Lab 1 - Fundamentals of Google Earth Engine (GEE) API

Introduction

Google Earth Engine (GEE) is a cloud-based computing platform hosted by Google. GEE provides direct access to a multi-petabyte catalog of satellite imagery and geospatial datasets, including the entire EROS (USGS/NASA) Landsat catalog, MODIS, NAIP, and Sentinel-1 imagery, and precipitation, elevation, sea surface temperature, and CHIRPS climate data. However, beyond simply being an image archive, GEE also provides APIs for JavaScript and Python to enable researchers to perform planetary-scale analysis of the Earth's surface. GEE is currently free for research, education and nonprofit use. The main components of GEE are: (1) a petabyte-scale archive of publicly available remotely sensed imagery and other data, (2) a computational infrastructure optimized for parallel processing of geospatial data, 3) application programming interfaces (APIs) for JavaScript and Python for making requests to the Earth Engine servers, and (4) an online Integrated Development Environment (IDE) for rapid prototyping and visualization of complex spatial analyses using the Javascript API.

Pre-lab requirement: i.e. you must complete this before lab starts

- Create (or verify) an email account associated with Google. e.g. abcdef@gmail.com.
- Register for a GEE account at https://earthengine.google.com/signup/.

Objectives

This lab focusses on introducing the fundamentals needed to use the GEE API. This lab introduces fundamental terms in GEE and provides guidance through several basic tasks. At the end of this lab you will be able to use GEE to perform the following tasks:

- Run basic Java commands.
- Display and clip image and vector.
- Composite and mosaic images.
- Explore image collections and their metadata.
- Filter image collections.
- Perform simple image band calculations.
- Explore and construct functions and map these functions over an image collection.
- Import and export raster and vector.
- Construct simple graphs based on a set of images e.g. change in vegetation index over time.

Getting started

The GEE Interface

Having created an account with GEE, the Code Editor is accessed at https://code.earthengine.google.com/. The code editor is a web-based IDE for the Earth Engine JavaScript API. As shown in **Error! Reference source not found.**,the code editor window has f our panels: (1) the Editor panel is where JavaScript code is written; (2) the right panel has tabs for printing (Console), querying map results (Inspector), and managing long running processes (Tasks); (3) the left panel has tabs for organizing programming scripts (Scripts), accessing documentation (Docs), and managing uploaded datasets (Assets); and (4) the interactive map window is used for visualizing map output. The code editor also has a search bar for finding datasets and places of interest, and a help menu that links to a user guide, help forum and a variety of other means of support.

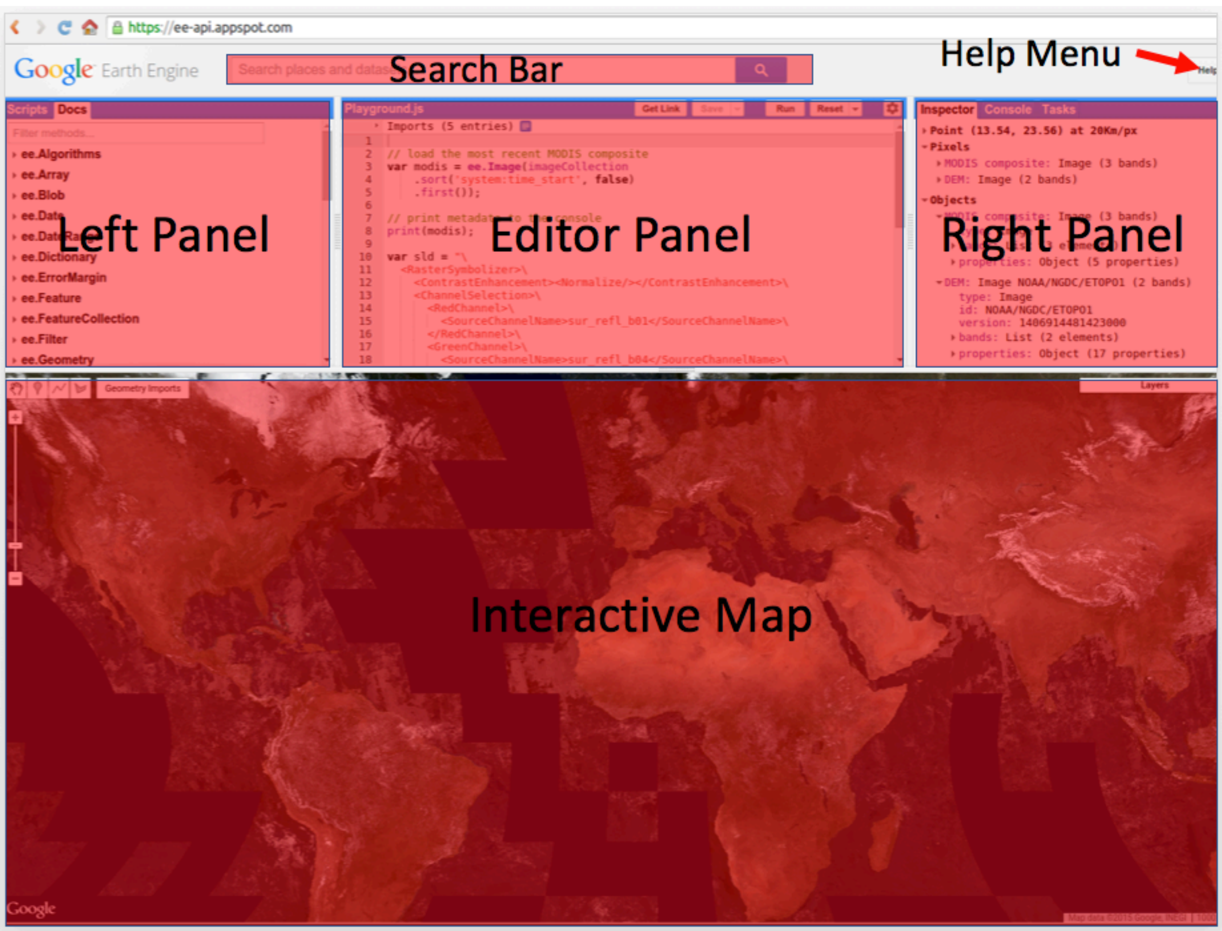

Figure 1. Image of Code Editor highlighting components of layout.

The interactive map window includes many familiar functions e.g. zoom control (on the left hand side of the window) and the ability to swap between map and satellite views for the background (on the right hand side of the window). There are also buttons (on the left) to pan, or draw point, line or area objects ( ). Items you draw in the map window will be automatically stored as layers in the Geometry Imports. You can interactively explore data values by first clicking on the Inspector tab in the right panel and then click on the interactive map window. This will show point coordinates, zoom level, map scaling, and values for any loaded datasets.

Programming in JavaScript

JavaScript is a programming language used for making web content. GEE builds upon JavaScript with modifications for use in the API interface. JavaScript is typed into the Editor panel in GEE. Some helpful tips when using this language:

- Every command line ends with a semi-colon (;)
- Use the keyword **var** to create a variable e.g. var my_grade = 100;
- Parentheses are used to pass parameters to functions. e.g.
  print('This string will print in the Console tab.');
- String objects (i.e. text) start and end with a single quote e.g. var opinion = 'GEE is awesome';
- Square brackets are used to specify items in a list.
  var my_list = ['eggplant', 'apple', 'wheat'];
- The zero index is the first item in the list e.g.
  print(my_list[0]); would print *eggplant* in the Console tab
- Square brackets can be used to access dictionary items by key e.g.
  print(my_dict['color']);
- Or you can use the dot notation to get the same result.
  print(my_dict.color);
- Curly brackets (or braces) are used to define dictionaries (key:value pairs).
  var my_dict = {'food':'bread', 'color':'red', 'number':42};
- Use comments to make your code readable
- Comments start with // and any text in a line after the // is a comment
- Multi-line comments can start with /* and end with */

For mathematical operations, you can use the built-in mathematical functions or you can use the mathematical symbols directly:

print(3.subtract(2)); would print *1* in the Console tab (Approach 1)
print(3-2); would also print *1* in in the Console tab (Approach 2)

You can add text descriptions to make items printed in the console clearer. The example below would print *Subtracting two from three equals 1* in the Console tab:

```
print('Subtracting two from three equals ',3-2);
```

Functions that combine commonly reused steps improve efficiency and make code easier to read. The *my_hello_function* below takes input from the user, adds some text and prints the result to the Console tab. The *add_function* takes an input value, adds 3, and then returns the results with an appropriate text description.

```
var my_hello_function = function(string) { return 'Hello ' + string + '!'; }
print(my_hello_function('world'));  //would print Hello world! in the Console tab

var add_function = function(a) { return a+3; };
var a = 1;
print(a + ' plus 3 is ' + add_function(a));  //would print 1 plus 3 is 4 in the Console tab
```

Performing simple image visualization and analysis in GEE

Displaying and clipping images and vectors

This section provides JavaScript code to perform some simple functions in GEE. Copy each sample into the code window in the Editor panel and then hit the Run button to execute the code. You should periodically save your code by creating a folder for this lab in your GEE scripts folder.

In GEE, ee.image is an object used to represent an Earth Engine image. In the script below, this is used to point to a particular Landsat 5 image that is stored in the variable called myimage.

```
// Create a variable that points to a particular Landsat image in the Earth Engine collection.
var myimage = ee.Image('LANDSAT/LT05/C01/T1_TOA/LT05_015030_20100531');
```

The Map.setCenter command sets the center of Interactive Map to a specified coordinates and controls the zoom level. A zoom level of 1in GEE would show the entire Earth, with larger numbers zooming in e.g. 5: Continent, 10: City, 15: Streets, 20: Buildings.

```
// Center the map and display the image.
Map.setCenter(-76.147, 43.046, 10); // Center on City of Syracuse using level 10 scale
Map.addLayer(myimage);
```

Unfortunately, if you load the image uses the defaults (as shown in the scripts above) the image is not easy to visualize. However, once an image is displayed, a Layers button appears in the map window (upper right). Clicking on this button will display a list of the layers available (in this case just Layer 1) with a check box to turn the display of a particular layer on or off, a sliding bar to change image transparency and a Gear icon ( ) to change display options. Click on the gear icon to change the band combination and display this image as a typical color infrared composite using bands B4, B3, B2 (RGB).

While using the layer controls in the map window is useful, it is often more convenient to establish visualization parameters up front. Type Map.addLayer on a new line in the code editor and hit Ctrl-Space. This keyboard shortcut displays the parameters that can be used within a function. In this case the options are:

```
map.addLayer(eeObject, visParams, name, shown, opacity);
```

In the simple example used initially, the only parameter specified was ee.Object, i.e. the image to display, which we had defined using the variable called myimage. In the example below we specify (in order) eeObject, visParams, and name.

```
//Display the image using a specific band combination and call the image ColorIR composite.
Map.addLayer(myimage, {bands:['B4', 'B3', 'B2']}, 'ColorIR composite');
```

In the example above, the parameters must be specified in the order listed when you hit Ctrl-Space, i.e. you can't skip the visualization parameters if you want to provide a name. However, you can use the dictionary format (with curly brackets) if you can want to define only specific parameters in Map.addLayer. The script below will give the same result as the script above.

```
Map.addLayer({visParams: {bands:['B4', 'B3', 'B2']}, eeObject: myimage, name: 'ColorIR compo
site2'});
```

If you do not comment out the previous Map.addLayer line, you will now see two layers are displayed in the map window, with the color infrared composite on top. In addition to band selection, there are many other visualization parameters that can be specified (see Table 1 in the appendix). While specifying these parameters can be done inline (as shown above), you can also create a variable to hold all the parameters:

```
// Define visualization parameters to display the image in the map window.
var vizParams = {
  bands: ['B4', 'B3', 'B2'],
  min: 0,
  max: 0.5,
  gamma: [0.95, 1.1, 1]
};

Map.addLayer(image, vizParams, 'Color IR composite');
```

You can also display portions of an image, e.g. to limit the display to a particular region of interest (ROI).

```
// Create a circle by drawing a 2000 meter buffer around a point and saving this to variable roi.
var roi = ee.Geometry.Point([-76.147, 43.046]).buffer(20000);

// Display the 2000meter buffer.
Map.addLayer(roi);

// Display a clipped version of the image.
Map.addLayer(myimage.clip(roi));
```

Exploring image collections and their metadata

An ImageCollection is a stack or time series of images. In addition to loading an ImageCollection using an Earth Engine collection ID, Earth Engine has methods to create image collections. The functions ee.ImageCollection() and ee.ImageCollection.fromImages() create image collections from lists of images. You can also create new image collections by merging existing collections.

```
// Specify a location and date range of interest
var point = ee.Geometry.Point(-76.147, 43.046); // Create a point in the City of Syracuse
var start = ee.Date('2014-06-01'); //Define a start date for filter
var end = ee.Date('2014-10-01'); //Define a end date for filter

// Filtering and Sorting an ImageCollection
var filteredCollection = ee.ImageCollection('LANDSAT/LC8_L1T') //import all Landsat 8 scenes
  .filterBounds(point) //filter all scenes using point geometry from above (i.e. limit to Syracuse)
  .filterDate(start, end) //filter all scenes using the dates defined above
  .sort('CLOUD_COVER', true); //sort all images within the ImageCollection by cloud cover
print(filteredCollection);
```

When the print() function is applied to an image, metadata about the image appears in the Console.

```
var first = filteredCollection.first(); //select the first image in the filtered image collection
print(first); //Based on the sort above, the first image here has the lowest cloud cover
```

Image collections can also be limited e.g. by path and row or characteristics that are stored in the metadata.

```
// Load a Landsat 8 ImageCollection for a single path-row.
var collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
    .filter(ee.Filter.eq('WRS_PATH', 15)) //Limit images to those in the path/row over Syracuse
    .filter(ee.Filter.eq('WRS_ROW', 30))
    .filterDate('2014-01-01', '2015-01-01'); //filter by a start and end date of interest

print('Collection: ', collection);
// Convert the collection to a list and get the number of images.
var size = collection.toList(100).length();
print('Number of images: ', size);

// Get the number of images.
var count = collection.size();
print('Count: ', count);

// Get the date range of images in the collection.
var dates = ee.List(collection.get('date_range'));
var dateRange = ee.DateRange(dates.get(0), dates.get(1));
print('Date range: ', dateRange);
```

```
// Get statistics for a property of the images in the collection.
var sunStats = collection.aggregate_stats('SUN_ELEVATION');
print('Sun elevation statistics: ', sunStats);

// Sort by a cloud cover property, get the least cloudy image.
var image = ee.Image(collection.sort('CLOUD_COVER').first());
print('Least cloudy image: ', image);

// Limit the collection to the 10 most recent images.
var recent = collection.sort('system:time_start', false).limit(10);
print('Recent images: ', recent);
```

Performing image band calculations

It is a common requirement in remote sensing to want to manipulate image bands, for example to calculate multi-band indices such as the Normalized Difference Vegetation Index. GEE provides multiple tools to support such analysis such as the normalizedDifference function.

```
/* Create a function to compute NDVI from Landsat 5 imagery where B4 is the NIR band and B3
 is the red band. */
var getNDVI = function(image) {
  return image.normalizedDifference(['B4', 'B3']);
};

// Load two Landsat 5 images, 20 years apart.
var image1 = ee.Image('LANDSAT/LT05/C01/T1_TOA/LT05_015030_19880619');
var image2 = ee.Image('LANDSAT/LT05/C01/T1_TOA/LT05_015030_20100531');

// Compute NDVI from the scenes.
var ndvi1 = getNDVI(image1);
var ndvi2 = getNDVI(image2);

// Compute the difference in NDVI.
var ndviDifference = ndvi2.subtract(ndvi1);

Map.addLayer(ndviDifference);
```

Export Images

While a significant amount of processing can be performed in GEE, there may be a need to export data.

```
// Create an image variable (named Landsat) and select three bands.
var landsat = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_123032_20140515')
  .select(['B4', 'B3', 'B2']); //only select band 4, 3 and 2 for import

// Create a geometry representing an export region.
var geometry = ee.Geometry.Rectangle([116.2621, 39.8412, 116.4849, 40.01236]);

// Export the image, specifying scale in meters and region.
```

```
Export.image.toDrive({
  image: landsat, //set the name of export image to be landsat
  description: 'imageToDriveExample', //set the export image task to be imageToDriveExample
  scale: 30, //define scale to 30 meters
  region: geometry //set the region of export to predefined geometry
});
```

Import Images

In addition to using the imagery in GEE, you can also upload image files (up to 10 GB each) to your Earth Engine user folder. To upload a GeoTiff (other raster formats are not currently supported), click the “NEW” button under the “Assets” tab in the left panel and then select “Image upload”. Earth Engine presents an upload dialog. Click the SELECT button and navigate to a GeoTiff on your local file system.

Give the image an appropriate asset ID (which doesn't already exist) in your user folder. If you want to upload the image into an existing folder or collection, prefix the asset ID with the folder or collection ID, for example /users/name/folder-or-collection-id/new-asset.

Click OK to start the upload.

Once you have started the upload, an “Asset ingestion” task appears on the Tasks tab in the right panel. Hovering over the task in the task manager shows a ? icon that is used to check the upload status. To cancel an upload, click on the spinning gear icon next to the task. Once the ingestion is complete, the asset will appear in your “Assets” tab in the left panel with an image icon. You can click on the image icon to see a preview of your uploaded image or import it into your current script.

Assignment – Answer the Following Questions

Submit your code and text answers for this assignment by clicking on "Get Link" in the Code Editor and sending the link generated to the TA (your link will look something like **https://code.earthengine.google.com/XXXXXXXX**. Any written responses should be in comments with each question separated by a line of forward slashes. For example:

```
//Q1. Text answer to Q1
Code answer to Q1  //All code responses need suitable comments.

////////////////////////////////////////////////////////////
//Q2. Text answer to Q2
Code to Q2
...
```

Basic Java

1. Use the Console to compute the value of 1 + 1. Print your result with "The answer of 1 + 1 is ...". You can use either of the approaches to mathematical operations mentioned in this lab earlier. **Hint:** If you cannot find a function, look in the Docs tab on the left hand panel.

2. What is the result of 1 + 1 * 2 / 3 - 4? Print your result with "The answer of 1 + 1 * 2 / 3 - 4 is ..." in console.

3. Construct a function that computes: (1) the square root of a number, and (2) the square a of a given value (x), name this function as sqpw. The function should return " The square root of x is ... and x to the power of 2 is ..." in the console.

Image Processing

Scenario: Your boss is interested in getting information on spatial and temporal patterns of vegetation in Onondaga County. You are tasked with investigating the possibility of getting this information through remote sensing means. The County has provided a boundary file and all project products should be clipped to the county boundary. You must deliver the following items:

1) **A collection of all cloud-free satellite images**, a list of when the images were taken and their image IDs.
2) **Maximum and minimum returns** for each band of the cloud-free image collection.
3) **NDVI and EVI image collections** based on the cloud-free image collection.
4) **A cloud-free image** of Onondaga County from any time of your choosing. Displayed as both natural color and color infrared.
5) **Timeseries of NDVI and EVI statistics** (Min, Max, Median) within the entire county.
6) **A copy of the code to generate the infomraiton in GEE**.

Now you have been tasked to deliver the above items. You can begin work using the following suggested procedure.

1. Import the county boundary as a vector layer and display it. Display the study area on the map as an appropriately named layer.
2. Filter an image collection (Landsat or Sentinal) of your choice based on several criteria: (1) all filtered images should be within the county boundary, (2) all filtered images should have 0 cloud cover. Show the filtered images on your map as an appropriately named layer.
3. Create images from the filtered ImageCollection (step 2) for both minimum and maximum value functions. Display the minimum and maximum image composites as two named layers.
4. Print a list of the dates when the filtered images were taken in the Console.
5. Print a list of the IDs of filtered images in the Console.
6. Select one image (from step 5) and clip it to the county boundary (from step1). Display the clipped image with the following two band combinations as appropriately named layers:
   a) Natural Color
   b) Color Infrared
7. Perform an NDVI and EVI calculation on the selected image from step 6. Display the NDVI and EVI images as appropriately named layers.
8. Construct a function to compute NDVI or EVI, and then map it over the filtered image collection from step 2. Print the resulting ImageCollection (from mapped function) to the Console.
9. Reduce the NDVI or EVI ImageCollection from step 8 to statistical values (median + maximum + minimum) per image over the entire county boundary. Display a chart of reduced median + maximum + minimum NDVI or EVI over time in the Console.
10. Write code to export: (1) the NDVI or EVI image from step 7 with 30-meter cell size, and (2) the Onondaga County boundary as a kml file. Note: you do not need to submit the export files, just the export code.

Appendix A - Glossary of Terms

- API: Application programming interface. This is similar to ArcMap except the software is in the cloud. Users do not need to install the actual program to use the application. API enables GEE commends from users to be processed remotely on a Google's large server farms.

- Code Editor (**Error! Reference source not found.**): A web-based Integrated Development E nvironment (IDE) for the Earth Engine JavaScript API.

- Image: Raster data are represented as Image objects in Earth Engine. Images are composed of one or more bands and each band has its own name, data type, scale, and projection. Each image has metadata stored as a set of properties. In addition to loading images from the archive by an image ID, you can also create images from constants, lists or other suitable Earth Engine objects.

- Image Visualization Parameters:

Table 1. Visualization parameters for Map.addLayer()

Parameter	Description	Type
bands	Comma-delimited list of three band names to be mapped to RGB	list
min	Value(s) to map to 0	number or list of three numbers, one for each band
max	Value(s) to map to 255	number or list of three numbers, one for each band
gain	Value(s) by which to multiply each pixel value	number or list of three numbers, one for each band
bias	Value(s) to add to each DN	number or list of three numbers, one for each band
gamma	Gamma correction factor(s)	number or list of three numbers, one for each band
palette	List of CSS-style color strings (single band images only)	comma-separated list of hex strings
opacity	Layer opacity from 0 (fully transparent) to 1 (fully opaque)	number
format	Either “jpg” or “png”	string

- ImageCollection: An ImageCollection is a stack or time series of images. In addition to loading an ImageCollection using an Earth Engine collection ID, Earth Engine has methods to create image collections. The constructor ee.ImageCollection() or the convenience method ee.ImageCollection.fromImages() create image collections from lists of images. You can also create new image collections by merging existing collections.

- Image Computation: Computations based on pixel values of images.

- Spatial Reducer: Functions that composite all the images in an Image Collection to a single image representing, for example, the min, max, mean or standard deviation of the images.
- ImageCollection Filter: A series of functions which can be applied to a ImageCollection to find the appropriate image(s) of interest. Specifically, many common use cases are handled by imageCollection.filterDate(), and imageCollection.filterBounds(). For general purpose filtering, use imageCollection.filter() with an ee.Filter as an argument.
- Compositing and Mosaicking: In general, compositing refers to the process combining spatially overlapping images into a single image based on an aggregation function. Mosaicking refers to the process of spatially assembling image datasets to produce a spatially continuous image. In Earth Engine, these terms are used interchangeably, though both compositing and mosaicking are supported.
- Vegetation Index: Nearly all satellite Vegetation Indices employ this difference formula to quantify the density of plant growth on the Earth — near-infrared radiation minus visible radiation divided by near-infrared radiation plus visible radiation. The result of this formula is called the Normalized Difference Vegetation Index (NDVI).
- Map a function: This is usually done by repeating the same function over all images in an ImageCollection. Functions can be directly mapped over a collection using collection_name.map(function_name).

Appendix B – Useful Resources

- Google Earth Engine Java Style Guide
  http://google.github.io/styleguide/javascriptguide.xml
- Google Earth Engine Guides/Cookbook/Dictionary
  https://developers.google.com/earth-engine/
- Google Earth Engine API Tutorials
  https://developers.google.com/earth-engine/tutorials
- Google Earth Engine Workshop (Beginning)
  https://docs.google.com/document/d/1ZxRKMie8dfTvBmUNOO0TFMkd7ELGWf3WjX0JvESZdOE
- Google Earth Engine Workshop (Intermediate)
  https://docs.google.com/document/d/1keJGLN-j5H5B-kQXdwy0ryx6E8j2D9KZVEUD-v9evys

This work was produced by Ge (Jeff) Pu and Dr. Lindi Quackenbush at State University of New York-College of Environmental Science and Forestry. Suggestions, questions and comments are all welcome and can be directed to Jeff at **gpu100@syr.edu**.