Code Tutorial

Central America Disasters | NASA DEVELOP | Fall 2021

Using Earth Observations to Map Flooding for Disaster Monitoring, to Inform Potential Risk, and Prepare for Possible Response

Caroline Williams, Lauren Carey, Maria De Los Santos, Deanna Fanelli, Payton Ireland

Table of Contents

I. Overview

This tutorial was produced by the Central America Disasters Team in the NASA DEVELOP National Program Fall 2021 term. Our project focused on flooding in Central America from Hurricanes Eta and Iota in November 2020. The goal of this tutorial is to give an overview of HYDRAFloods and the other methods used for the Case Study Analysis and the Precipitation Analysis performed by our team so that our partner organizations can replicate these studies.

II. HYDRAFloods

The Hydrologic Remote Sensing Analysis for Floods (HYDRAFloods) tool is a Python based application developed by NASA SERVIR that uses satellite imagery to produce flood water maps. HYDRAFloods was designed to provide near real-time flood monitoring which can assist with response to flood disasters.

The Case Study Analysis portion of our project utilized HYDRAFloods to map the flooding related to Hurricanes Eta and Iota.

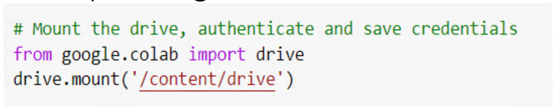
Link to Colab Notebook Script: HYDRAFloods Code Tutorial

The complete documentation for HYDRAFloods can be found here.

A. Set-Up

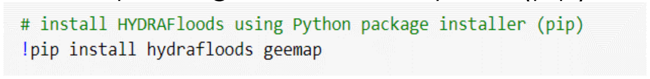
1. For this project, we used Google Colab as our integrated development environment (IDE), as it was freely available to anyone with a Google Drive and Google Earth Engine (GEE) account. The first step is to open Colab in your browser.

2. Mount and authenticate your Google Drive so that your credentials can be saved. When this step is running, a link will appear; click the link, select your Google Drive account, copy the authentication token, and paste it into the corresponding box.



```
# Mount the drive, authenticate and save credentials
from google.colab import drive
drive.mount('/content/drive')
```

3. Use the package installer for Python (pip) to install HYDRAFloods.



```
# install HYDRAFloods using Python package installer (pip)
!pip install hydrafloods geemap
```

4. Import other needed packages: **ee**, **datetime**, **hydrafloods**, and **geemap.eefolium**.

```
# imports
import ee

import datetime

import hydrafloods as hf

import geemap.eefolium as geemap
```

5. Authenticate your Google Earth Engine account. The same steps will be required as with authenticating your Google Drive (refer to step 1).

```
# authenticate GEE Account
_ = geemap.Map()
```

B. Data Acquisition

1. Define your region of interest. HYDRAFloods has a built-in method called **country_bbox** that allows you to enter a country name as a parameter, but only one country may be listed. If you wish to enter multiple countries, you may do so with a feature collection. For this example, we will be examining Honduras.

```
region = hf.country_bbox("Honduras");
```

2. Define the time period using the **datetime** function from the **datetime** package. For this tutorial, we will look at 11/2/2020 – 11/20/2020 corresponding to Hurricanes Eta and Iota.

```
# define time period
# for this example, we will use November 2 - 20, 2020 corresponding to Hurricanes Eta and Iota
startTime = datetime.datetime(2020,11,2);
endTime = datetime.datetime(2020, 11, 20);
```

3. Import imagery using the defined region and time period. Below is an example of retrieving SAR and optical imagery that we will use for this tutorial.

```
# getting SAR imagery from Sentinel-1 based on time period & region
s1 = hf.Sentinel1(region,startTime,endTime);
```

```
# getting optical imagery from Landsat 8 based on time period & region
ls8 = hf.Landsat8(region, startTime, endTime);
```

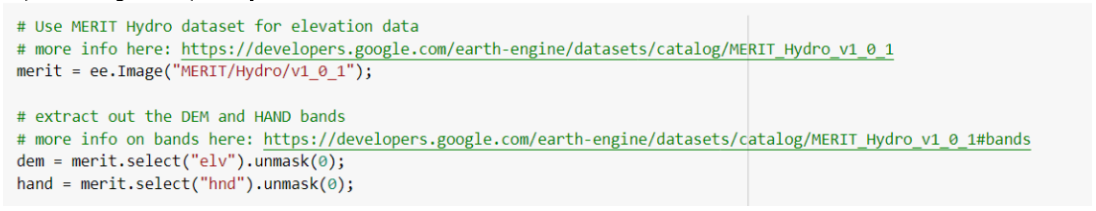
4. Check the number of images you were able to obtain based on the criteria you entered. If the time period is short (one or two days), there may not have been imagery available, so it is good practice to make sure that images were available.

```
# checking for number of images we were able to retrieve from Sentinel-1
s1.n_images;
```

```
# checking number of images we were able to retrieve from Landsat 8
ls8.n_images;
```

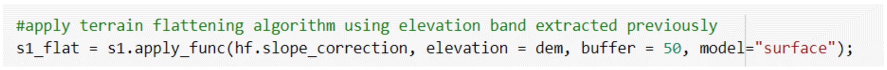
C. Synthetic Aperture Radar (SAR) Imagery Analysis

1. Use the MERIT Hydro dataset to extract the Digital Elevation Model (DEM) and Height Above Nearest Drainage (HAND) bands. This dataset has been hydrologically adjusted for elevation data.

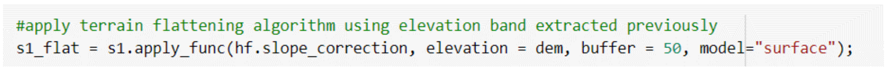


```
# Use MERIT Hydro dataset for elevation data
# more info here: https://developers.google.com/earth-engine/datasets/catalog/MERIT_Hydro_v1_0_1
merit = ee.Image("MERIT/Hydro/v1_0_1");

# extract out the DEM and HAND bands
# more info on bands here: https://developers.google.com/earth-engine/datasets/catalog/MERIT_Hydro_v1_0_1#bands
dem = merit.select("elv").unmask(0);
hand = merit.select("hnd").unmask(0);
```

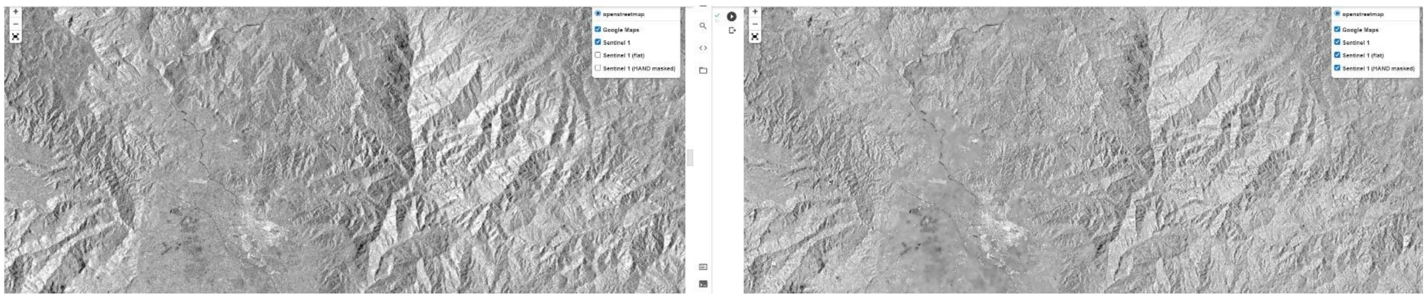
2. Apply a pseudo-terrain flattening algorithm provided by the HYDRAFloods package using the elevation data previously obtained. The map below provides a visual representation of why this is needed. This step is necessary in the processing of SAR imagery and will provide more accurate results. The image was taken from the tutorial and is of Montaña de Comayagua in



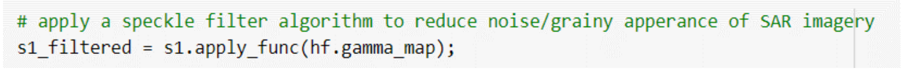
```
#apply terrain flattening algorithm using elevation band extracted previously
s1_flat = s1.apply_func(hf.slope_correction, elevation = dem, buffer = 50, model="surface");
```



```
#apply terrain flattening algorithm using elevation band extracted previously
s1_flat = s1.apply_func(hf.slope_correction, elevation = dem, buffer = 50, model="surface");
```

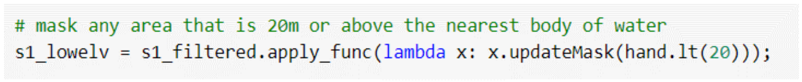


3. Apply a speckle filter to reduce the grainy appearance of the SAR images.



```
# apply a speckle filter algorithm to reduce noise/grainy apperance of SAR imagery
s1_filtered = s1.apply_func(hf.gamma_map);
```

4. Mask any area that has greater than 20 meters Height Above Nearest Drainage (HAND). This will help to provide a more accurate depiction of flooding.



```
# mask any area that is 20m or above the nearest body of water
s1_lowelv = s1_filtered.apply_func(lambda x: x.updateMask(hand.lt(20)));
```

5. Apply a water thresholding algorithm. HYDRAFloods has a built-in method that utilizes the edge otsu algorithm. We specified an edge buffer of 300 meters and a scale of 90 meters. The band is set to VV. If the imagery is

known to be particularly cloudy, the VH band may be better suited. Note: Try setting the scale to 30 meters. If image processing or image exportation is taking too long or times out, then try a scale of 90 meters.

```
# apply edge_otsu water thresholding algorithm
# more info on edge_otsu here: (https://doi.org/10.3390/rs12152469)
water = s1_lowelv.apply_func(hf.edge_otsu,initial_threshold=-16,band="VV",edge_buffer=300,scale=90);
```

6. Reduce the collection into a single image using the **mode** reducer.

```
#reduce collection into single image using reducer "mode"
water_img = water.collection.reduce("mode");
```

7. Map Sentinel-1 imagery to see effects of terrain flattening and HAND masking. An example of this map can be seen above in Step 2.

```
# view the results of SAR water mapping. Layers: Sentinel-1 Data, Sentinel-1 after terrain flattening, after masking HAND of 20m.
Map = geemap.Map(center=(14.4551, -86.9699), zoom=8)

# map generated with SAR data, new map layer
Map.addLayer(s1.collection.median(),{"bands": "VV", "min":-25, "max": 0}, 'Sentinel 1')

# map after applying slope_correction algorithm, new layer
Map.addLayer(s1_flat.collection.median(),{"bands": "VV", "min":-25, "max": 0}, 'Sentinel 1 (flat)')

# map after mask all areas with elevations greater than 20 m above the nearest water body & applying edge_otsu (water thresholding algorithm)
Map.addLayer(s1_lowelv.collection.median(),{"bands": "VV", "min":-25, "max": 0}, 'Sentinel 1 (HAND masked)')

# allows us to switch between layers
Map.addLayerControl()
Map
```

8. Delineate permanent water from the last five years to compare current imagery against. In the GEE Catalog is a dataset from the European Commission's Joint Research Centre (EC JRC) that has global surface water imagery from 1985 to present. However, using only the last five years of data provides a better representation of the current permanent water in the study area.

```
# get the previous 5 years of permanent water for context
# using only previous 5 years allow for more accurate visualization of current permanent water
permanent_water = (
    ee.ImageCollection("JRC/GSW1_2/YearlyHistory") # get the JRC historical dataset
    .filterDate("1985-01-01",endTime) # filter for historical data up to date of interest
    .limit(5, "system:time_start", False) # grab the 5 latest images
    .map(lambda x: x.select("waterClass").eq(3)) # extract out the permanent water class
    .sum() # check if a pixel has been classified as permanent water in the past 5 years
    .unmask(0)
    .gt(0)
).updateMask(water_img.gte(0)) # mask for only the water image we just calculated
```

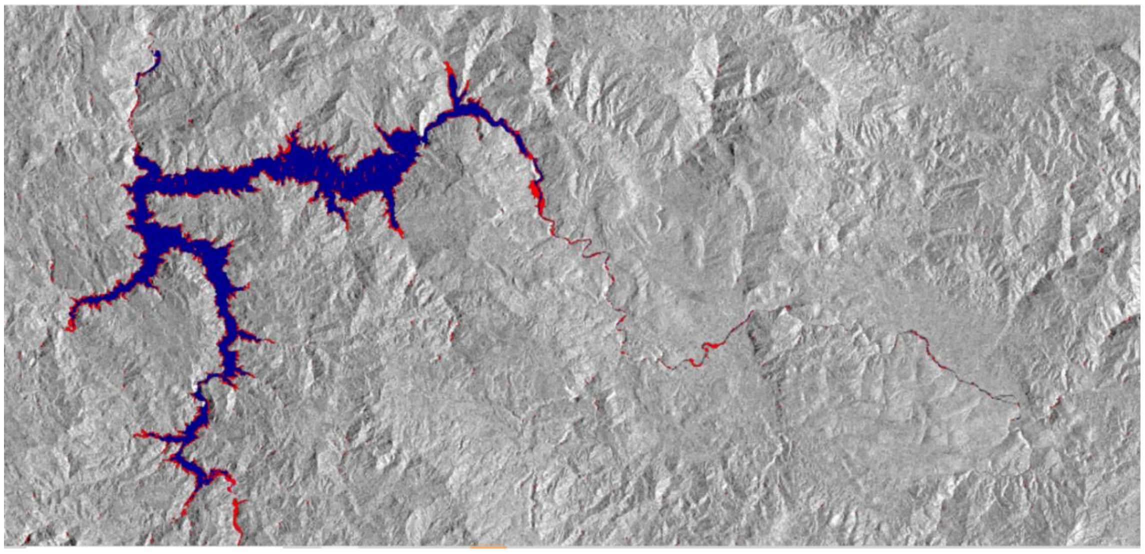
9. Compare the current surface water image versus the permanent surface water to differentiate the flood water.

```
# find where the surface water image says there is water but not permanent water
floods = water_img.add(permanent_water).eq(1)
```

10. View the results by mapping the resulting flood image along with the permanent water image as different layers. We set the color of permanent water to blue and the color of flood water to red.

```
# view the results of SAR water mapping

Map = geemap.Map(center=(14.4551, -86.9699), zoom=8)

#add layer 1 with Senitenl 1 data, layer 2 shows the permanent water
Map.addLayer(s1.collection.median(),{"bands": "VV", "min":-25, "max": 0}, 'Sentinel 1')
Map.addLayer(permanent_water.selfMask(),{"min":0, "max":1,"palette":"black,darkblue"}, 'Permanent Water map')

#this layer shows the "flood" water
#water present that is not permanent water
Map.addLayer(floods.selfMask(),{"min":0, "max":1,"palette":"black,red"}, 'Flood map')


Map.addLayerControl()
Map
```

The image below was taken from the map generated in this tutorial and shows flooding of Río Humuya near Terreritos, Honduras.



11. Export the image(s) if needed using the HYDRAFloods **export_image** function.

```
#export image using HYDRAFloods export image function. below is function with its parameters
#hf.export_image(image, region, asset_id=None, description=None, scale=1000, crs='EPSG:4326', pyramiding=None, export_type='toAsset', folder=None)
#please note the scale is abritrarily set to 1000m, we are exporting at 30m in our example, but
# if the export times out or is taking too long, try exporting at a larger scale (ex 90m)
hf.export_image(floods, region, scale=30, crs='EPSG:4326', pyramiding=".default:mode", export_type='toDrive')
```

D. Optical Imagery Analysis

1. Reduce the image collection using the reducer **mode**.

```
#appy mode reducer to image collection
ls8_image = ls8.collection.reduce('mode')
```

2. Rename the bands for the Modified Normalized Difference Water Index (MNDWI) function.

```
#rename bands for MNDWI function
ls8bands = ls8_image.select(['blue_mode', 'green_mode', 'red_mode', 'nir_mode', 'swir1_mode', 'swir2_mode']).rename(
    ['blue', 'green', 'red', 'nir', 'swir1', 'swir2'])
```

3. Calculate MNDWI using the **apply_func** method provided by the HYDRAFloods package.

```
#get mNDWI (modified Normalized Difference Water Index)
ls8_mndwi = hf.mndwi(ls8bands)
```

4. Apply a water thresholding algorithm. We used the edge otsu algorithm provided by the HYDRAFloods package as we did with the SAR imagery above. In this example, the buffer is set to 300 meters and the scale is set to 30 meters. Note: Try setting the scale to 30 meters. If image processing or image exportation is taking too long or times out, then try a scale of 90 meters.

```
#apply water thresholding algorithm edge otsu, using 300m buffer and 30m scale
landsat8_water = hf.edge_otsu(ls8_mndwi, region = region, edge_buffer = 300, scale = 30, invert = True, thresh_no_data =0.0)
```

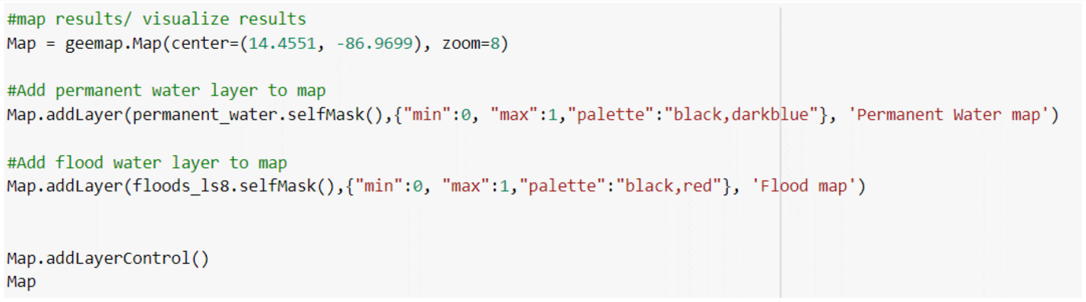
5. Find the permanent water from the last five years to compare our imagery against. The JRC dataset in GEE has global surface water imagery from 1985 to present. However, using only the last five years of data provides a better representation of the current permanent water in the study area.

```
#retrieve last 5 years of permanent water to compare against current water
#this will allow us to differentiate between flood water and permanent surface water
#we use the JRC dataset to retrieve the 'permanent water'
permanent_water = (
    ee.ImageCollection("JRC/GSW1_2/YearlyHistory") # get the JRC historical dataset
    .filterDate("1985-01-01",endTime) # filter for historical data up to date of interest
    .limit(5, "system:time_start", False) # grab the 5 latest images
    .map(lambda x: x.select("waterClass").eq(3)) # extract out the permanent water class
    .sum() # check if a pixel has been classified as permanent water in the past 5 years
    .unmask(0)
    .gt(0)
).updateMask(landsat8_water.gte(0)) # mask for only the water image we just calculated
```

6. Compare the current surface water image versus the permanent surface water to differentiate the flood water.

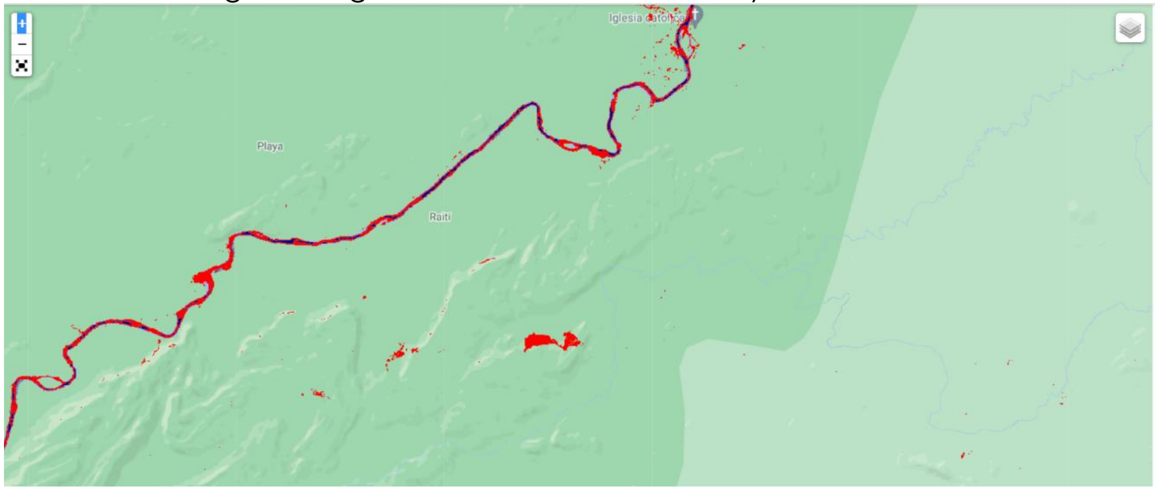
```
#compare our landsat image water detected vs the permanent water map to differentiate flood water
# from permanent surface water
floods_ls8 = landsat8_water.add(permanent_water).eq(1).selfMask()
```

7. Create a map using the result of the previous comparison to visualize the flood water and permanent surface water we found in step 5.

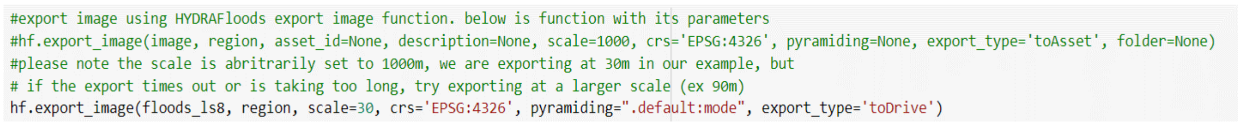


```
#map results/ visualize results
Map = geemap.Map(center=(14.4551, -86.9699), zoom=8)

#Add permanent water layer to map
Map.addLayer(permanent_water.selfMask(),{"min":0, "max":1,"palette":"black,darkblue"}, 'Permanent Water map')

#Add flood water layer to map
Map.addLayer(floods_ls8.selfMask(),{"min":0, "max":1,"palette":"black,red"}, 'Flood map')


Map.addLayerControl()
Map
```

In the map generated in this tutorial, permanent surface water is represented as blue and flood water is represented as red. The image below is showing flooding of the Río Patuca near Playa, Honduras.



8. Export the image(s) if needed using the HYDRAFloods **export_image** function.



```
#export image using HYDRAFloods export image function. below is function with its parameters
#hf.export_image(image, region, asset_id=None, description=None, scale=1000, crs='EPSG:4326', pyramiding=None, export_type='toAsset', folder=None)
#please note the scale is abritrarily set to 1000m, we are exporting at 30m in our example, but
# if the export times out or is taking too long, try exporting at a larger scale (ex 90m)
hf.export_image(floods_ls8, region, scale=30, crs='EPSG:4326', pyramiding=".default:mode", export_type='toDrive')
```

III. Post-Processing of HYDRAFloods Output with Landcover

After mapping flooding in this Case Study Analysis using HYDRAFloods, we also needed to incorporate landcover into our maps. Doing this allows users to better visualize flooding and see what types of landcover were most affected. This will help to better prepare for future flood events.

A. Layering Landcover in Colab

1. Bring in a landcover dataset such as ESA WorldCover 2020.
2. Create a cropland mask. In ESA and Copernicus landcover datasets, cropland corresponds to 40.
3. Create a visualization palette with desired color scheme for landcover.
4. Create a map with permanent water and flood water layers as described in Section II: HYDRAFloods. Add this cropland layer with the visualization palette to the map.
5. Run the script and export as needed.

B. Zonal Statistics

1. In Google Earth Engine, import flood maps generated using HYDRAFloods.
2. Import a watershed shapefile.
3. Import landcover such as ESA WorldCover 2020 and add as a layer to the map.
4. Create masks for each landcover class.
5. Reproject flooding to be the same scale as the landcover scale.
6. Calculate the amount of tree landcover that is flooded. To do this, use the reprojected flood image and use the **and** function with the tree mask created previously.
7. Calculate the area of each pixel for the tree landcover by using the result of the previous calculation and using the **multiply** function with the **ee.Image.pixelArea()**.
8. Calculate the total area (m^2) of tree landcover that was affected. Apply the **reduceRegions** function to the result calculated in step 7. The parameters of this function should be as follows: the collection should be the region of interest (the watershed file variable), the reducer should be the **ee.Reducer.sum()**, and the scale should be the landcover dataset's scale using the **.nominalScale** function.
9. Create a variable defining the properties for the tree landcover type and include the following: **Acres**, **Basin**, **Basin ID**, **Hectares**, **POPN2000**, **Watershed**, and **Tree Cover Sum**.
10. Repeat steps 6-9 for each landcover type.
11. Create a variable defining the generalized properties and include the following: **Acres**, **Basin**, **Basin ID**, **Hectares**, **POPN2000**, **Watershed**, and **Sum**.
12. Export table as needed.
13. To convert the total area flooded for each landcover type from m^2 to km^2, simply divide the result by 1,000,000.

14. To calculate the percentage of each landcover type that was flooded, take the total area flooded and divide it by the total area of the landcover type. To convert this to a percentage, multiply by 100.

IV. Precipitation Analyses

A. Precipitation Analysis – Hurricanes Eta and Iota

A precipitation analysis for Hurricanes Eta and Iota was also conducted as part of this project. Outputs were daily precipitation, total event precipitation, and a daily precipitation graph per country. We performed these analyses in Google Earth Engine.

1. Open the Google Earth Engine Code Editor.
2. Upload or select a region of interest.
3. Import the Climate Hazards Group InfraRed Precipitation with Satellite Data (CHIRPS).
4. Clip CHIRPS to the region of interest.
5. Select the dates to analyze.
6. Create a palette to visualize the precipitation. Use a minimum of 0 and a maximum that corresponds to the highest precipitation amount for the region of interest during the dates selected.
7. Import a watershed file to overlay with precipitation.
8. Run the script.
9. Export from GEE and import into ArcGIS Pro.
10. Upload a shapefile of the region of interest into ArcGIS Pro.
11. In the **Spatial Analyst** toolbox, use the **Extract by Mask** tool to extract the cells of the precipitation raster using the shapefile of the region as a mask.
12. In the **Spatial Analyst** toolbox, use the **Int** tool on the output of step 11 to convert raster cell values to integer values.
13. Symbolize the processed raster as desired.

B. Annual Mean Precipitation

Finding the mean annual precipitation for the area will help to contextualize the results from the above precipitation analysis for the two hurricanes. This analysis was performed in Google Earth Engine following a tutorial from Spatial Thoughts that can be found here.

1. Upload a shapefile of your region of interest.
2. Import the CHIRPS Daily dataset or other gridded rainfall dataset such as CHIRPS Pentad or GPM.
3. Create a map of total rainfall by filtering the collections of images to a year and then applying a **sum()** reducer to obtain a single image. Each pixel is the sum of precipitation for all images in the year.

4. Use the **reduceRegion()** function to compute average total rainfall in your area of interest.
5. Using the previous total rainfall map, calculate the rainfall for one year, and then **map()** that function to your desired list of years.
6. Obtain a CSV table of values.

V. Acknowledgements

The Central America Disasters team would like to thank our science advisors at the Marshall Space Flight Center, Dr. Jeffrey Luvall and the team at the NASA SERVIR Science Coordination Office: Kel Markert, Emil Cherrington, Betzy Hernandez, Vanesa Martin, and Eric Anderson. We would also like to thank Ronan Lucey from the NASA Applied Sciences Disasters Program, Dr. Robert Griffin from the University of Alabama in Huntsville, and our NASA DEVELOP Fellow Paxton LaJoie. Lastly, we would like to recognize our partners and thank them for the collaboration throughout this project: Berta Olmedo Vernaza from CCRH, Marcelo Oyuela from CEPREDENAC, and Jorge Cabrera from SICA.

This material contains modified Copernicus Sentinel data (2020), processed by ESA.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration. This material is based upon work supported by NASA through contract NNL16AA05C.