [0.22493833303451538], [0.26862651109695435], [0.3331824243068695], [0.4338604211807251], [0.4091622531414032]]}

as a nested array, or an array composed of multiple arrays.

QUESTION 10. What does arrayProject([0]) do to the structure of tc_image? (Hint: add two versions of tc_image to your map, one with arrayProject([0]) and one without, and use the Inspector tool.)

Looking at the structure how data is stored for a pixel from each image using the inspector tool, we see:

.arrayProject([0])
[0.3270285083159804, -0.025411500795930627, -0.039170106972754, -0.036204202143102895, 0.05679761876091361, -0.03807716405540705]

No .arrayProject([0])
[[0.3270285083159804], [-0.025411500795930627], [-0.039170106972754], [-0.036204202143102895], [0.05679761876091361], [-0.03807716405540705]]

The *.arrayProject()* function has converted the array for data in each pixel to from a multidimensional to a one dimensional array, by removing the array nesting and changing each member of the array from its own discrete array to a scalar value.

QUESTION 11. In your own words, what does `arrayFlatten()` *do*? (Hint: use the information under the Docs tab.) Why do we not need to declare `tc_image` as a variable in this command?

To see what the *.arrayFlatten()* function does, we can reduce the flattened image to a single pixel and print out its value:

{'Brightness': 0.7251212641149759, 'Greenness': -0.04552636864483353, 'Wetness': -0.22380392573177812, 'fifth': 0.18854941806793213, 'fourth': -0.04957301802635 1945, 'sixth': -0.05573344391286372}

From this, we can see that the data for each pixel is again stored as a dictionary, where the keys are our chosen band names, and the associated values are those that were in the respective column.

QUESTION 12. Add your `langtang_landsat` to the map and visualize it using the standard false-color visualization (near-infrared, red, green). In what way does using these Tasseled Cap values help to distinguish features in the image compared to the standard false-color image?

We can see a much more drastic difference between landcover types in the Tasseled Cap image than in the standard false-color image. It's also more intuitive to relate hues to each land cover type just by viewing the image. For example the snow atop the Himalayan mountain range is bright and wet, so it's got a pink-purpleish hue; a mixture of Brightness which we have encoded in the red color channel, and Wetness which we have encoded in the blue color channel. If I wanted to make similar inferences about land cover type from the false-color image, I would have to reference the spectral signature for different types of surface materials.

QUESTION 13. In your image, what areas or features tend to exhibit the highest Brightness, and how do they measure in terms of Greenness and Wetness? (Using the inspector tab and switching to a single band may help to compare the three bands.)

The snow covered surfaces show the highest Brightness values from ~1 to ~1.5, and also have moderate Wetness values from ~0.4 to ~0.6, but negative Greenness values. The dry surfaces on the northern side of the mountain range, also have high Brightness values, but typically negative Greenness and Wetness values.

QUESTION 14. Which areas or features have the highest and lowest Wetness?

The dry surfaces on the northern side of the mountain range have the lowest Wetness values from approx. -0.3 to 0.0, while the snow covered surfaces atop the mountains have the highest Wetness values from ~1 to ~1.5. The pixels with healthy vegetation on the southern side of the mountain range seem to have moderate Wetness values from ~0.0 to ~0.5.

QUESTION 15. Re-use your script from last week to display a scatter plot between Greenness and Wetness. Run this scatterplot over a 1km radial buffer region surrounding langtang. Please hand in a screen grab or PNG of your scatter plot.

I created a buffered geometry surrounding the langtang point specified earlier, and generated the following scatter plot using Matplotlib using Wetness and Greenness values from the Tasseled Cap transformation image *(Fig. 3)*.

QUESTION 16. What types of features fall within this buffered region? How are they each depicted in the Tasseled Cap visualization using tc_vizParams?

It looks like a mostly snowy region on the northern face of a mountain, about halfway down the mountain descending into a nearby valley. Most of the pixels are pink, a mixture of our chosen symbology for Wetness in the blue channel and Brightness in the red. Some pixels are a darker

shade of pink, perhaps because of lower reflectance caused by the relief shadow from the drastic relief.

QUESTION 17. Describe the shape of this plot. Can you identify a trend(s) between the Greenness and Wetness values? What features most strongly align with this trend(s)?

The upper portion of the plot *(Fig. 3)* is flat, with high wetness and low greenness, suggesting this grouping of points corresponds to a snow-covered area. Moving down the wetness axis from top to bottom, there's a long arc that shows a negative correlation between wetness in greenness, meaning that as greenness increases, wetness decreases, which might suggest that a good part of

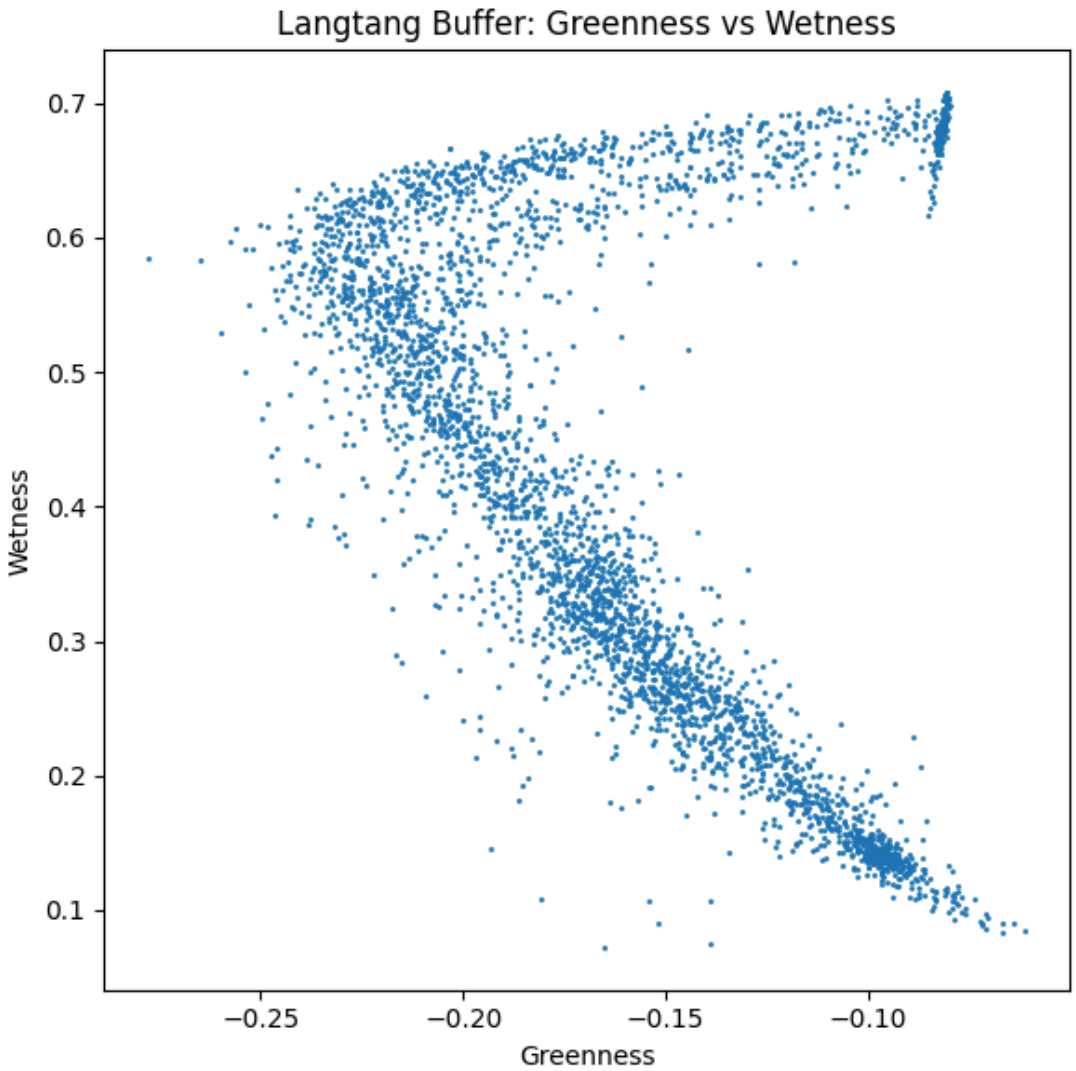

Fig. 3. Scatter plot of Greenness vs Wetness in a 1km zone around our Langtang point, from a Tasseled Cap transform of Landsat 5 imagery.

the buffered zone contains a gradient from the snowy mountain top down into an increasingly vegetated valley. The cluster in the bottom right most corner of the plot, shows that a large number of pixels are of low wetness and greenness approaching zero, which perhaps suggests a concentration of dry vegetated surfaces.

PART 2

QUESTION 18. In your own words, please describe the solar zenith angle and solar azimuth angle.

The solar zenith and azimuth describe the positionality of the sun relative to a location on Earth. The zenith is the angle of the sun measured from vertical, while the azimuth is the angle from North.

QUESTION 19. In your own words, how does the toRadians() function convert an input image's degree values to radian values? How does the toFloat() function affect the data type of the output image?

The *toRadians()* function converts an input image's degree values to radians values by multiplying by the degrees-to-radian conversion ratio given by PI / 180, and outputs a floating point number. This ratio makes sense because radians *(θ)* are directly proportional to arc length *(s)* for a given radius *(r)* expressed by $s=r\theta$, where the angle for half a circle is PI radians, as opposed to 180 degrees. The *toFloat()* function makes sure that the output is a type floating point number as opposed to an integer.

QUESTION 20. In the toRadians() function, what does img represent? (Please see https://developers.google.com/earth-engine/tutorial_js_03 for information on writing/calling functions.) Why do we need to use this function in this script in the first place?

For the *toRadians()* function, *img* is an input argument of arbitrary type. However, in order apply the functions inside of the *toRadians()* function, it is implied that the input argument will be an Earth Engine Image, or the function will not succeed. We can see that when called, the inputs to *toRadians()* are *ee.Image* type objects generated by the with constant azimuth and solar values for each pixel. This is useful for iterating the multiplication, division, and float conversion over every pixel of the entire input image.

QUESTION 21. Visualize the entire image in your map and set the respective ranges to 100%. (This ensures consistency between answers.) Is there any differentiation between the mapped appearance of the input `langtang_landsat` and output `langtang_corrected` images? In answering this question, make sure to use a 100% stretch over the entire image area. How could you check to see whether your output image has the correct values?

We can see that pixels that are dark near what appears to be the most drastic areas of relief in *langtang_landsat* have been brightened by the *langtang_corrected.* In my image of *langtang_corrected*, in the southern mountainous portion, it seems that many of the pixels have been overcorrected, causing whiteness on what would be the shaded side of the most drastic relief, and yellowing on the snow covered surfaces. This is possibly due to the drastic nature of the relief in general, leading me to believe that correction might be more effective for areas that have more subtle relief.

QUESTION 22. When mapped, do the IC and shade images correlate to each other? In what way, and why?

Quickly referencing ArcGIS's documentation, the hillshade formula appears to be exactly the same as the illumination condition formula, which suggests the illumination condition and hillshade are the same. This would stand to reason, as dividing the illumination condition from the observed image would correct reflectance differences caused by relief including shadowing from the sun's rays. However, these two images are clearly different; the IC and shade images appear to have high and low pixel values on opposite sides of the relief.

This suggests that the solar geometry used to calculate the hillshade was different from the actual solar geometry from the Landsat 5 dataset. By printing the "AZ" value, we can see our solar azimuth angle is 142.43251582 degrees, meaning that the sun was to the south east, and that appears to be the case when viewing the image. The hillshade however, shows sunlight approaching from the northeast, meaning that the solar azimuth angle used was probably closer to ~300. The illumination condition image also shows lower values where sunlight is blocked by the peaks, meaning the solar zenith in the hillshade computation was likely higher than the actual solar zenith from the Landsat 5 dataset.

QUESTION 23. When using trigonometric functions of sin() and cos(), why don't we have an angle value within the parentheses of the function call?

We're passing in an image to the trigonometric functions of *sin()* and *cos()*, which has constant values throughout that correspond to the zenith and azimuth angles. We need to match the dimensionality to perform the mathematical operations .mutiply(), .add(), and .subtract(), and to be performed on an image object they require the input to be an image object. This way, we can do pixel-wise math.

QUESTION 24. In your own words, step through the calculation for `IC` and describe the calculation components and how they are connected to each other in the given order of operations.

To understand the calculation for IC, we should reference the formula:

$$IC = cos(z) * cos(s) + sin(z) * sin(s) * cos(a-o)$$

Where *z* is the solar zenith angle, *s* is the topographic slope angle, *a* is the solar azimuth angle, and *o* is the topographic aspect angle. With this knowledge, the calculation for IC can be broken down as follows:

AZ_R.subtract(ASP).cos()

The topographic aspect angle image is subtracted by the image of constant solar azimuth angles in radians, and the cosine of the resulting values is obtained.

.multiply(SLP.sin())

The output of the previous step is multiplied by the sine of the topographic slope angle.

.multiply(ZE_R.sin())

The output of the previous step is multiplied by the sine of the solar zenith angle.

.add(ZE_R.cos().multiply(SLP.cos()))

The output of the previous step is added to the cosine of the solar zenith angle multiplied by the topographic slope. It should be noted that the solar zenith angle and topographic slope are multiplied before being added to the cosine of the solar zenith angle.

QUESTION 25. Why do we need to use a chain of functions (subtract(), add(), multiply(), etc.) rather than writing out the calculation with -, +, and * characters?

Chaining these functions is required for performing per-pixel operations on Google Earth Engine Image type objects.

QUESTION 26. Please interpret the langtang_corrected output. When mapped, compare how the langtang_corrected and shade images visually correlate to each other? In what way, and why?

The hillshade shows low values on the southeast side of the relief from the *langtang_corrected* output. This is the opposite side of the relief that has been corrected for by the illumination condition, where low values in illumination condition (shaded areas) divided by the observed image, should result in brightened pixels in the corrected image.

QUESTION 27. How do values of darker, shaded regions in the input `langtang_landsat` image change in this output `langtang_corrected` image? What about brighter regions in the input image?

The values of darker shaded regions in the original *langtang_landsat* image have been corrected for by the illumination condition, where a pixel with a low value (because it's shaded),

when divided by a small value between 0.0 and 1.0 in the illumination condition (because it's where shade is calculated to be given the solar and relief geometry), it will result in a high value. Regions outside of the calculated shadow, will be divided by a value near 1, so will be relatively unaffected between the *langtang_corrected* and *langtang_landsat* images.

QUESTION 28. In your opinion, what are the visually evident benefits of applying this Cosine Correction to the image? What regions or features in the output image illustrate some of this correction method's limitations?

The benefits are certainly visible where the relief is subtle, such as in the northern portions of the Langtang image where there are hills, and more limited and prone to overcorrection and discoloration in the southern portion of the image.

QUESTION 29. Include the two "Image IDs" to the corrected and uncorrected image assets you exported to your Earth Engine Assets.

Uncorrected ID:

users/riccellm/langtang_uncorrected

Uncorrected Link:

https://code.earthengine.google.com/?asset=users/riccellm/langtang_uncorrected

Corrected ID:

users/riccellm/langtang_corrected

Corrected Link:

https://code.earthengine.google.com/?asset=users/riccellm/langtang_corrected