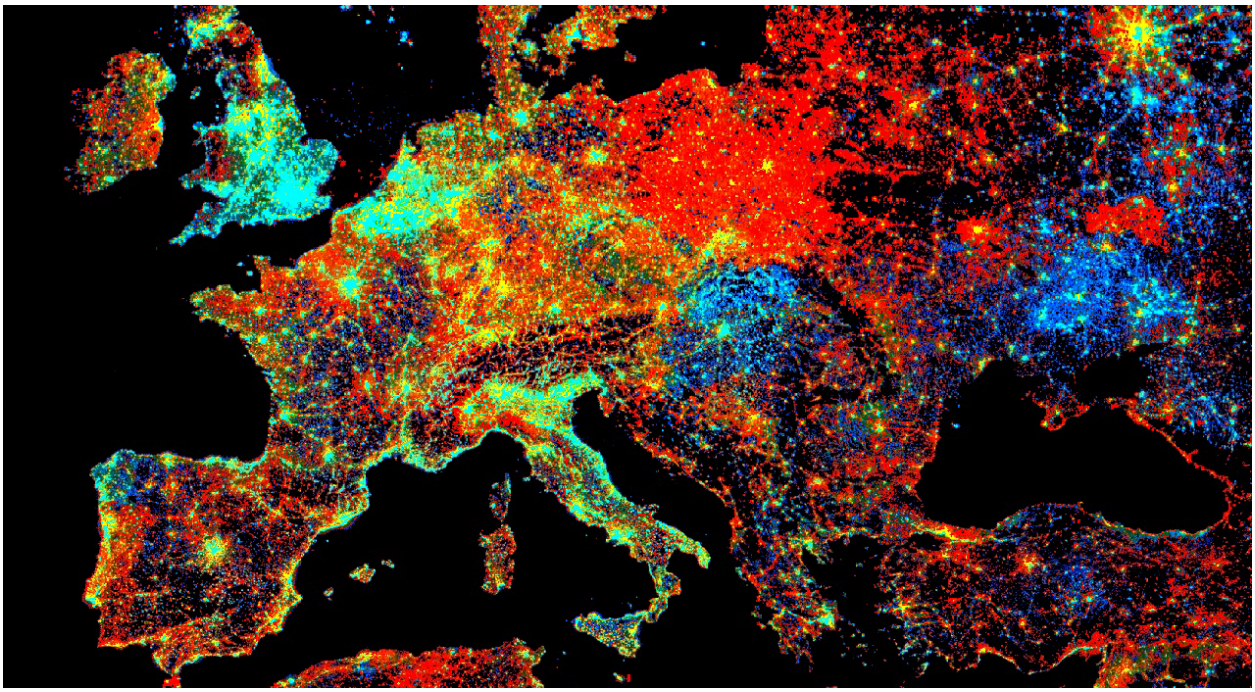
Cloud-Based Remote Sensing with Google Earth Engine



Fundamentals and Applications

or, click here to get back to the master document to access different sections)

Part A2: Aquatic and Hydrological Applications

Earth Engine's global scope and long time series allow analysts to understand the water cycle in new and unique ways. These include surface water in the form of floods and river characteristics, long-term issues of water balance, and the detection of subsurface ground water.

Chapter A2.1: Groundwater Monitoring with GRACE

Authors
A.J. Purdy, J.S. Famiglietti

Overview
The following tutorial details how to use observations from the Gravity Recovery and Climate Experiment (GRACE) to evaluate changes in groundwater storage for a large river basin. Here, you will learn how to apply remote sensing estimates of total water storage anomalies, land surface model output, and in situ observations to resolve groundwater storage changes in California's Central Valley. The following method has been applied to study water storage changes around the world, and can be ported to quantify groundwater storage change for major river basins.

Learning Outcomes
- Plotting changes in total water storage using GRACE.
- Mapping trends in water storage.
- Resolving changes in groundwater storage for a river basin.
- Import image collections and create image collections from assets.
- Create charts by reducing an `ImageCollection` with a feature geometry.

Helps if you know how to:
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Write a function and `map` it over an ImageCollection (Chap. F4.0).
- Fit linear and nonlinear functions with regression in an `ImageCollection` time series (Chap. F4.6).
- Use `ee.Join` to join one `ImageCollection` to another to compute differences (Chap. F4.9).
- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).

Introduction to Theory

Since 2002, GRACE and the follow-on mission, GRACE-FO, have provided a new vantage to track changes in water resources (Tapley et al. 2004). GRACE holds the

unique ability to directly track changes in total water storage anomalies (TWSa), according to the following equation:

$$TWSa = CANa + SWa + SMa + SWEa + GWa \quad (A2.1.1)$$

where CANa is canopy water storage anomaly, SWa is the surface water anomaly, SMa is the soil moisture anomaly, SWEa is the snow water equivalent anomaly, and GWa is the groundwater storage anomaly.

By utilizing supplemental observations from other remote sensing platforms and land surface models and rearranging Eq. A2.1.1, scientists have been able to resolve changes in groundwater storage within major river basins around the planet (Famiglietti et al. 2014). From Bangladesh (Purdy et al. 2019) and India (Rodell et al. 2009) to the Middle East (Voss et al. 2013) and the American Southwest (Castle et al. 2014), the problem of declining groundwater storage has emerged with varying levels of severity (Richey et al. 2015). Along with many other regions around the world, California shares an overreliance on groundwater (Famiglietti et al. 2011). This tutorial demonstrates the analytical steps to resolve groundwater storage changes using GRACE for California's Central Valley.

Practicum

Section 1. Exploring the Study Area

Evaluating changes in hydrologic storage requires examining change within a hydrologically connected system. Watersheds and basins represent areas of land where precipitation drains to a common point. We will use already-generated basins from the Watershed Boundary Dataset (WBD) to delineate the drainage area for California's Central Valley. The WBD includes hydrologic unit codes (HUCs) to identify connected basins within the United States.

In the following sections of code, we will load three basins by their unique four-digit HUCs and merge the basins together. To accomplish this task, we will use the `ee.Filter.inList` function to filter the `basins` variable by the `'huc4'` property, extracting three to a variable `basin`.

```
// Import Basins.
var basins = ee.FeatureCollection('USGS/WBD/2017/HUC04');
```

```
// Extract the 3 HUC 04 basins for the Central Valley.
var codes = ['1802', '1803', '1804'];
var basin = basins.filter(ee.Filter.inList('huc4', codes));

// Add the basin to the map to show the extent of our analysis.
Map.centerObject(basin, 6);
Map.addLayer(basin, {
    color: 'green'
}, 'Central Valley Basins', true, 0.5);
```

Section 1.1. Map the Extent of Agriculture in the Region

To get a sense for the extent of agriculture in California, we can visualize all cultivated land in the Central Valley. This will map where the greatest need for water occurs.

```
var landcover = ee.ImageCollection('USDA/NASS/CDL')
    .filter(ee.Filter.date('2019-01-01', '2019-12-31'))
    .select('cultivated');

Map.addLayer(landcover.first().clip(basin), {}, 'Cropland', true,
    0.5);
```

The extent of cultivated lands shows up as a translucent purple (Fig. A2.1.1).

Fig. A2.1.1 California's Central Valley Basin, including agricultural lands

Section 1.2. Load Reservoir Locations

California has over 150 reservoirs distributed across the state. These reservoirs vary in size and capacity and the regions of the state that they support. For the Central Valley, water conveyance infrastructure allows the transport of water from north to south. We will use our basin boundary to select the reservoirs within our basin to quantify changes in surface water storage. The list of reservoirs was gathered from the California Department of Water Resources' Data Exchange Center (CDEC). For an application in another study region, acquiring in situ surface water storage would be required to resolve that region's groundwater storage changes.

```
// This table was generated using the index from the CDEC website
var res = ee.FeatureCollection(
    'projects/gee-book/assets/A2-1/ca_reservoirs_index');
// Filter reservoir locations by the Central Valley geometry
var res_cv = res.filterBounds(basin);
Map.addLayer(res_cv, {
    'color': 'blue'
}, 'Reservoirs');
```

The blue dots that now appear on the map represent the distribution of water storage across the Central Valley. Water conveyance infrastructure and natural rivers deliver water to farms across the valley. Despite all these reservoirs, many water users continue to rely on groundwater to meet their needs. A 2011 study detailed the magnitude of this reliance using gravity-sensing satellites (Famiglietti et al. 2011). The next sections of this chapter reveal how to resolve groundwater storage changes using these methods.

Code Checkpoint A21a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Tracking Total Water Storage Changes in California with GRACE

GRACE can directly track changes in TWSa. Changes in TWSa indicate which regions are gaining or losing water.

Section 2.1. Import GRACE Data and Plot Changes in Total Water Storage in California

First, we will import the image collection and select the proper band to chart.

```
var GRACE = ee.ImageCollection('NASA/GRACE/MASS_GRIDS/MASCON_CRI');
// Subset GRACE for liquid water equivalent dataset
var basinTWSa = GRACE.select('lwe_thickness');
```

The GRACE data imported here have already been processed to provide units of TWSa. The data contained in this dataset are units of "equivalent water thickness" anomalies. GRACE hydrologic data are presented as anomalies because GRACE does not directly observe the gravitational pull of only water. The gravity observed also includes Earth's surface (e.g., mountains). To disentangle the signal of water we can look at changes relative to a longer term mean gravity signal. The anomalies represent the difference between a given month's observation and a multi-year mean. We will now plot TWSa for basins in a large part of California (Fig. A2.1.2).

```
// Make plot of TWSa for Basin Boundary
var TWSaChart = ui.Chart.image.series({
        imageCollection: basinTWSa.filter(ee.Filter.date(
            '2003-01-01', '2016-12-31')),
```

```
        region: basin,
        reducer: ee.Reducer.mean(),
    })
    .setOptions({
        title: 'TWSa',
        hAxis: {
            format: 'MM-yyyy'
        },
        vAxis: {
            title: 'TWSa (cm)'
        },
        lineWidth: 1,
    });
print(TWSaChart);
```

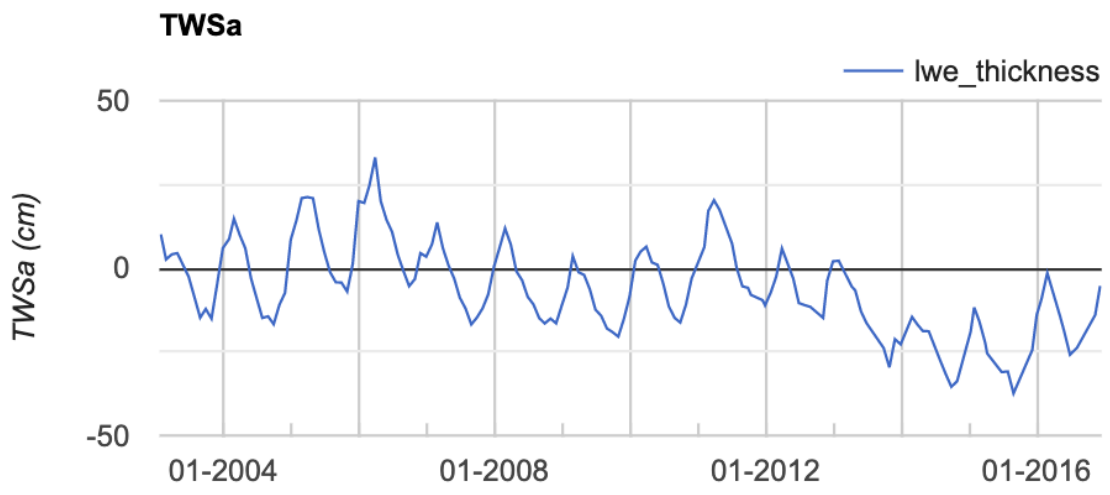


Fig. A2.1.2 TWSa for the Sacramento-San Joaquin Basin

In the **Console**, you will see a plot of TWSa. Notice the seasonality and interannual variations in TWSa. Winter months reveal periods of maximum water storage due to snowpack, full reservoirs, and wet soil. Summer and early fall reveal less TWSa, as the snow has melted, reservoir water has been used, and soil is drying out. Additionally, summer months are periods when groundwater is extracted and used to supplement a

limited surface water supply. Evidence of drought emerged through declining TWSa between 2006–2009 and 2012–2017.

Next, we will look at the trend in TWSa for the entire period of record.

Section 2.2. Estimate the Linear Trend in TWSa Over Time

As presented in Chap. F4.6, Earth Engine can fit linear models to time series data, with unique linear fits for each pixel based on the values through time. Consider the following linear model, where et is a random error:

$$p_t = \beta_0 + \beta_1 t + e_t \quad (1) \qquad \text{(A2.1.2)}$$

This is the model behind the trendline added to the chart we just created. This model is useful for detrending data and reducing non-stationarity in the time series (Shumway and Stoffer 2017). The goal of the regression is to discover the values of the β's in each pixel.

To fit this trend model to the GRACE-based TWSa series using ordinary least squares, we can use the `linearRegression` reducer.

```
// Compute Trend for each pixel to map regions of most change
var addVariables = function(image) {
    // Compute time in fractional years since the epoch.
    var date = ee.Date(image.get('system:time_start'));
    var years = date.difference(ee.Date('2003-01-01'), 'year');
    // Return the image with the added bands.
    return image
        // Add a time band.
        .addBands(ee.Image(years).rename('t').float())
        // Add a constant band.
        .addBands(ee.Image.constant(1));
};
var cvTWSa = basinTWSa.filterBounds(basin).map(addVariables);
print(cvTWSa);
// List of the independent variable names
var independents = ee.List(['constant', 't']);
```

```
// Name of the dependent variable.
var dependent = ee.String('lwe_thickness');
// Compute a linear trend.  This will have two bands: 'residuals' and
// a 2x1 band called coefficients (columns are for dependent
variables).
var trend = cvTWSa.select(independents.add(dependent))
    .reduce(ee.Reducer.linearRegression(independents.length(), 1));
```

The image of coefficients, computed below, is a two-band image in which each pixel contains values for β0 and β1. The β1 value will represent the temporal slope for the GRACE mascon.

```
// Flatten the coefficients into a 2-band image
var coefficients = trend.select('coefficients')
    .arrayProject([0])
    .arrayFlatten([independents]);
```

Next, we can visualize the GRACE trends to capture the spatial scales on which GRACE can resolve TWSa. GRACE is adept at capturing these changes only for larger basins.

```
// Create a layer of the TWSa slope to add to the map
var slope = coefficients.select('t');
// Set visualization parameters to represent positive (blue) &
negative (red) trends
var slopeParams = {
    min: -3.5,
    max: 3.5,
    palette: ['red', 'white', 'blue']
};
Map.addLayer(slope.clip(basin), slopeParams, 'TWSa Trend', true,
    0.75);
```

The slope layer reveals that the Tulare Basin (the southernmost basin in the Central Valley) experienced the largest negative changes in TWSa over the time period (Fig. A2.1.3). Darker reds indicate greater negative change and blue represents positive

change. This is a result of the region not receiving winter rain or snow and having the most junior surface water rights in the Central Valley.

The next steps in this chapter will review how to unpack the TWSa signal to resolve changes in groundwater storage anomalies for the basin.

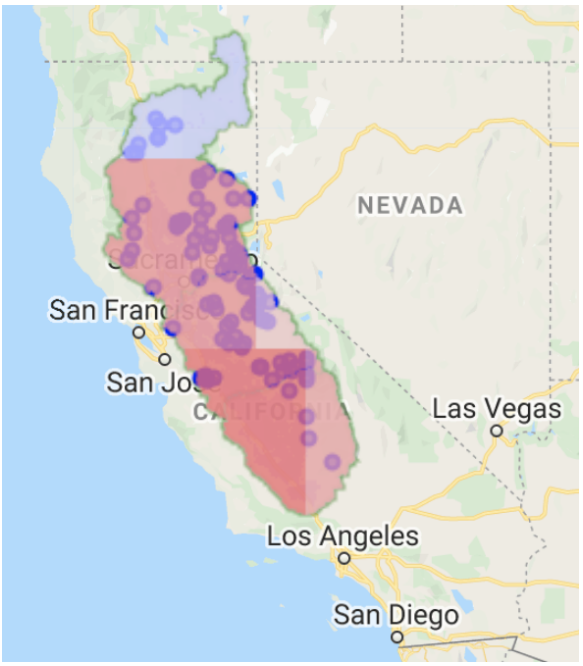


Fig. A2.1.3 Slope in TWSa for California. Darker reds indicate greater declines in total water storage. Blue represents increases in water storage. White represents no change in water storage.

Code Checkpoint A21b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Tracking Changes in Soil Water Storage and Snow Water Equivalent in California

The Global Land Data Assimilation System (GLDAS) utilizes multiple land surface models to globally resolve fluxes in storage of water (like soil moisture and snow) and energy at a three-hour frequency (Rodell et al. 2004). An example of how to convert three-hourly GLDAS snow water equivalent to annual SWEa for 2003 can be found in script **A21s1** in the book's repository. Running the supplemental script is an optional part

of this lab: it is added to provide clarity on how the image assets were created for each GLDAS variable in this chapter.

For the next analysis, you will be starting with a script that imports the GLDAS SMa and SWEa processed by the methods above. GLDAS estimates of soil moisture and snow water equivalent are resolved at a three-hour temporal frequency. Therefore, we have taken the time to reduce the GLDAS data to annual means from the three-hour estimates.

Additionally, we aggregated monthly GRACE observations to annual average estimates to improve the efficiency of running this analysis. More experienced users can adapt these methods to resolve monthly changes. However, it should be noted that to replicate the same methods at a monthly cadence would require the interpolation of missing months of GRACE observations. Please use the code starting point below, as the script imports the necessary assets to complete the final analysis.

Section 3.1. Load GLDAS Soil Moisture Images from an Asset to an Image Collection

Code Checkpoint A21c. The book's repository contains a script to use to begin this section. You will need to start with that script and paste code below into it.

When you run the script, you will see a number of assets being imported and an annual time series of GRACE. Additionally, the script is set to convert the list of annual mean soil moisture images to an `ImageCollection`.

```
var gldas_sm_list = ee.List([sm2003, sm2004, sm2005, sm2006, sm2007,
    sm2008, sm2009, sm2010, sm2011, sm2012, sm2013, sm2014,
    sm2015, sm2016
]);
var sm_ic = ee.ImageCollection.fromImages(gldas_sm_list);
```

Before we compute groundwater storage anomalies from GRACE and GLDAS data following Eq. A2.1.1, we should inspect the units to ensure that our math is sound. In the **search bar** of Earth Engine, search for "GLDAS" and click on **GLDAS-2.1: Global Land Data Assimilation System**, then navigate to **Bands** to see what the units are for `'RootMoist_inst'` and `'SWE_inst'`.

The units for GLDAS are currently showing as kg/m^2. We need to convert the soil moisture and snow values to equivalent water depth units of centimeters. Define the following conversion variable and map this over the `ImageCollection`. As described in Chap. F4.0 and Chap. F4.1, mapping over an `ImageCollection` is similar to running a loop: You apply the same function to each image and return the value back to the `ImageCollection`.

```
var kgm2_to_cm = 0.10;
var sm_ic_ts = sm_ic.map(function(img) {
    var date = ee.Date.fromYMD(img.get('year'), 1, 1);
    return img.select('RootMoist_inst').multiply(kgm2_to_cm)
        .rename('SMa').set('system:time_start', date);
});
```

In addition to converting the units, the code renames the variable and sets properties such as `'system:time_start'`, which is necessary in Earth Engine to plot data and compare it with other image collections. Note that you might print out the variables `sm_ic` and `sm_ic_ts` to explore the differences between them. You should notice the new band name and properties (e.g., `'system:time_start'`).

Next, plot the data to evaluate soil moisture anomalies during the study period.

```
// Make plot of SMa for Basin Boundary
var SMaChart = ui.Chart.image.series({
        imageCollection: sm_ic_ts.filter(ee.Filter.date(
            '2003-01-01', '2016-12-31')),
        region: basin,
        reducer: ee.Reducer.mean(),
        scale: 25000
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Soil Moisture anomalies',
        trendlines: {
            0: {
                color: 'CC0000'
```

```
            }
        },
        hAxis: {
            format: 'MM-yyyy'
        },
        vAxis: {
            title: 'SMa (cm)'
        },
        lineWidth: 2,
        pointSize: 2
    });
print(SMaChart);
```

You may notice that SMa is of a similar magnitude to TWSa, but is slightly out of phase with TWSa.

Section 3.2. Load GLDAS Snow Water Equivalent Images from an Asset to an Image Collection

Use similar code to load the snow water equivalent data to Earth Engine.

```
var gldas_swe_list = ee.List([swe2003, swe2004, swe2005, swe2006,
    swe2007, swe2008, swe2009, swe2010, swe2011, swe2012,
    swe2013, swe2014, swe2015, swe2016
]);
var swe_ic = ee.ImageCollection.fromImages(gldas_swe_list);
```

Next, convert the snow values to equivalent water depth units of centimeters.

```
var swe_ic_ts = swe_ic.map(function(img) {
    var date = ee.Date.fromYMD(img.get('year'), 1, 1);
    return img.select('SWE_inst').multiply(kgm2_to_cm).rename(
        'SWEa').set('system:time_start', date);
});
```

Next, we will visualize the new `ImageCollection`. If you did not do the previous step, your code here will not run.

```
// Make plot of SWEa for Basin Boundary
var SWEaChart = ui.Chart.image.series({
        imageCollection: swe_ic_ts.filter(ee.Filter.date(
            '2003-01-01', '2016-12-31')),
        region: basin,
        reducer: ee.Reducer.mean(),
        scale: 25000
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Snow Water Equivalent anomalies',
        trendlines: {
            0: {
                color: 'CC0000'
            }
        },
        hAxis: {
            format: 'MM-yyyy'
        },
        vAxis: {
            title: 'SWEa (cm)'
        },
        lineWidth: 2,
        pointSize: 2
    });
print(SWEaChart);
```

You successfully plotted soil moisture and snow water equivalent (Fig. A2.1.4). You may notice that SWEa is much smaller in magnitude than the other two variables.

Code Checkpoint A21d. The book's repository contains a script that shows what your code should look like at this point.

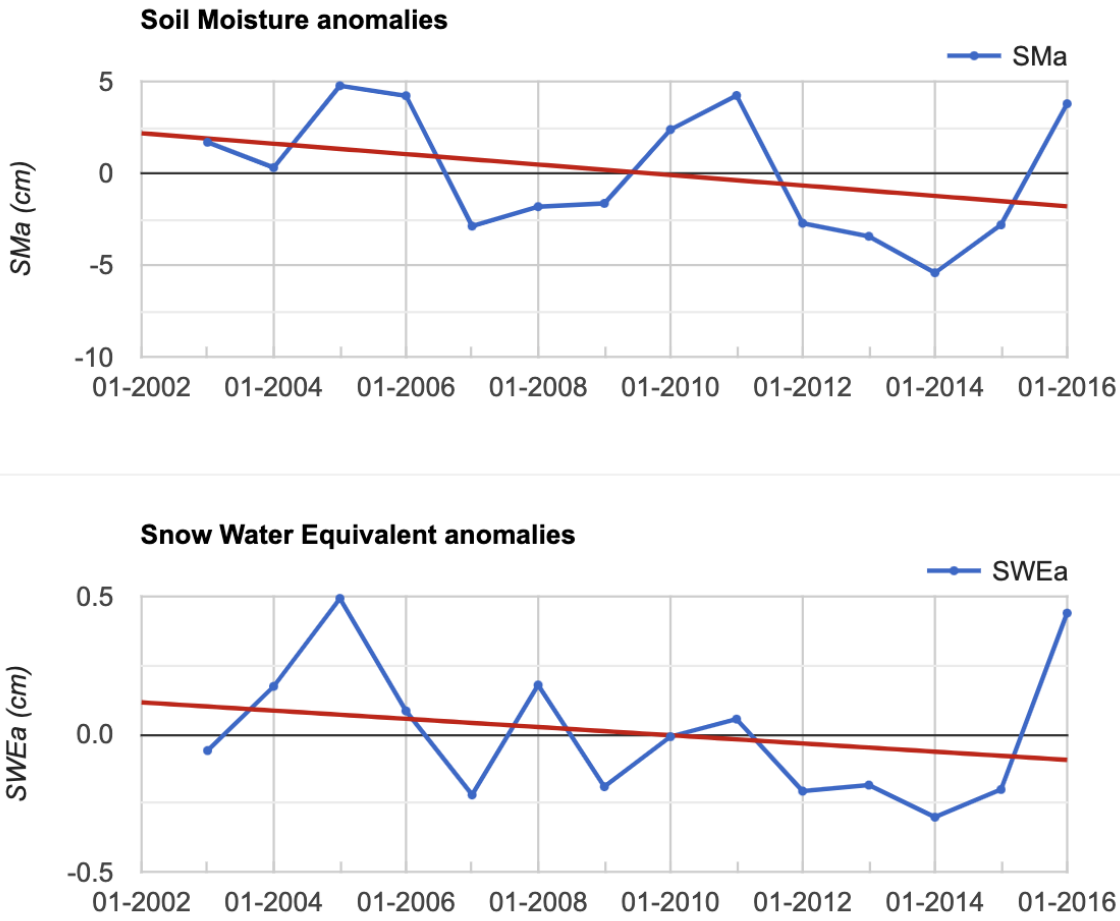


Fig. A2.1.4 Time-series charts of SMa and SWEa in units of equivalent water height (centimeters)

Section 4. Importing a Table of Surface Water Storage

Reservoir storage data from the California Data Exchange Center (CDEC) facilitated computing Surface Water storage anomalies (SWa) for the Sacramento-San Joaquin Basin. Surface water storage, unlike the other components of water storage, is not represented in land surface models. Instead, SWa is sourced from in situ observations. Here, the reservoir storage observations were summed for the Central Valley and then total converted to annual anomalies to directly compare with SWEa and SMa. Prior to reading the table of reservoir storage, we compute the area from the combined HUC8 basins.

```
// Extract geometry to convert time series of anomalies in km3 to cm
var area_km2 = basin.geometry().area().divide(1000 * 1000);
var km_2_cm = 100000;
```

Next, the imported table `res_table` is converted to an `ImageCollection` of constant values to facilitate combining the data with GRACE and GLDAS-resolved water storage anomalies.

```
// Convert csv to image collection
var res_list = res_table.toList(res_table.size());
var yrs = res_list.map(function(ft) {
    return ee.Date.fromYMD(ee.Feature(ft).get('YEAR'), 1, 1);
});
var SWanoms = res_list.map(function(ft) {
    return ee.Image.constant(ee.Feature(ft).get('Anom_km3'));
});
var sw_ic_ts = ee.ImageCollection.fromImages(
    res_list.map(
        function(ft) {
            var date = ee.Date.fromYMD(ee.Feature(ft).get('YEAR'),
                1, 1);
            return ee.Image.constant(ee.Feature(ft).get(
                'Anom_km3')).divide(area_km2).multiply(
                km_2_cm).rename('SWa').set(
                'system:time_start', date);
        }
    )
);
```

Plot SWa in equivalent units of centimeters per year (Fig. A2.1.5).

```
// Create a time series of Surface Water Anomalies
var SWaChart = ui.Chart.image.series({
        imageCollection: sw_ic_ts.filter(ee.Filter.date(
            '2003-01-01', '2016-12-31')),
```

```
        region: basin,
        reducer: ee.Reducer.mean(),
        scale: 25000
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Surface Water anomalies',
        trendlines: {
            0: {
                color: 'CC0000'
            }
        },
        hAxis: {
            format: 'MM-yyyy'
        },
        vAxis: {
            title: 'SWa (cm)'
        },
        lineWidth: 2,
        pointSize: 2
    });
print(SWaChart);
```

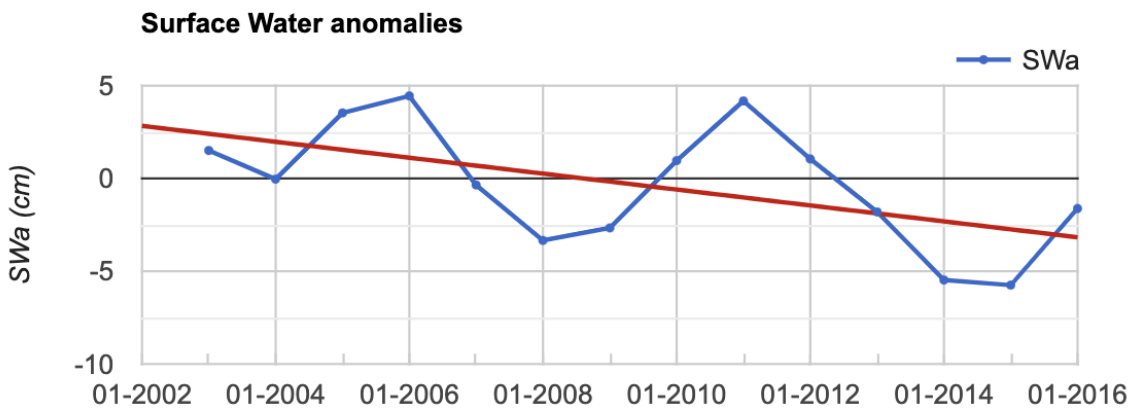


Fig. A2.1.5 Time-series chart of SWa in units of equivalent water height (centimeters)

The chart shows that surface water anomalies are of a similar magnitude to soil moisture anomalies. As expected, SWa decreases during each drought period in California (2006–2008 and 2012–2016). These reservoir storage declines show use is greater than inputs during each period.

Now, we will combine the previous datasets to resolve changes in groundwater during the period of record. Unfortunately, it's still hard to quantify change without having all the variables on one plot. It might be best to compute the differences via Eq. A2.1.1 from the introductory paragraph at the top of the document.

Code Checkpoint A21e. The book's repository contains a script that shows what your code should look like at this point.

Section 5. Combining Image Collections

Here, you will see how to combine multiple image collections and compute differences via an expression. We will start by joining the GLDAS image collections together. This is accomplished with the `ee.Join.inner` function.

```
// Combine GLDAS & GRACE Data to compute change in human accessible
water
var filter = ee.Filter.equals({
    leftField: 'system:time_start',
    rightField: 'system:time_start'
});
// Create the join.
var joindata = ee.Join.inner();
// Join GLDAS data
var firstJoin = ee.ImageCollection(joindata.apply(swe_ic_ts, sm_ic_ts,
    filter));
var join_1 = firstJoin.map(function(feature) {
    return ee.Image.cat(feature.get('primary'), feature.get(
        'secondary'));
});
print('Joined', join_1);
```

Next, we join the reservoir data.

```
// Repeat to append Reservoir Data now
var secondJoin = ee.ImageCollection(joindata.apply(join_1, sw_ic_ts,
    filter));
var res_GLDAS = secondJoin.map(function(feature) {
    return ee.Image.cat(feature.get('primary'), feature.get(
        'secondary'));
});
```

Lastly, we need to repeat this step by joining GRACE to the output from the last join.

```
// Repeat to append GRACE now
var thirdJoin = ee.ImageCollection(joindata.apply(res_GLDAS, GRACE_yr,
    filter));
var GRACE_res_GLDAS = thirdJoin.map(function(feature) {
    return ee.Image.cat(feature.get('primary'), feature.get(
        'secondary'));
});
```

Take a moment to print out the `ImageCollection` `GRACE_res_GLDAS`.

To resolve groundwater storage changes in the basin, one can rearrange Eq. A2.1.1 to solve for GWa. Here we assume canopy storage anomalies are very small relative to other storage components and ignore them in the equation below.

GWa = TWSa - SWa - SMa - SWEa (A2.1.2)

To execute this step we map an expression across an `ImageCollection` to produce a new variable named `GWa`.

```
// Compute groundwater storage anomalies
var GWa = ee.ImageCollection(GRACE_res_GLDAS.map(function(img) {
    var date = ee.Date.fromYMD(img.get('year'), 1, 1);
    return img.expression(
        'TWSa - SWa - SMa - SWEa', {
            'TWSa': img.select('TWSa'),
            'SMa': img.select('SMa'),
```

```
            'SWa': img.select('SWa'),
            'SWEa': img.select('SWEa')
        }).rename('GWa').copyProperties(img, [
        'system:time_start'
    ]);
}));
print('GWa', GWa);
```

You can see how the variable `img` is used to extract bands from the combined `ImageCollection` and create a new one with just one band.

We'll plot this to see how groundwater storage is changing (Fig. A2.1.6).

```
// Chart Results
var GWaChart = ui.Chart.image.series({
        imageCollection: GWa.filter(ee.Filter.date('2003-01-01',
            '2016-12-31')),
        region: basin,
        reducer: ee.Reducer.mean(),
        scale: 25000
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Changes in Groundwater Storage',
        trendlines: {
            0: {
                color: 'CC0000'
            }
        },
        hAxis: {
            format: 'MM-yyyy'
        },
        vAxis: {
            title: 'GWa (cm)'
        },
        lineWidth: 2,
```

```
        pointSize: 2
    });
print(GWaChart);
```

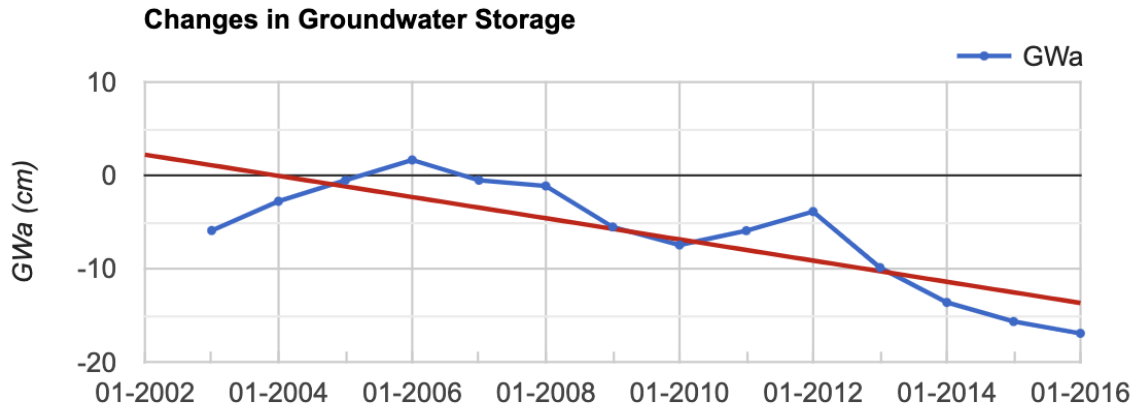


Fig. A2.1.6 Time-series chart of GWa in units of equivalent water height (centimeters)

You can see how reliant California is on groundwater. The chart shows large declines during recent drought periods. Using the chart, you can estimate how much groundwater was used during the 2012–2016 drought period. In the **Console**, hover your mouse over the chart and jot down the value for GWa in 2012 and in 2016. You will use this information, in addition to the area and unit conversion, to estimate groundwater usage during this period in cubic kilometers.

```
// Now look at the values from the start of 2012 to the end of 2016
drought.
// 2012 -3.874 cm --> 2016 -16.95 cm
// This is a ~13 cm / 100000 (cm/km) * Area 155407 km2 =
var loss_km3 = ee.Number(-3.874).subtract(-16.95).divide(km_2_cm)
    .multiply(area_km2);
print('During the 2012-2016 drought, CA lost ', loss_km3,
    'km3 in groundwater');
```

Code Checkpoint A21f. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Assignment 1. This chapter provides a roadmap to monitor changes in groundwater storage at a basin scale using observations of TWSa from the GRACE satellites and hydrologic data from GLDAS. Now you can apply these methods to another river basin anywhere in the world.

Conclusion

In this chapter, we reviewed how GRACE observations can be used to estimate changes in water storage for a region of interest like California's Central Valley. Specifically, this chapter demonstrated how to combine equivalent water thickness observations from GRACE with model simulations of soil moisture, snow water equivalent, and in situ reservoir storage observations to quantify groundwater storage declines. Along the way, some advanced Earth Engine skills were explored, including creating an `ImageCollection` from a table and joining multiple image collections. Earth Engine users now have the skills and the knowledge of GRACE observations to apply this methodology to other regions around the world.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Castle SL, Thomas BF, Reager JT, et al (2014) Groundwater depletion during drought threatens future water security of the Colorado River Basin. Geophys Res Lett 41:5904–5911. https://doi.org/10.1002/2014GL061055

Famiglietti JS (2014) The global groundwater crisis. Nat Clim Chang 4:945–948. https://doi.org/10.1038/nclimate2425

Famiglietti JS, Lo M, Ho SL, et al (2011) Satellites measure recent rates of groundwater depletion in California's Central Valley. Geophys Res Lett 38 https://doi.org/10.1029/2010GL046442

Purdy AJ, David CH, Sikder MS, et al (2019) An open-source tool to facilitate the processing of GRACE observations and GLDAS outputs: An evaluation in Bangladesh. Front Environ Sci 7 https://doi.org/10.3389/fenvs.2019.00155

Richey AS, Thomas BF, Lo MH, et al (2015) Quantifying renewable groundwater stress with GRACE. Water Resour Res 51:5217–5237. https://doi.org/10.1002/2015WR017349

Rodell M, Houser PR, Jambor U, et al (2004) The global land data assimilation system. Bull Am Meteorol Soc 85:381–394. https://doi.org/10.1175/BAMS-85-3-381

Rodell M, Velicogna I, Famiglietti JS (2009) Satellite-based estimates of groundwater depletion in India. Nature 460:999–1002. https://doi.org/10.1038/nature08238

Voss KA, Famiglietti JS, Lo M, et al (2013) Groundwater depletion in the Middle East from GRACE with implications for transboundary water management in the Tigris-Euphrates-Western Iran region. Water Resour Res 49:904–914. https://doi.org/10.1002/wrcr.20078

Chapter A2.2: Benthic Habitats

Authors:
Dimitris Poursanidis, Aurélie C. Shapiro, Spyros Christofilakos

Overview

Shallow-water coastal benthic habitats, which can comprise seagrasses, sandy soft bottoms, and coral reefs, are essential ecosystems, supporting fisheries, providing coastal protection, and sequestering "blue" carbon. Multispectral satellite imagery, particularly with blue and green spectral bands, can penetrate clear, shallow water, allowing us to identify what lies on the seafloor. In terrestrial habitats, atmospheric and topographic corrections are important, whereas in shallow waters, it is essential to correct the effects of the water column, as different depths can change the reflectance observed by the satellite sensor. Once you know water depth, you can accurately assess benthic habitats such as seagrass, sand, and coral. In this chapter, we will describe how to estimate water depth from high-resolution Planet data and map benthic habitats.

Learning Outcomes

- Separating land from water with a supervised classification.
- Identifying and classifying benthic habitats using machine learning.
- Removing surface sunglint and wave glint.
- Developing regression models to estimate water depth using training data.
- Correcting for water depth to derive bottom surface reflectance through a regression approach.
- Evaluating training data and model accuracy.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks, classify images (Part F2).
- Use `normalizedDifference` to calculate vegetation indices (Chap. F2.0).
- Use drawing tools to create points, lines, and polygons (Chap. F2.1).
- Perform a supervised Random Forest image classification (Chap. F2.1).
- Obtain accuracy metrics from classifications (Chap. F2.2).

- Use reducers to implement linear regression between image bands (Chap. F3.0).
- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).
- Convert between raster and vector data (Chap. F5.1).

Introduction to Theory

Coastal benthic habitats include several ecosystems across the globe. One common element is the seagrass meadow, a significant component of coastal marine ecosystems (UNEP/MAP 2012). Seagrass meadows are among the most productive habitats in the coastal zone, performing essential ecosystem functions and providing essential ecosystem services (Duarte et al. 2011), such as water oxygenation and nutrient provision, seafloor and beach stabilization (as sediment is controlled and trapped within the rhizomes of the meadows), carbon burial, and nursery areas and refuge for commercial and endemic species (Boudouresque et al. 2012, Vassallo et al. 2013, Campagne et al. 2015).

However, seagrass meadows are experiencing a global decline due to intensive human activities and climate change (Boudouresque et al. 2009). Threats from climate change include sea surface temperature increases and sea level rise, as well as more frequent and intensive storms (Pergent 2014). These threats represent a pressing challenge for coastal management and are predicted to have deleterious effects on seagrasses.

Several programs have focused on coastal seabed mapping, and a wide range of methods have been utilized for mapping seagrasses (Borfecchia et al. 2013, Eugenio et al. 2015). Satellite remote sensing has been employed for the mapping of seagrass meadows and coral reefs in several areas (Goodman et al. 2013, Hedley et al. 2016, Knudby and Nordlund 2011, Koedsin et al. 2016, Lyons et al. 2012).

In this chapter, we will show you how to map coastal habitats using high-resolution Earth observation data from Planet with updated field data collected in the same period as the imagery acquisition. You will learn how to calculate coastal bathymetry using high-quality field data and machine learning regression methods. In the end, you will be able to adapt the code and use your own data to work in your own coastal area of interest, which can be tropical or temperate as long as there are clear waters up to 30 m deep.

Practicum

Section 1. Inputting Data

The first step is to define the data that you will work on. By uploading raster and vector data via the Earth Engine asset inventory, you will be able to analyze and process them through the Earth Engine API. The majority of satellite data are stored with values scaled by 10000 and truncated in order to occupy less memory. In this setting, it is crucial to scale them back to their physical values before processing them.

```
// Section 1
// Import and display satellite image.
var planet = ee.Image('projects/gee-book/assets/A2-2/20200505_N2000')
    .divide(10000);

Map.centerObject(planet, 12);
var visParams = {
    bands: ['b3', 'b2', 'b1'],
    min: 0.17,
    max: 0.68,
    gamma: 0.8
};
Map.addLayer({
    eeObject: planet,
    visParams: visParams,
    name: 'planet initial',
    shown: true
});
```

Code Checkpoint A22a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Preprocessing Functions

Code Checkpoint A22b. The book's repository contains a script to use to begin this section. You will need to start with that script and paste code below into it. When you run

the script, you will see several assets being imported for water, land, sunglint and sandy patches to correct for the water column.

With the Planet imagery imported into Earth Engine, we now need to prepare for the classification and bathymetry procedures that will follow. The aim of the preprocessing is to correct or minimize spectral alterations due to physical conditions like sunglint, waves, and the water column. The `landmask` function below will remove the land area from our image in order to focus on the marine area of interest. This prevents the land reflectance values, which are relatively high, from biasing the main process of detection over relatively dark water. To implement this step, we will perform a supervised classification to obtain the land/water mask. To prepare for the classification, we drew water and land geometry imports and set them as a `FeatureCollection` with property `'class'` and values 1 and 0, respectively (Fig. A2.2.1). We will employ the Normalized Difference Water Index (NDWI) (Gao 1996) using a Random Forest classifier (Breiman 2001). Due to the distinct spectral reflectances of terrain and water surfaces, there is no need to ensure a balanced dataset between the two classes. However, the size of the training dataset is still important and therefore, the more the better. For this task, we create 'line' geometries because we can get more training points than 'point' geometries with fewer clicks. For more information regarding generating training lines, points or polygons, please see Chap. F2.1.

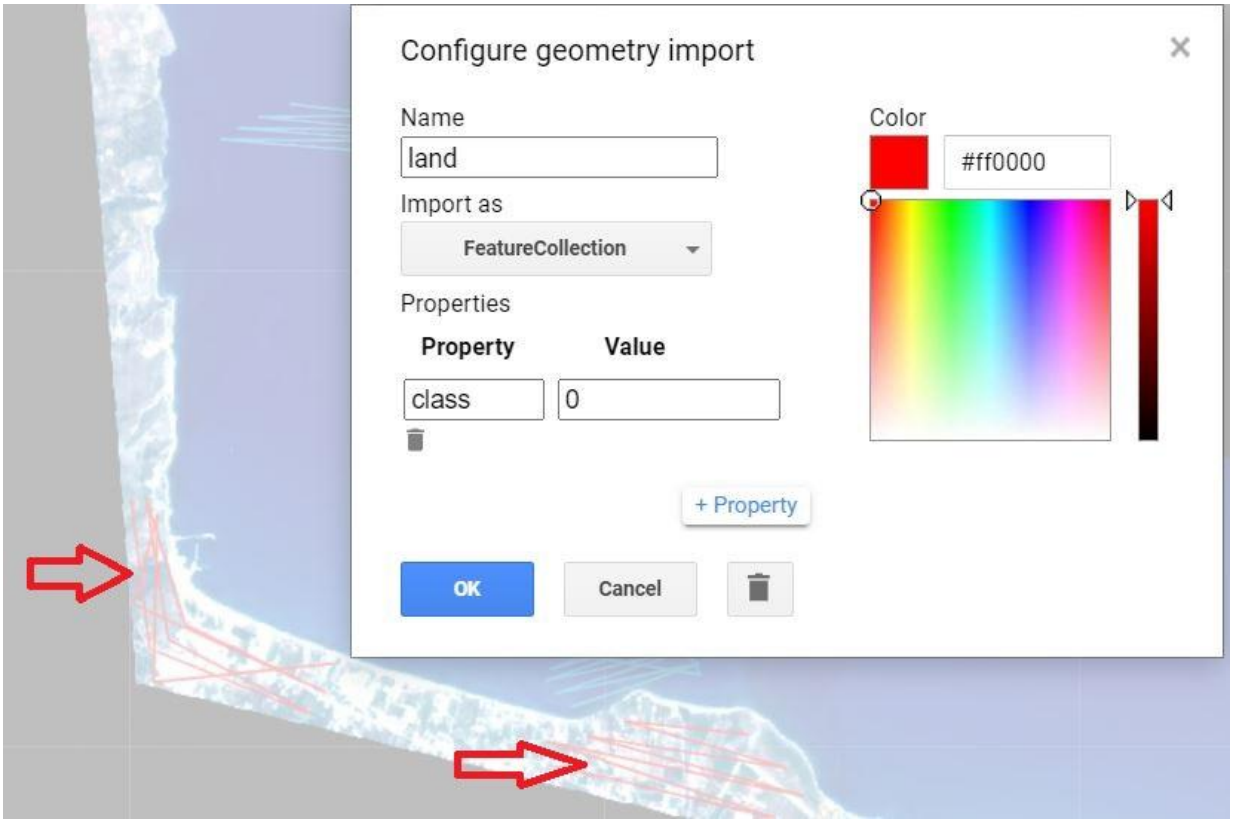


Fig. A2.2.1 Labeling of land and water regions using lines in preparation for the Random Forests classification

```
// Section 2
// Mask based to NDWI and RF.
function landmask(img) {
    var ndwi = img.normalizedDifference(['b2', 'b4']);
    var training = ndwi.sampleRegions(land.merge(water), ['class'],
        3);
    var trained = ee.Classifier.smileRandomForest(10)
        .train(training, 'class');
    var classified = ndwi.classify(trained);
    var mask = classified.eq(1);

    return img.updateMask(mask);
```

```
}

var maskedImg = landmask(planet);

Map.addLayer(maskedImg, visParams, 'maskedImg', false);
```

Sunglint is a phenomenon that occurs when the sun angle and the sensor are positioned such that there is a mirror-like reflection at the water surface. In areas of glint, we cannot detect reflectance from the ocean floor. This will affect image processing and needs to be corrected. The user adds polygons identifying areas of glint (Fig. A2.2.2), and these areas are removed using linear modeling (Hedley et al. 2005).

```
// Sun-glint correction.
function sunglintRemoval(img) {
    var linearFit1 = img.select(['b4', 'b1']).reduceRegion({
        reducer: ee.Reducer.linearFit(),
        geometry: sunglint,
        scale: 3,
        maxPixels: 1e12,
        bestEffort: true,
    });
    var linearFit2 = img.select(['b4', 'b2']).reduceRegion({
        reducer: ee.Reducer.linearFit(),
        geometry: sunglint,
        scale: 3,
        maxPixels: 1e12,
        bestEffort: true,
    });
    var linearFit3 = img.select(['b4', 'b3']).reduceRegion({
        reducer: ee.Reducer.linearFit(),
        geometry: sunglint,
        scale: 3,
        maxPixels: 1e12,
        bestEffort: true,
    });

    var slopeImage = ee.Dictionary({
```

```
        'b1': linearFit1.get('scale'),
        'b2': linearFit2.get('scale'),
        'b3': linearFit3.get('scale')
    }).toImage();

    var minNIR = img.select('b4').reduceRegion({
        reducer: ee.Reducer.min(),
        geometry: sunglint,
        scale: 3,
        maxPixels: 1e12,
        bestEffort: true,
    }).toImage(['b4']);

    return img.select(['b1', 'b2', 'b3'])
        .subtract(slopeImage.multiply((img.select('b4')).subtract(
            minNIR)))
        .addBands(img.select('b4'));
}
var sgImg = sunglintRemoval(maskedImg);
Map.addLayer(sgImg, visParams, 'sgImg', false);
```

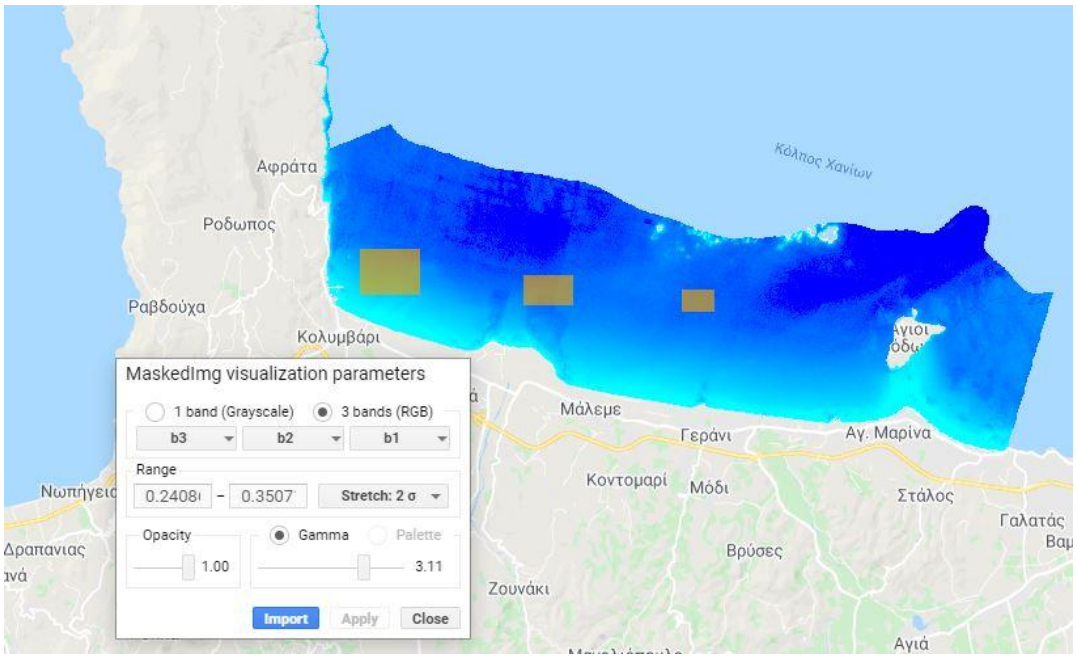


Fig. A2.2.2 Identification of areas with sunglint. Raising the gamma parameter during the visualization makes the phenomenon appear more intense, making it easier to draw the glint polygons.

Question 1. If you design more polygons for sunglint correction, can you see any improvement in the results visually and by examining values in pixels? Keep in mind that in already drawn rectangles (sunglint polygons), you can not draw free shaped polygons or lines. Try instead adding some more points or rectangles

You can design more polygons, or select areas with severe or moderate sunglint to see how the site selection influences the final results.

The Depth Invariant Index (DIV) is a tool that creates a proxy image that helps minimize the bias of the spectral values due to the water column during classification and bathymetry procedures. Spectral signatures tend to be affected by the depth of the water column due to suspended material and the absorption of light. A typical result is that shallow seagrasses and deeper sandy seafloors will have similar spectral signatures. The correction of that error is based on the correlation between depth and logged bands (Lyzenga 1981).

In our example, the correction of water column alterations is based on the ratio of the green and blue bands, because of their higher penetrating properties compared to the red and near-infrared (NIR) bands. As in the previous functions, a requirement for this procedure is to identify sandy patches in different depth ranges. If needed, you can use the satellite layer to do so.

Since we will use log values in the current step, it is crucial to transform all the negative values to positive before estimating DIV. Therefore, and since the majority of the values are in optically deep waters, the value 0.0001 will be assigned to values less than 0 in an attempt to avoid altering the pixels with positive values at the coastal zone. Moreover, a low-pass filter will be applied to normalize the observed noise that occurred during the previous steps (see Sect. F3.2).

```
// DIV procedure.
function kernel(img) {
    var boxcar = ee.Kernel.square({
```

```
        radius: 2,
        units: 'pixels',
        normalize: true
    });
    return img.convolve(boxcar);
}

function makePositive(img) {
    return img.where(img.lte(0), 0.0001);
}

function div(img) {
    var band1 = ee.List(['b1', 'b2', 'b3', 'b1', 'b2']);
    var band2 = ee.List(['b3', 'b3', 'b2', 'b2', 'b1']);
    var nband = ee.List(['b1b3', 'b2b3', 'b3b2', 'b1b2', 'b2b1']);

    for (var i = 0; i < 5; i += 1) {
        var x = band1.get(i);
        var y = band2.get(i);
        var z = nband.get(i);

        var imageLog = img.select([x, y]).log();

        var covariance = imageLog.toArray().reduceRegion({
            reducer: ee.Reducer.covariance(),
            geometry: DIVsand,
            scale: 3,
            maxPixels: 1e12,
            bestEffort: true,
        });

        var covarMatrix = ee.Array(covariance.get('array'));
        var var1 = covarMatrix.get([0, 0]);
        var var2 = covarMatrix.get([1, 1]);
        var covar = covarMatrix.get([0, 1]);

        var a = var1.subtract(var2).divide(covar.multiply(2));
```

```
        var attenCoeffRatio = a.add(((a.pow(2)).add(1)).sqrt());

        var depthInvariantIndex = img.expression(
            'image1 - (image2 * coeff)', {
                'image1': imageLog.select([x]),
                'image2': imageLog.select([y]),
                'coeff': attenCoeffRatio
            });

        img = ee.Image.cat([img, depthInvariantIndex.select([x], [
            z
        ])]);
    }
    return img;
}

var divImg = div(kernel(makePositive(sgImg))).select('b[1-3]',
    'b1b2');
var vivVisParams = {
    bands: ['b1b2'],
    min: -0.81,
    max: -0.04,
    gamma: 0.75
};
Map.addLayer(divImg, vivVisParams, 'divImg', false);
```

Question 2. How can the selection of sandy patches influence the final result?

Sandy patches can also include signals from sparse seagrass or other types of benthic cover. Select different areas of variable depths and therefore different spectral reflectance to explore the changes in the final DIV result.

Code Checkpoint A22c. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Supervised Classification

To create accurate classifications, it is beneficial for the training data set to be created from in situ data. This is especially important in marine remote sensing, due to dynamic, varying ecosystems. In our example, all the reference data were acquired during scientific dives. The reference data consist of three classes: SoftBottom for Sandy patches, rockyBottom for rocky patches and pO for posidonia-seagrass patches. Here, we provide a dataset that will be split using a 70%-30% partitioning strategy into training and validation data (see Sect. F2.1) and is already pre-loaded in the assets of this book.

```
// Section 3, classification
// Import of reference data and split.
var softBottom = ee.FeatureCollection(
    'projects/gee-book/assets/A2-2/SoftBottom');
var rockyBottom = ee.FeatureCollection(
    'projects/gee-book/assets/A2-2/RockyBottom');
var pO = ee.FeatureCollection('projects/gee-book/assets/A2-2/PO');

var sand = ee.FeatureCollection.randomPoints(softBottom, 150).map(
    function(s) {
        return s.set('class', 0);
    }).randomColumn();
var sandT = sand.filter(ee.Filter.lte('random', 0.7)).aside(print,
    'sand training');
var sandV = sand.filter(ee.Filter.gt('random', 0.7)).aside(print,
    'sand validation');
Map.addLayer(sandT, {
    color: 'yellow'
}, 'Sand Training', false);
Map.addLayer(sandV, {
    color: 'yellow'
}, 'Sand Validation', false);

var hard = ee.FeatureCollection.randomPoints(rockyBottom, 79).map(
    function(s) {
        return s.set('class', 1);
    }).randomColumn();
var hardT = hard.filter(ee.Filter.lte('random', 0.7)).aside(print,
```

```
    'hard training');
var hardV = hard.filter(ee.Filter.gt('random', 0.7)).aside(print,
    'hard validation');
Map.addLayer(hardT, {
    color: 'red'
}, 'Rock Training', false);
Map.addLayer(hardV, {
    color: 'red'
}, 'Rock Validation', false);

var posi = pO.map(function(s) {
        return s.set('class', 2);
    })
    .randomColumn('random');
var posiT = posi.filter(ee.Filter.lte('random', 0.7)).aside(print,
    'posi training');
var posiV = posi.filter(ee.Filter.gt('random', 0.7)).aside(print,
    'posi validation');
Map.addLayer(posiT, {
    color: 'green'
}, 'Posidonia Training', false);
Map.addLayer(posiV, {
    color: 'green'
}, 'Posidonia Validation', false);
```

For this procedure, we chose the `ee.Classifier.libsvm` classifier because of its established performance in aquatic environments (Poursanidis et al. 2018, da Silveira et al. 2021). The function below does the classification and also estimates overall, user's, and producer's accuracy (see Chap. F2.2):

```
// Classification procedure.
function classify(img) {
    var mergedT = ee.FeatureCollection([sandT, hardT, posiT])
    .flatten();
    var training = img.sampleRegions(mergedT, ['class'], 3);
```

```
var trained = ee.Classifier.libsvm({
    kernelType: 'RBF',
    gamma: 1,
    cost: 500
}).train(training, 'class');
var classified = img.classify(trained);

var mergedV = ee.FeatureCollection([sandV, hardV, posiV])
    .flatten();
var accuracyCol = classified.unmask().reduceRegions({
    collection: mergedV,
    reducer: ee.Reducer.first(),
    scale: 10
});
var classificationErrorMatrix = accuracyCol.errorMatrix({
    actual: 'class',
    predicted: 'first',
    order: [0, 1, 2]
});
var classNames = ['soft_bot', 'hard_bot', 'seagrass'];
var accuracyOA = classificationErrorMatrix.accuracy();
var accuraccyCons = ee.Dictionary.fromLists({
    keys: classNames,
    values: classificationErrorMatrix.consumersAccuracy()
        .toList()
        .flatten()
});
var accuracyProd = ee.Dictionary.fromLists({
    keys: classNames,
    values: classificationErrorMatrix.producersAccuracy()
        .toList()
        .flatten()
});

var classificationErrormatrixArray = classificationErrorMatrix
    .array();
```

```
var arrayToDatatable = function(array) {
    var classesNames = ee.List(classNames);

    function toTableColumns(s) {
        return {
            id: s,
            label: s,
            type: 'number'
        };
    }
    var columns = classesNames.map(toTableColumns);

    function featureToTableRow(f) {
        return {
            c: ee.List(f).map(function(c) {
                return {
                    v: c
                };
            })
        };
    }
    var rows = array.toList().map(featureToTableRow);
    return ee.Dictionary({
        cols: columns,
        rows: rows
    });
};

var dataTable = arrayToDatatable(classificationErrormatrixArray)
    .evaluate(function(dataTable) {
        print('------------- Error matrix -------------',
            ui.Chart(dataTable, 'Table')
            .setOptions({
                pageSize: 15
            }),
            'rows: reference, cols: mapped');
    });
```

```
    print('Overall Accuracy', accuracyOA);
    print('Users accuracy', accuraccyCons);
    print('Producers accuracy', accuracyProd);
    return classified;
}

var svmClassification = classify(divImg);
var svmVis = {
    min: 0,
    max: 2,
    palette: ['ffffbf', 'fc8d59', '91cf60']
};
Map.addLayer(svmClassification, svmVis, 'classification');
```

Code Checkpoint A22d. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Bathymetry by Random Forests regression

For the bathymetry procedure, we will exploit the `setOutputMode('REGRESSION')` option of `ee.Classifier.smileRandomForest`. For this example, reference data came from a sonar that was mounted on a boat. In contrast to the classification accuracy assessment, the accuracy assessment of bathymetry is based on R^2 and the root-mean-square error (RMSE).

With regard to visualization of the resulting bathymetry, we have to consider the selection of colors and their physical meanings. In the classification, which is a categorical image, we use a diverging palette, while in bathymetry, which shows a continuous value, we should use a sequential palette. Tip: "cold" colors better convey depth. For the satellite derived bathymetry we use preloaded assets with the in-situ depth measurement. The quality of these measurements is crucial to the success of the classifier.

```
// Section 4, Bathymetry
// Import and split training and validation data for the bathymetry.
```

```
var depth = ee.FeatureCollection(
    'projects/gee-book/assets/A2-2/DepthDataTill09072020_v2');
depth = depth.randomColumn();
var depthT = depth.filter(ee.Filter.lte('random', 0.7));
var depthV = depth.filter(ee.Filter.gt('random', 0.7));
Map.addLayer(depthT, {
    color: 'black'
}, 'Depth Training', false);
Map.addLayer(depthV, {
    color: 'gray'
}, 'Depth Validation', false);
```

So that every pixel contains at most one measurement, the vector depth assets are rasterized prior to using them for the regression.

```
function vector2image(vector) {
    var rasterisedVectorData = vector
        .filter(ee.Filter.neq('Depth',
            null)) // Filter out NA depth values.
        .reduceToImage({
            properties: ['Depth'],
            reducer: ee.Reducer.mean()
        });
    return (rasterisedVectorData);
}

var depthTImage = vector2image(depthT)
    .aside(Map.addLayer, {
        color: 'white'
    }, 'Depth Training2', false);
var depthVImage = vector2image(depthV)
    .aside(Map.addLayer, {
        color: 'white'
    }, 'Depth Validation2', false);
```

Finally, we need to write down and execute the function to calculate the satellite derived bathymetry function, based on the Random Forest classifier.

```
function rfbathymetry(img) {
    var training = img.sampleRegions({
        collection: depthT,
        scale: 3
    });

    var regclass = ee.Classifier.smileRandomForest(15)
        .train(training, 'Depth');
    var bathyClass = img
        .classify(regclass.setOutputMode('REGRESSION')).rename(
            'Depth');
    var sdbEstimate = bathyClass.clip(depthV);

    // Prepare data by putting SDB estimated data and in situ data
    // in one image to compare them afterwards.
    var imageI = ee.Image.cat([sdbEstimate, depthVImage]);
    // Calculate covariance.
    var covariance = imageI.toArray().reduceRegion({
        reducer: ee.Reducer.covariance(),
        geometry: depthV,
        scale: 3,
        bestEffort: true,
        maxPixels: 1e9
    });
    var covarMatrix = ee.Array(covariance.get('array'));
    var rSqr = covarMatrix.get([0, 1]).pow(2)
        .divide(covarMatrix.get([0, 0])
            .multiply(covarMatrix.get([1, 1])));
    var deviation = depthVImage.select('mean')
        .subtract(sdbEstimate.select('Depth')).pow(2);
    var rmse = ee.Number(deviation.reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: depthV,
            scale: 3,
            bestEffort: true,
            maxPixels: 1e12
```

```
        }).get('mean'))
        .sqrt();

    // Print together, so that they appear in the same output.
    print('R²', rSqr, 'RMSE', rmse);
    return bathyClass;
}

var rfBathymetry = rfbathymetry(divImg);
var bathyVis = {
    min: -50,
    max: 0,
    palette: ['084594', '2171b5', '4292c6', '6baed6',
        '9ecae1', 'c6dbef', 'deebf7', 'f7fbff'
    ]
};
Map.addLayer(rfBathymetry, bathyVis, 'bathymetry');
```

Question 3. Does the selection of different color ramps lead to misinterpretations? By choosing different color ramps for the same data set, you can see how visual interpretation can be changed based on color. Trying different color ramps will reveal how some can better visualize the final results for further use in maritime spatial planning activities, while others, including commonly seen rainbow ramps, can lead to erroneous decisions.

Code Checkpoint A22e. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

With what you learned in this chapter, you can analyze Earth observation data—here, specifically from the Planet Cubesat constellation—to create your own map of coastal benthic habitats and coastal bathymetry for a specific case study. Feel free to test out the approach in another part of the world using your own data or open-access data, or use your own training data for a more refined classification model.

You can add your own point data to the map, collected via a fieldwork campaign or by visually interpreting the imagery, and merge with the training data to improve a

classification, or clean areas that need to be removed by drawing polygons and masking them in the classification.

For the bathymetry, you can select different calibration/validation ratio approaches and test the optimum ratio of splitting to see how it influences the final bathymetry map. You can also add a smoothing filter to create a visually smoother image of coastal bathymetry.

Conclusion

Many coastal habitats, especially seagrass meadows, are dynamic ecosystems that change over time and can be lost through natural and anthropogenic causes.

The power of Earth Engine lies in its cloud-based, lightning-fast, automated approach to workflows, especially its automated training data collection and processing power. This process would take days when performed offline in traditional remote-sensing software, especially over large areas. And the Earth Engine approach is not only fast but also consistent: The same method can be applied to images from different dates to assess habitat changes over time, both gain and loss.

The availability of Planet imagery allows us to use a high-resolution product. Natively, Earth Engine hosts the archives of Sentinel-2 and Landsat data. The Landsat archive spans from 1984 to the present, while Sentinel-2, a higher-resolution product, is available from 2017 to today. All of this imagery can be used in the same workflow in order to map benthic habitats and monitor their changes over time, allowing us to understand the past of the coastal seascape and envision its future.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Borfecchia F, Micheli C, Carli F, et al (2013) Mapping spatial patterns of Posidonia oceanica meadows by means of Daedalus ATM airborne sensor in the coastal area of Civitavecchia (Central Tyrrhenian Sea, Italy). Remote Sens 5:4877–4899. https://doi.org/10.3390/rs5104877

Boudouresque CF, Bernard G, Pergent G, et al (2009) Regression of Mediterranean seagrasses caused by natural processes and anthropogenic disturbances and stress: A critical review. Bot Mar 52:395–418. https://doi.org/10.1515/BOT.2009.057

Boudouresque CF, Bernard G, Bonhomme P, et al (2012) Protection and Conservation of Posidonia oceanica Meadows. RAMOGE and RAC/SPA

Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010933404324

Campagne CS, Salles JM, Boissery P, Deter J (2014) The seagrass Posidonia oceanica: Ecosystem services identification and economic evaluation of goods and benefits. Mar Pollut Bull 97:391–400. https://doi.org/10.1016/j.marpolbul.2015.05.061

da Silveira CBL, Strenzel GMR, Maida M, et al (2021) Coral reef mapping with remote sensing and machine learning: A nurture and nature analysis in marine protected areas. Remote Sens 13:2907. https://doi.org/10.3390/rs13152907

Duarte CM, Kennedy H, Marbà N, Hendriks I (2013) Assessing the capacity of seagrass meadows for carbon burial: Current limitations and future strategies. Ocean Coast Manag 83:32–38. https://doi.org/10.1016/j.ocecoaman.2011.09.001

Eugenio F, Marcello J, Martin J (2015) High-resolution maps of bathymetry and benthic habitats in shallow-water environments using multispectral remote sensing imagery. IEEE Trans Geosci Remote Sens 53:3539–3549. https://doi.org/10.1109/TGRS.2014.2377300

Gao BC (1996) NDWI - A normalized difference water index for remote sensing of vegetation liquid water from space. Remote Sens Environ 58:257–266. https://doi.org/10.1016/S0034-4257(96)00067-3

Goodman J, Purkis S, Phinn SR (2011) Coral Reef Remote Sensing: A Guide for Mapping, Monitoring and Management. Springer

Hedley JD, Harborne AR, Mumby PJ (2005) Simple and robust removal of sun glint for mapping shallow-water benthos. Int J Remote Sens 26:2107–2112. https://doi.org/10.1080/01431160500034086

Hedley JD, Roelfsema CM, Chollett I, et al (2016) Remote sensing of coral reefs for monitoring and management: A review. Remote Sens 8:118. https://doi.org/10.3390/rs8020118

Knudby A, Nordlund L (2011) Remote sensing of seagrasses in a patchy multi-species environment. Int J Remote Sens 32:2227–2244. https://doi.org/10.1080/01431161003692057

Koedsin W, Intararuang W, Ritchie RJ, Huete A (2016) An integrated field and remote sensing method for mapping seagrass species, cover, and biomass in Southern Thailand. Remote Sens 8:292. https://doi.org/10.3390/rs8040292

Lyzenga DR (1981) Remote sensing of bottom reflectance and water attenuation parameters in shallow water using aircraft and Landsat data. Int J Remote Sens 2:71–82. https://doi.org/10.1080/01431168108948342

Pergent G, Bazairi H, Bianchi CN, et al (2014) Climate change and Mediterranean seagrass meadows: A synopsis for environmental managers. Mediterr Mar Sci 15:462–473. https://doi.org/10.12681/mms.621

Poursanidis D, Topouzelis K, Chrysoulakis N (2018) Mapping coastal marine habitats and delineating the deep limits of the Neptune's seagrass meadows using very high resolution Earth observation data. Int J Remote Sens 39:8670–8687. https://doi.org/10.1080/01431161.2018.1490974

UNEP/MAP (2009) State of the Mediterranean marine and coastal environment. In: Ecological Applications. pp 1047–1056

Chapter A2.3: Surface Water Mapping

Authors
K. Markert, G. Donchyts, A. Haag

Overview

In this chapter, you will learn the step-by-step implementation of an efficient and robust approach for mapping surface water. You will also learn how the extracted surface water information can be used in conjunction with historical surface water information to extract flooded areas. This chapter will focus mostly on the use of Sentinel-1 Synthetic Aperture Radar (SAR) data, but the approaches apply to both SAR and optical remotely sensed data.

Learning Outcomes

- Applying Otsu thresholding techniques for surface water mapping.
- Understanding the considerations of global versus adaptive histogram sampling.
- Implementing an adaptive histogram sampling approach.
- Extracting flooded areas from a surface water map.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Create a graph using `ui.Chart` (Chap. F1.3).
- Write a function and `map` it over an ImageCollection (Chap. F4.0).
- Understand basics of working with Synthetic Aperture Radar images (Chap. A1.8).

Introduction to Theory

Flooding impacts more people than any other environmental hazard, and flood exposure is expected to increase in the future (Tellman et al. 2021). Remote sensing data plays a pivotal role in mapping historical flood zones and producing spatial maps of flood events that can be used to guide response efforts (Oddo and Bolten 2019). Oftentimes, flood maps need to be created and delivered to disaster managers within hours of image acquisition. Thus, computationally efficient approaches are required to reduce latency.

Furthermore, these approaches need to produce accurate results without increasing processing time.

Image thresholding is an efficient method for mapping surface water (Schumann et al. 2009). Among the numerous methods available for image thresholding, a popular one is "Otsu's method" (Otsu 1979). Otsu's method is a histogram-based thresholding approach where the inter-class variance between two classes, a foreground class and a background class, is maximized. As this approach assumes that only two classes are present within an image, which is rarely the case, methods have been developed (Donchyts et al. 2016, Cao et al. 2019) to constrain histogram sampling to areas that are more likely to represent a bimodal histogram of water/no water. Otsu's method applied on such a constrained histogram provides a more accurate estimation of a water threshold without sacrificing the computational efficiency of the method.

This chapter explores surface water mapping using Otsu's method, and walks through an adaptive thresholding technique initially developed by Donchyts et al. 2016 and applied on optical imagery, then adapted by Markert et al. 2020 for detecting surface water in SAR imagery. Furthermore, the resulting surface water map will be compared with the Joint Research Centre's (JRC) Global Surface Water dataset (Pekel et al. 2016) to extract the flooded areas. This chapter will focus on the use of Sentinel-1 (S1) SAR data, but the concepts apply to other satellite imagery where we can distinguish water.

Practicum

Section 1. Otsu Thresholding

Otsu's method is widely used for determining the optimal threshold of an image with two classes. In this section, we will explore the use of Otsu's method for segmenting water using an image-wide histogram (also known as a global histogram).

We will start by accessing Sentinel-1 data. We will focus our analysis on Southeast Asia, which experiences yearly flooding and so provides plenty of good test cases. To do this, we will assign the Sentinel-1 collection to a variable and filter by space, time, and metadata properties to get our image for processing.

```
// Define a point in Cambodia to filter by location.
var point = ee.Geometry.Point(104.9632, 11.7686);
```

```
Map.centerObject(point, 11);

// Get the Sentinel-1 collection and filter by space/time.
var s1Collection = ee.ImageCollection('COPERNICUS/S1_GRD')
    .filterBounds(point)
    .filterDate('2019-10-05', '2019-10-06')
    .filter(ee.Filter.eq('orbitProperties_pass', 'ASCENDING'))
    .filter(ee.Filter.eq('instrumentMode', 'IW'));

// Grab the first image in the collection.
var s1Image = s1Collection.first();
```

Now we can add our image to the map.

```
// Add the Sentinel-1 image to the map.
Map.addLayer(s1Image, {
    min: -25,
    max: 0,
    bands: 'VV'
}, 'Sentinel-1 image');
```

The map should now have a Sentinel-1 image in Cambodia that looks like Fig. A2.3.1.

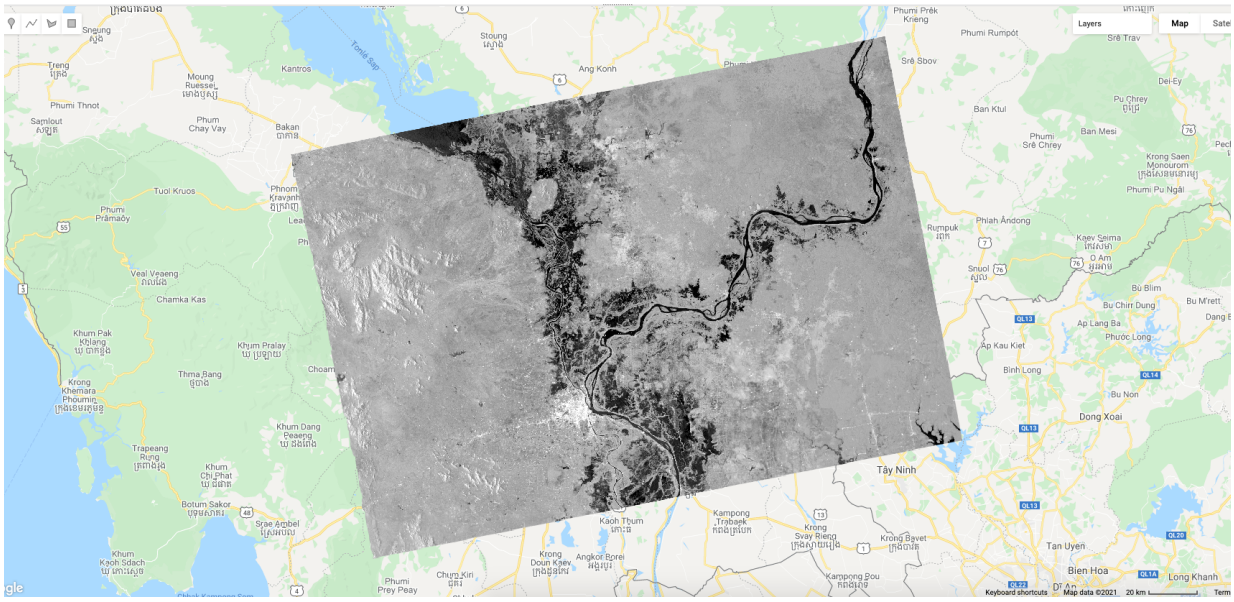


Fig. A2.3.1 Sentinel-1 VV image from October 5, 2019 highlighting a flooding event in Cambodia

Now that we have our image, we can begin our processing to extract surface water information using Otsu's threshold. Otsu's thresholding algorithm uses histograms, so we will need to create one for the pixels of the image. Otsu's method works on a single band of values, so we will use the VV band from Sentinel-1, as seen in Fig. A2.3.1. Earth Engine allows us to easily calculate a histogram using the reducer `ee.Reducer.histogram` applied across the image using the `reduceRegion` operation:

```
// Specify band to use for Otsu thresholding.
var band = 'VV';

// Define a reducer to calculate a histogram of values.
var histogramReducer = ee.Reducer.histogram(255, 0.1);

// Reduce all of the image values.
var globalHistogram = ee.Dictionary(
    s1Image.select(band).reduceRegion({
        reducer: histogramReducer,
        geometry: s1Image.geometry(),
        scale: 90,
```

```
        maxPixels: 1e10
    }).get(band)
);

// Extract out the histogram buckets and counts per bucket.
var x = ee.List(globalHistogram.get('bucketMeans'));
var y = ee.List(globalHistogram.get('histogram'));

// Define a list of values to plot.
var dataCol = ee.Array.cat([x, y], 1).toList();

// Define the header information for data.
var columnHeader = ee.List([
    [
    {
        label: 'Backscatter',
        role: 'domain',
        type: 'number'
    },
    {
        label: 'Values',
        role: 'data',
        type: 'number'
    }, ]
]);

// Concat the header and data for plotting.
var dataTable = columnHeader.cat(dataCol);

// Create plot using the ui.Chart function with the dataTable.
// Use 'evaluate' to transfer the server-side table to the client.
// Define the chart and print it to the console.
dataTable.evaluate(function(dataTableClient) {
    var chart = ui.Chart(dataTableClient)
        .setChartType('AreaChart')
        .setOptions({
            title: band + ' Global Histogram',
```

```
            hAxis: {
                title: 'Backscatter [dB]',
                viewWindow: {
                    min: -35,
                    max: 15
                }
            },
            vAxis: {
                title: 'Count'
            }
        });
    print(chart);
});
```

After running that code, you should see the chart in Fig. A2.3.2 printed in the **Console**. This is the histogram of values across the image.

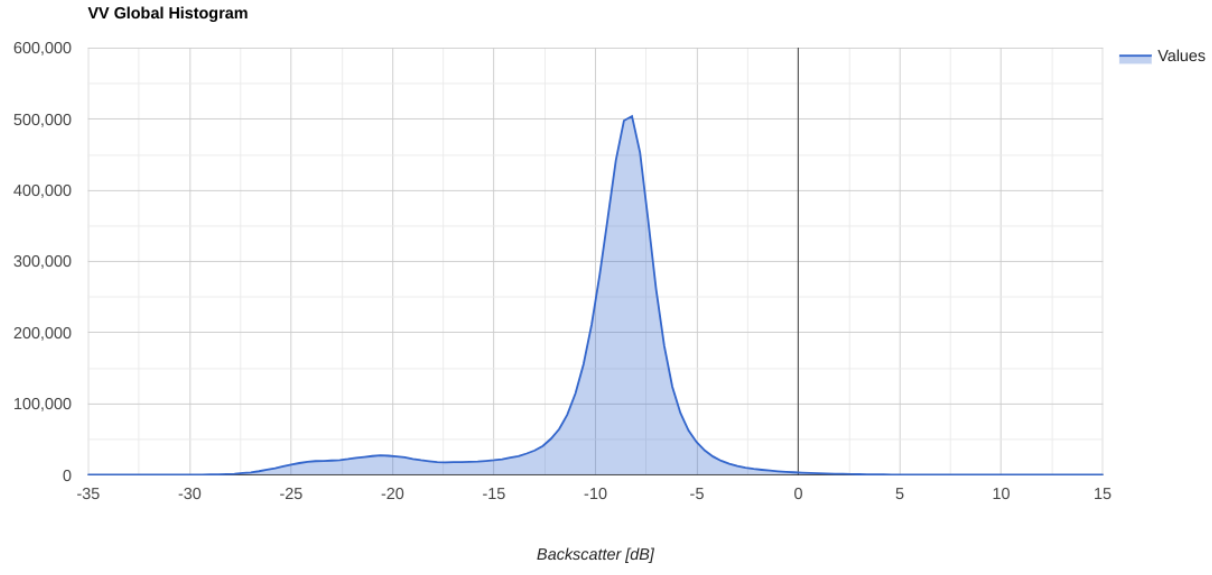


Fig. A2.3.2 Histogram of values from the Sentinel-1 VV image from October 5, 2019

In the histogram, the data points are heavily concentrated around −9 dB. We can see a small peak of low backscatter values around −22 dB; these are the likely open-water

values. The SAR signal bounces off large, smooth surfaces like water, so likely open-water values are low, and non-water values are high.

How should we split the histogram into two parts to create a high-quality partition of the image into two classes? That is the job of Otsu's thresholding algorithm. Earth Engine does not have a built-in function for Otsu's method, so we will create a function that implements the algorithm's logic. This function takes in a histogram as input, applies the algorithm, and returns a single value where Otsu's method suggests breaking the histogram into two parts. (More information about Otsu's method can be found in the "For Further Reading" section of this book.)

```
function otsu(histogram) {
    // Make sure histogram is an ee.Dictionary object.
    histogram = ee.Dictionary(histogram);
    // Extract relevant values into arrays.
    var counts = ee.Array(histogram.get('histogram'));
    var means = ee.Array(histogram.get('bucketMeans'));
    // Calculate single statistics over arrays
    var size = means.length().get([0]);
    var total = counts.reduce(ee.Reducer.sum(), [0]).get([0]);
    var sum = means.multiply(counts).reduce(ee.Reducer.sum(), [0])
        .get([0]);
    var mean = sum.divide(total);
    // Compute between sum of squares, where each mean partitions the
data.
    var indices = ee.List.sequence(1, size);
    var bss = indices.map(function(i) {
        var aCounts = counts.slice(0, 0, i);
        var aCount = aCounts.reduce(ee.Reducer.sum(), [0])
            .get([0]);
        var aMeans = means.slice(0, 0, i);
        var aMean = aMeans.multiply(aCounts)
            .reduce(ee.Reducer.sum(), [0]).get([0])
            .divide(aCount);
        var bCount = total.subtract(aCount);
        var bMean = sum.subtract(aCount.multiply(aMean))
            .divide(bCount);
```

```
        return aCount.multiply(aMean.subtract(mean).pow(2))
            .add(
                bCount.multiply(bMean.subtract(mean).pow(2)));
    });
    // Return the mean value corresponding to the maximum BSS.
    return means.sort(bss).get([-1]);
}
```

When the threshold is calculated, it can be applied to the imagery and we can inspect where it falls within our histogram. The following code creates a new array of values and checks where the threshold is so that we can see how the algorithm performed.

```
// Apply otsu thresholding.
var globalThreshold = otsu(globalHistogram);
print('Global threshold value:', globalThreshold);

// Create list of empty strings that will be used for annotation.
var thresholdCol = ee.List.repeat('', x.length());
// Find the index where the bucketMean equals the threshold.
var threshIndex = x.indexOf(globalThreshold);
// Set the index to the annotation text.
thresholdCol = thresholdCol.set(threshIndex, 'Otsu Threshold');

// Redefine the column header information with annotation column.
columnHeader = ee.List([
    [
    {
        label: 'Backscatter',
        role: 'domain',
        type: 'number'
    },
    {
        label: 'Values',
        role: 'data',
        type: 'number'
    },
    {
```

```
        label: 'Threshold',
        role: 'annotation',
        type: 'string'
    }]
]);

// Loop through the data rows and add the annotation column.
dataCol = ee.List.sequence(0, x.length().subtract(1)).map(function(
i) {
    i = ee.Number(i);
    var row = ee.List(dataCol.get(i));
    return row.add(ee.String(thresholdCol.get(i)));
});

// Concat the header and data for plotting.
dataTable = columnHeader.cat(dataCol);

// Create plot using the ui.Chart function with the dataTable.
// Use 'evaluate' to transfer the server-side table to the client.
// Define the chart and print it to the console.
dataTable.evaluate(function(dataTableClient) {
    // loop through the client-side table and set empty strings to
null
    for (var i = 0; i < dataTableClient.length; i++) {
        if (dataTableClient[i][2] === '') {
            dataTableClient[i][2] = null;
        }
    }
    var chart = ui.Chart(dataTableClient)
        .setChartType('AreaChart')
        .setOptions({
            title: band +
                ' Global Histogram with Threshold annotation',
            hAxis: {
                title: 'Backscatter [dB]',
                viewWindow: {
                    min: -35,
```

```
                    max: 15
                }
            },
            vAxis: {
                title: 'Count'
            },
            annotations: {
                style: 'line'
            }
        });
    print(chart);
});
```

Once you have run the above code, you should see another histogram chart that looks like Fig. A2.3.3. Note the addition of the text and light vertical line indicating the location of the threshold value.

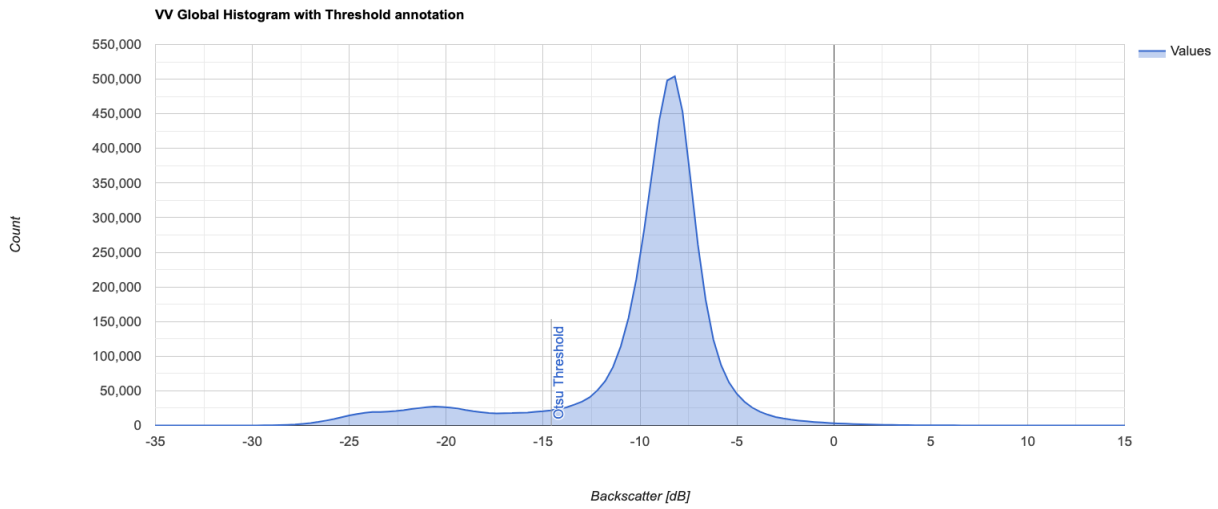


Fig. A2.3.3 Histogram of values from the Sentinel-1 VV image with the threshold calculated using Otsu's method

We can see that the threshold is around -15 dB, which is about halfway between the two peaks. We can now apply that threshold on the imagery and inspect how the extracted

water looks compared to the original image. Using the code below, we apply the threshold and add the water image to the map.

```
// Apply the threshold on the image to extract water.
var globalWater = s1Image.select(band).lt(globalThreshold);

// Add the water image to the map and mask 0 (no-water) values.
Map.addLayer(globalWater.selfMask(),
    {
        palette: 'blue'
    },
    'Water (global threshold)');
```

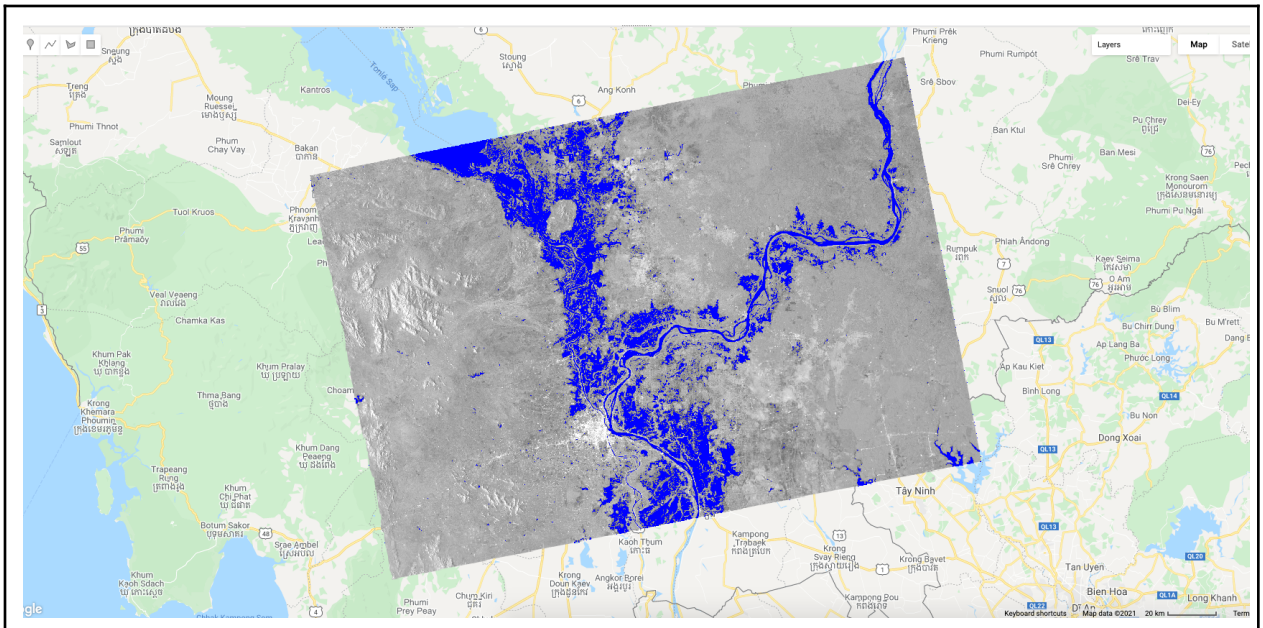


Fig. A2.3.4 Extracted surface water from Otsu's method for the Sentinel-1 VV image from October 5, 2019 highlighting a flooding event in Cambodia

The results look promising. The blue areas overlap with the low backscatter (specular reflectance) that is representative of open water in C-band SAR imagery. However, upon closer inspection we can see that the extracted water overestimates in some areas.

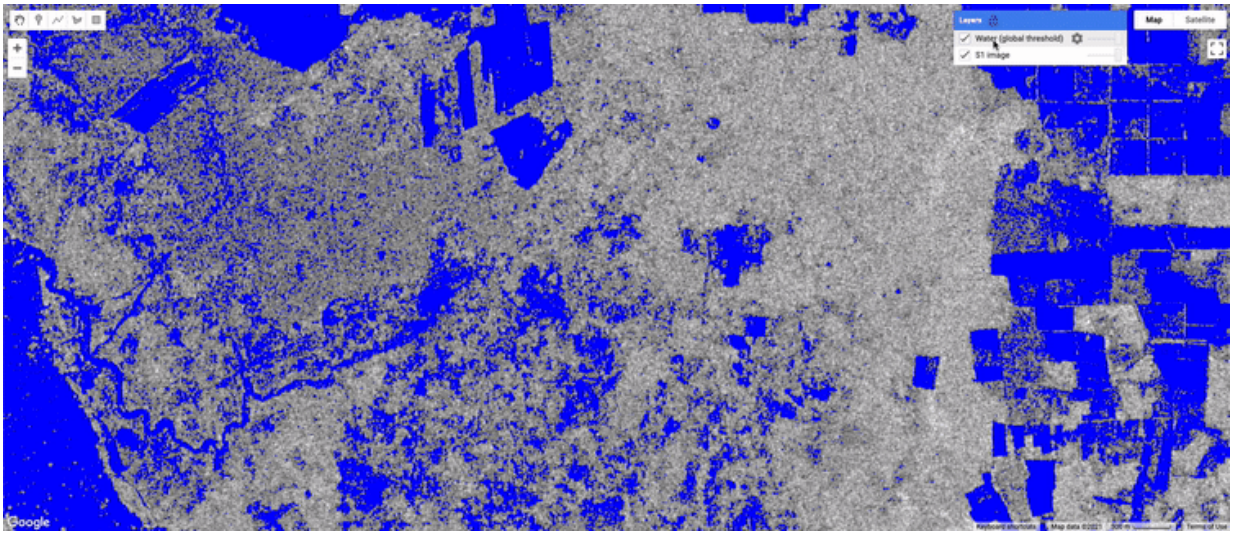


Fig. A2.3.5 Close-up inspection of extracted surface water. The extracted water is toggled on and off to illustrate where global Otsu thresholding overestimated water.

We see an overestimation as large local errors may be introduced when calculating a constant threshold for distinguishing water from land when using an image-wide histogram. It is due to this issue that algorithms have been developed to constrain the histogram sampling and estimate a more locally contextual threshold.

Code Checkpoint A23a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. Do some reading on Otsu's method (the Wikipedia page has a good description). How does Otsu's threshold work? What underlying assumptions does Otsu's threshold make?

Question 2. Based on the results and your understanding of Otsu's thresholding, why do you think the calculated histogram overestimated water areas?

Section 2. Adaptive Thresholding

Surface water usually constitutes only a small fraction of the overall land cover within an Earth observation image. This makes it harder to apply threshold-based methods to

extract water. The challenge is to establish a varying threshold that can be derived automatically. In images that show flooding, like the one from the previous section, this limitation is not so significant. Nevertheless, this section walks through an adaptive thresholding technique designed to overcome the challenges of using a global threshold.

The method we will discuss was developed by Donchyts et al. (2016) and applied to the Modified Normalized Difference Water Index (MNDWI) from Landsat 8 imagery. The algorithm finds edges within the image, buffers the areas around the identified edges, and uses the buffered area to sample a histogram for Otsu thresholding. This approach assumes that the edges detected are from water. The result is a bimodal histogram from the area around water edges that can be used to calculate a refined threshold. The overall workflow of the algorithm is shown in Fig. A2.3.6.

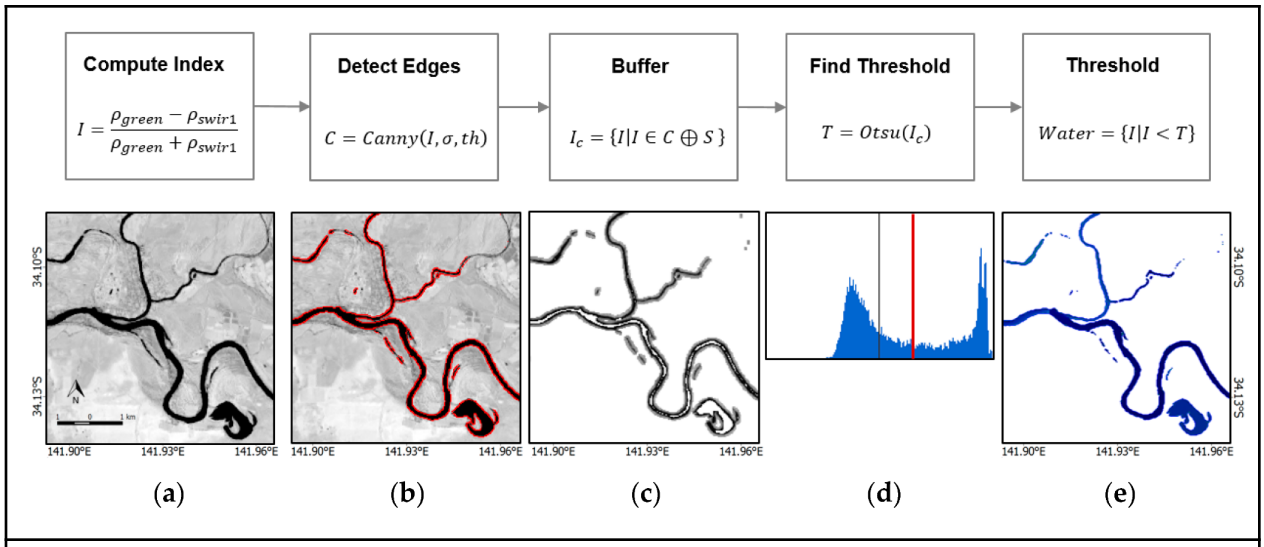


Fig. A2.3.6 Workflow of adaptive thresholding techniques. Figure taken from Donchyts et al. (2016) under the Creative Commons Attribution License.

This approach was refined by Markert et al. (2020), where the main change is that instead of calculating the edges on the raw values (from an index or otherwise), an initial segmentation threshold is provided to create a binary image as input for the edge detection. This overcomes issues with SAR speckle and other artifacts, as well as with any edges being defined from other classes that are present in imagery (e.g., urban areas or forests). The defined edges are then filtered by length to omit small edges that can occur and can skew the histogram sampling. This requires that a few parameters be tuned, namely the initial threshold, edge length, and buffer size. Here, we define a few of those parameters.

```
// Define parameters for the adaptive thresholding.
// Initial estimate of water/no-water for estimating the edges
var initialThreshold = -16;
// Number of connected pixels to use for length calculation.
var connectedPixels = 100;
// Length of edges to be considered water edges.
var edgeLength = 20;
// Buffer in meters to apply to edges.
var edgeBuffer = 300;
// Threshold for canny edge detection.
var cannyThreshold = 1;
// Sigma value for gaussian filter in canny edge detection.
var cannySigma = 1;
// Lower threshold for canny detection.
var cannyLt = 0.05;
```

With these parameters defined, we can begin the process of constraining the histogram sampling.

```
// Get preliminary water.
var binary = s1Image.select(band).lt(initialThreshold)
    .rename('binary');

// Get projection information to convert buffer size to pixels.
var imageProj = s1Image.select(band).projection();

// Get canny edges.
var canny = ee.Algorithms.CannyEdgeDetector({
    image: binary,
    threshold: cannyThreshold,
    sigma: cannySigma
});

// Process canny edges.

// Get the edges and length of edges.
```

```
var connected = canny.updateMask(canny).lt(cannyLt)
    .connectedPixelCount(connectedPixels, true);

// Mask short edges that can be noise.
var edges = connected.gte(edgeLength);

// Calculate the buffer in pixel size.
var edgeBufferPixel = ee.Number(edgeBuffer).divide(imageProj
    .nominalScale());

// Buffer the edges using a dilation operation.
var bufferedEdges = edges.fastDistanceTransform().lt(edgeBufferPixel);

// Mask areas not within the buffer .
var edgeImage = s1Image.select(band).updateMask(bufferedEdges);
```

Now that we have the edge information and the data to sample processed, we can visually inspect what the algorithm is doing. Here, we will display the calculated edges as well as the buffered edges to highlight which data is being sampled.

```
// Add the detected edges and buffered edges to the map.
Map.addLayer(edges, {
    palette: 'red'
}, 'Detected water edges');
var edgesVis = {
    palette: 'yellow',
    opacity: 0.5
};
Map.addLayer(bufferedEdges.selfMask(), edgesVis,
    'Buffered water edges');
```

You should now have the data added to the map, which should look like the images in Fig. 2.3.7 when zoomed in.

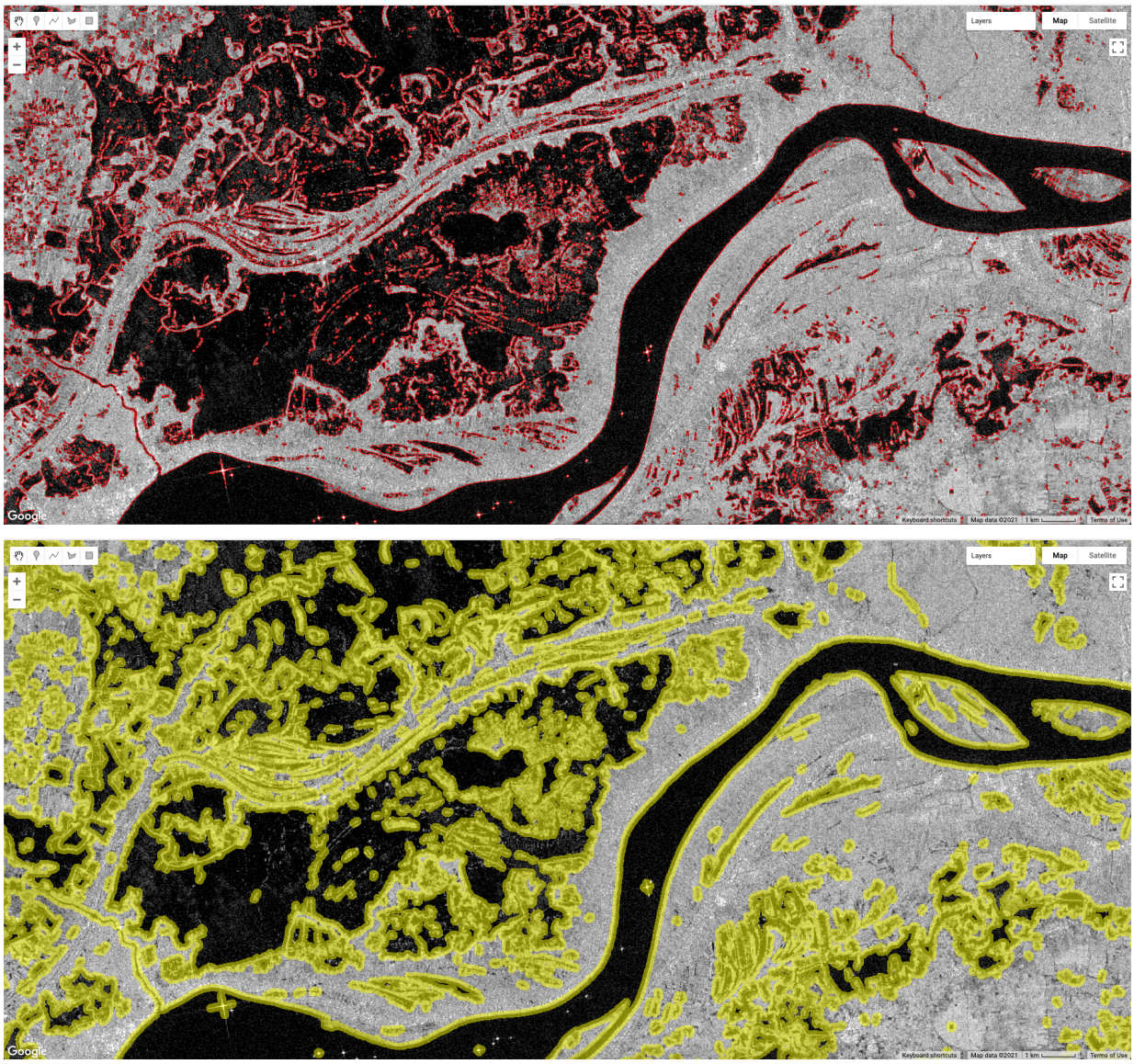


Fig. A2.3.7 Results from the water edge detection process, where the edges are shown in red (top image) and the buffered edges highlighting the sampling regions in yellow (bottom image)

At this point, we have our regions that we want to sample that are more representative of a bimodal histogram, and we have masked out areas that we don't want to sample. Now we can calculate the histogram as before and make a plot.

```
// Reduce all of the image values.
var localHistogram = ee.Dictionary(
    edgeImage.reduceRegion({
        reducer: histogramReducer,
        geometry: s1Image.geometry(),
        scale: 90,
        maxPixels: 1e10
    }).get(band)
);

// Apply otsu thresholding.
var localThreshold = otsu(localHistogram);
print('Adaptive threshold value:', localThreshold);

// Extract out the histogram buckets and counts per bucket.
var x = ee.List(localHistogram.get('bucketMeans'));
var y = ee.List(localHistogram.get('histogram'));

// Define a list of values to plot.
var dataCol = ee.Array.cat([x, y], 1).toList();

// Concat the header and data for plotting.
var dataTable = columnHeader.cat(dataCol);

// Create list of empty strings that will be used for annotation.
var thresholdCol = ee.List.repeat('', x.length());
// Find the index that bucketMean equals the threshold.
var threshIndex = x.indexOf(localThreshold);
// Set the index to the annotation text.
thresholdCol = thresholdCol.set(threshIndex, 'Otsu Threshold');

// Redefine the column header information now with annotation col.
columnHeader = ee.List([
    [
    {
        label: 'Backscatter',
        role: 'domain',
```

```
        type: 'number'
    },
    {
        label: 'Values',
        role: 'data',
        type: 'number'
    },
    {
        label: 'Threshold',
        role: 'annotation',
        type: 'string'
    }]
]);

// Loop through the data rows and add the annotation col.
dataCol = ee.List.sequence(0, x.length().subtract(1)).map(function(
i) {
    i = ee.Number(i);
    var row = ee.List(dataCol.get(i));
    return row.add(ee.String(thresholdCol.get(i)));
});

// Concat the header and data for plotting.
dataTable = columnHeader.cat(dataCol);

// Create plot using the ui.Chart function with the dataTable.
// Use 'evaluate' to transfer the server-side table to the client.
// Define the chart and print it to the console.
dataTable.evaluate(function(dataTableClient) {
    // Loop through the client-side table and set empty strings to
null.
    for (var i = 0; i < dataTableClient.length; i++) {
        if (dataTableClient[i][2] === '') {
            dataTableClient[i][2] = null;
        }
    }
    var chart = ui.Chart(dataTableClient)
```

```
        .setChartType('AreaChart')
        .setOptions({
            title: band +
                ' Adaptive Histogram with Threshold annotation',
            hAxis: {
                title: 'Backscatter [dB]',
                viewWindow: {
                    min: -35,
                    max: 15
                }
            },
            vAxis: {
                title: 'Count'
            },
            annotations: {
                style: 'line'
            }
        });
    print(chart);
});
```

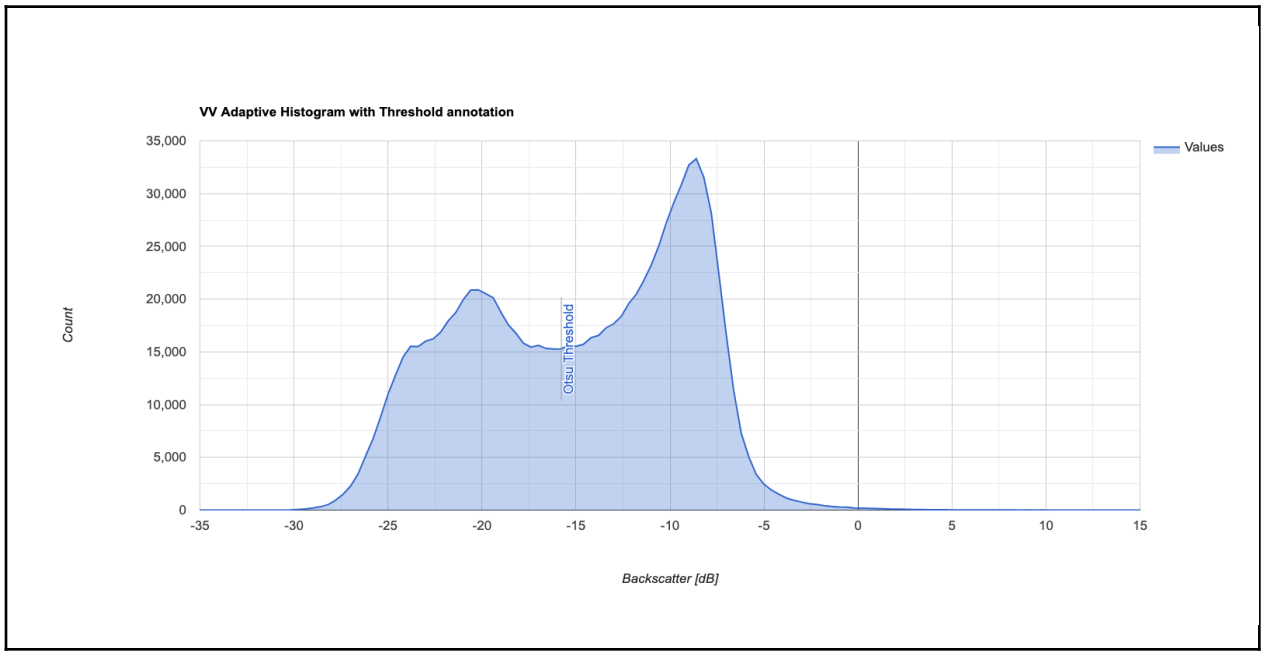


Fig. A2.3.8 Histogram of values from the Sentinel-1 VV image using the adaptive thresholding

We can see from the histogram that we have two distinct peaks. This meets the assumption of Otsu thresholding that only two classes are present within an image, which allows the algorithm to more accurately calculate the threshold for water. The last thing left to do is to apply the calculated adaptive threshold on the imagery and add it to the map.

```
// Apply the threshold on the image to extract water.
var localWater = s1Image.select(band).lt(localThreshold);

// Add the water image to the map and mask 0 (no-water) values.
Map.addLayer(localWater.selfMask(),
    {
        palette: 'darkblue'
    },
    'Water (adaptive threshold)');
```

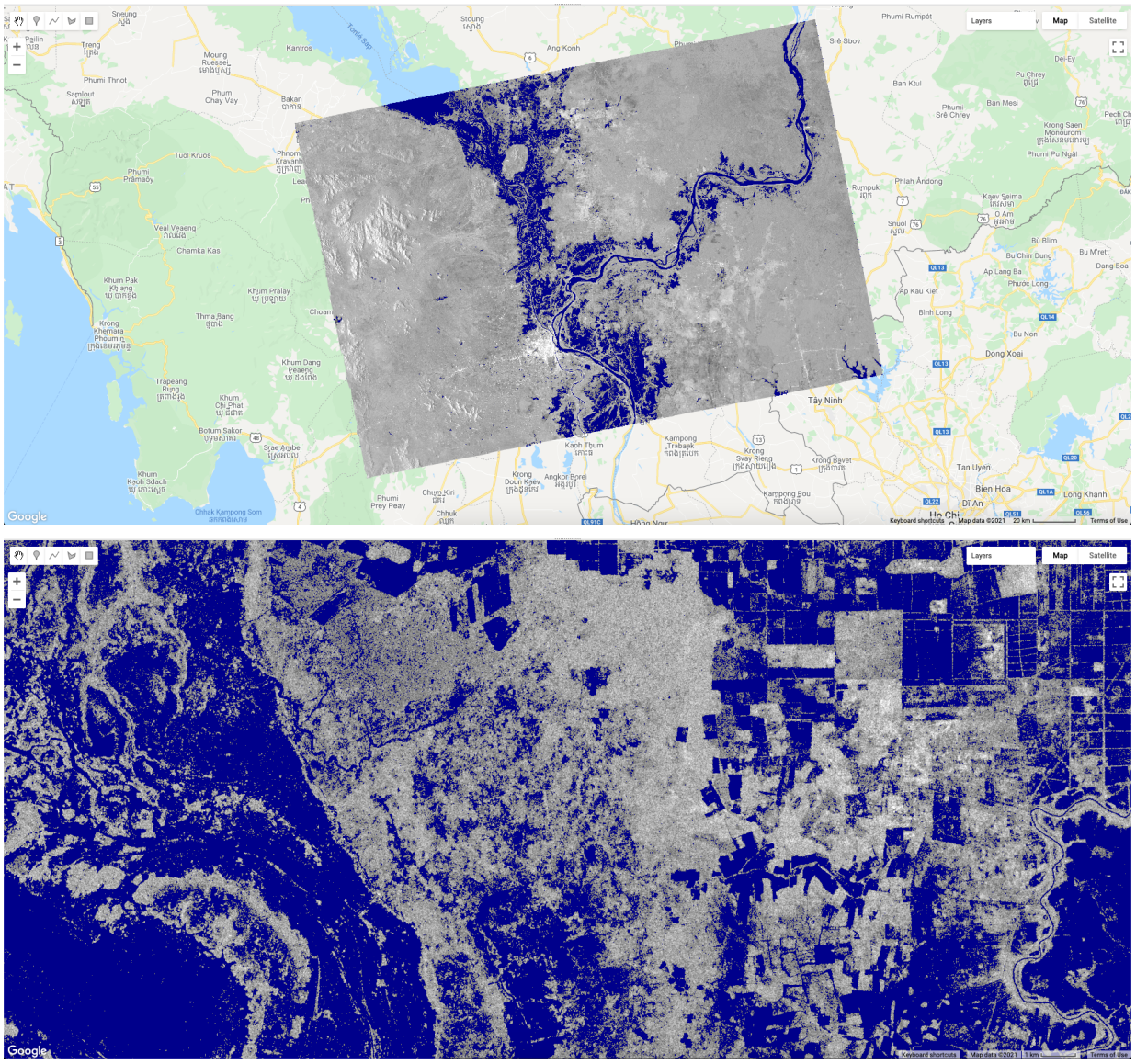


Fig. A2.3.9 Extracted surface water using the adaptive Otsu thresholding method for Sentinel-1 VV image (top image) and close-up inspection of extracted surface water for almost the same area as in Sect. 1 (bottom image)

It can be seen from the resulting images that the adaptive thresholding technique produces a reasonable surface water map. Furthermore, the resulting threshold value was −15.799, as compared to −14.598 from the global thresholding. A lower threshold in this case means less surface water area extracted. However, less surface water area

does not necessarily mean more accuracy. There needs to be a balance between producer's and user's accuracy.

Now that we have a surface water map and we are moderately confident that it represents the actual surface water for that day, we can begin to identify flooded areas by differencing our map with historical information.

Code Checkpoint A23b. The book's repository contains a script that shows what your code should look like at this point.

Question 3. Why do we apply an initial threshold to the SAR imagery prior to detecting edges? What happens to the detected edges if we do not apply an initial threshold? Explain why you are or are not getting a difference in detected edges. Recall that we defined some parameters for the Canny edge detection. Show a comparison.

Question 4. Compare the threshold calculated with the adaptive technique to the global threshold. In your own words, explain why the two thresholds are different.

Question 5. Change the parameters used for the adaptive thresholding to see how the results change. Which parameters is the algorithm most sensitive to?

Section 3. Extracting Flood Areas

Up to this point, we have been mapping surface water, which includes permanent and seasonal water that was observed by the sensor. What we need to do now is to identify areas from our image that are considered permanent water. There are typically two approaches to mapping flooded areas with a thematic surface water map: (1) comparing pre- and post-event images to estimate changes; or (2) comparing extracted surface water with historically observed permanent water.

To achieve the goal of flood mapping, we will use the historical JRC Global Surface Water dataset to define permanent water and then find the difference to extract flooded areas. We already have our post-event surface water map. Now we need to access and use the JRC data.

```
// Get the previous 5 years of permanent water.
```

```
// Get the JRC historical yearly dataset.
var jrc = ee.ImageCollection('JRC/GSW1_3/YearlyHistory')
    // Filter for historical data up to date of interest.
    .filterDate('1985-01-01', s1Image.date())
    // Grab the 5 latest images/years.
    .limit(5, 'system:time_start', false);
```

Because this data is a yearly classification of permanent and seasonal water, we need to reclassify the imagery to just permanent water.

```
var permanentWater = jrc.map(function(image) {
        // Extract out the permanent water class.
        return image.select('waterClass').eq(3);
        // Reduce the collection to get information on if a pixel has
        // been classified as permanent water in the past 5 years.
    }).sum()
    // Make sure we have a value everywhere.
    .unmask(0)
    // Get an image of 1 if permanent water in the past 5 years,
otherwise 0.
    .gt(0)
    // Mask for only the water image we just calculated.
    .updateMask(localWater.mask());

// Add the permanent water layer to the map.
Map.addLayer(permanentWater.selfMask(),
    {
        palette: 'royalblue'
    },
    'JRC permanent water');
```

The final thing we need to do is apply a simple differencing between the surface water map from Sentinel-1 and the JRC permanent water.

```
// Find areas where there is not permanent water, but water is
```

```
observed.
var floodImage = permanentWater.not().and(localWater);

// Add flood image to map.
Map.addLayer(floodImage.selfMask(), {
    palette: 'firebrick'
}, 'Flood areas');
```

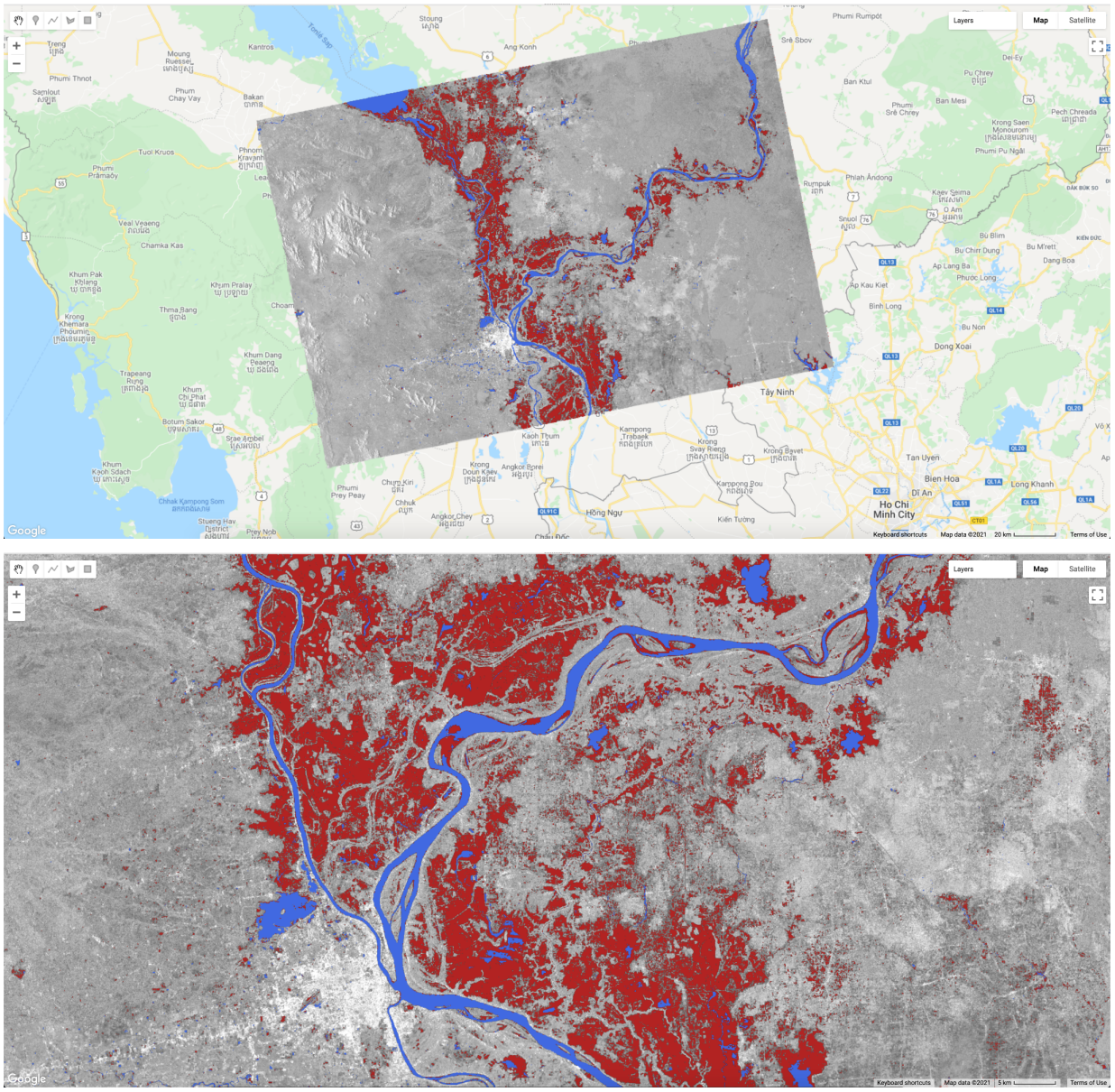


Fig. A2.3.10 Extracted flood areas by comparing the calculated surface water map against the JRC permanent water data. Bottom image shows a close-up of the flood map around the confluence of the Mekong and Tonle Sap rivers near Phnom Penh, Cambodia.

There are nuances associated with comparing optically derived water information (like from JRC) with SAR water maps. For example, any surface that is large enough and smooth can "look" like water in SAR imagery because of specular reflectance and can

be wrongly classified as a flooded area. Examples of this are airports, exposed channel beds, and highways. It should also be noted that there is a component of seasonal flooding—flooding that occurs every year and is expected. Currently, our flood map contains areas of seasonal flooding as well. Therefore, to accurately map the "abnormal" area of a flood, we'd also have to account for seasonal patterns. Lastly, rivers are in constant flux, changing patterns, and even change due to flooding events, so comparing historical observations with flooding events may yield some areas that have changed. Therefore, comparing pre- and post-event imagery from the same sensor is best. However, it is challenging to define events in seasonal flooding (such as this case), making a pre- and post-event comparison a little more complicated.

Code Checkpoint A23c. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

In this chapter, we covered a common image segmentation method, Otsu's thresholding, and applied it to map surface water using Sentinel-1 SAR imagery. Furthermore, we illustrated an image processing technique to constrain the histogram sampling for input into the Otsu thresholding method. Lastly, we created a flood map from the segmented surface water map using historical permanent surface water data. You should now have a good grasp on the Otsu thresholding technique for surface water mapping, understand the considerations of global versus localized histogram sampling, be able to implement an adaptive histogram sampling approach, and take a surface water map and convert it to a flood map.

Assignment 1. Identify a flooding event of interest (a good source is https://floodlist.com) and walk through the process of creating a flood map for the event you chose. Do you notice anything different with the resulting flood map? Make note of the identified threshold value. Does the threshold value represent water versus no-water areas? Keep in mind the physical properties of SAR—some areas naturally have low backscatter like water does.

Assignment 2. In your own words, describe the difference between a surface water map and a flood map. Conceptually, what do you need to take into consideration when extracting flood areas?

Assignment 3. Find a pre-event Sentinel-1 image for the case we have gone through, extract surface water from the pre-event image, and compare it to the post- event image. Do you find differences in the flood areas derived from pre/post comparison versus historical comparison?

Assignment 4. Refactor the code to make the adaptive thresholding algorithm and flood mapping into a callable function that can be mapped over an image collection.

Conclusion

It should be noted that Sentinel-1 SAR imagery ideally should undergo preprocessing to remove the effects of terrain and speckle in imagery, as in Mullissa et al. (2021), before applying surface water mapping algorithms. However, SAR preprocessing is outside the scope of this chapter.

There are many more sophisticated algorithms for mapping surface water from satellite imagery (such as Mayer et al. 2021). The application illustrated in this chapter is meant to highlight a practical workflow for mapping surface water and floods that can be implemented in Earth Engine.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Cao H, Zhang H, Wang C, Zhang B (2019) Operational flood detection using Sentinel-1 SAR data over large areas. Water (Switzerland) 11:786. https://doi.org/10.3390/w11040786

Donchyts G, Schellekens J, Winsemius H, et al (2016) A 30 m resolution surface water mask including estimation of positional and thematic differences using Landsat 8, SRTM and OpenStreetMap: A case study in the Murray-Darling basin, Australia. Remote Sens 8:386. https://doi.org/10.3390/rs8050386

Markert KN, Markert AM, Mayer T, et al (2020) Comparing Sentinel-1 surface water mapping algorithms and radiometric terrain correction processing in Southeast Asia

utilizing Google Earth Engine. Remote Sens 12:2469. https://doi.org/10.3390/RS12152469

Mayer T, Poortinga A, Bhandari B, et al (2021) Deep learning approach for Sentinel-1 surface water mapping leveraging Google Earth Engine. ISPRS Open J Photogramm Remote Sens 2:100005. https://doi.org/10.1016/j.ophoto.2021.100005

Mullissa A, Vollrath A, Odongo-Braun C, et al (2021) Sentinel-1 SAR backscatter analysis ready data preparation in Google Earth Engine. Remote Sens 13:1954. https://doi.org/10.3390/rs13101954

Oddo PC, Bolten JD (2019) The value of near real-time Earth observations for improved flood disaster response. Front Environ Sci 7:127. https://doi.org/10.3389/fenvs.2019.00127

Otsu N (1979) A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern SMC-9:62–66. https://doi.org/10.1109/tsmc.1979.4310076

Pekel JF, Cottam A, Gorelick N, Belward AS (2016) High-resolution mapping of global surface water and its long-term changes. Nature 540:418–422. https://doi.org/10.1038/nature20584

Schumann G, Di Baldassarre G, Bates PD (2009) The utility of spaceborne radar to render flood inundation maps based on multialgorithm ensembles. IEEE Trans Geosci Remote Sens 47:2801–2807. https://doi.org/10.1109/TGRS.2009.2017937

Tellman B, Sullivan JA, Kuhn C, et al (2021) Satellite imaging reveals increased proportion of population exposed to floods. Nature 596:80–86. https://doi.org/10.1038/s41586-021-03695-w

Chapter A2.4 River Morphology

Authors
Xiao Yang, Theodore Langhorst, Tamlin M. Pavelsky

Overview

The purpose of this chapter is to showcase Earth Engine's application in fluvial hydrology and geomorphology. Specifically, we show examples demonstrating how to use Earth Engine to extract a river's centerline and width, and how to calculate the bank erosion rate. At the end of this chapter, you will be able to distinguish rivers from other water bodies, perform basic morphological analyses, and detect changes in river form over time.

Learning Outcomes

- Working with Landsat surface water products.
- Calculating river centerline location and width.
- Quantifying river bank erosion.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Perform image morphological operations (Chap. F3.2).
- Write a function and `map` it over an ImageCollection (Chap. F4.0).
- Use `reduceRegions` to summarize an image with zonal statistics in irregular shapes (Chap. F5.0, Chap. F5.2).
- Work with vector data (Chap. F5.1)

Introduction to Theory

The shape of a river viewed from above, known as its "planview geometry," can reveal many things about the river, including its morphological evolution and the flow of water and sediment within its channel. For example, hydraulic geometry establishes that a river's width and its discharge satisfy a power-law relation (rating curve). Thus, one can use such a relationship to monitor a river's discharge from river widths derived from

remote sensing images (Smith et al. 1996). Similarly, in addition to the natural variability of river size, rivers also adjust their courses on the landscape as water flows from the headwaters towards lowland downstream regions. These adjustments result in meandering, lateral variations in a river's course resulting from the erosion and accretion of sediment along the banks. Many tools have been developed to study morphological changes of rivers using remotely sensed images.

Early remote sensing of river form was done by manual interpretation of aerial imagery, but advances in computing power have facilitated and the volume of imagery from satellites has necessitated automated processing. The RivWidth software (Pavelsky and Smith 2008) first presented an automated method for river width extraction, and RivWidthCloud (RWC) (Yang et al. 2019) later applied and expanded these methods to Earth Engine. Similarly, methods for detecting changes in river form have evolved from simple tools that track hand-drawn river centerlines (Shields et al. 2000) to automated methods that can process entire basins (Constantine et al. 2014, Rowland et al. 2016). This progression towards automated software for studies in fluvial geomorphology has paired well with the capabilities of Earth Engine, and as a result many tools are being built for large-scale analysis in the cloud (Boothroyd et al. 2020).

Practicum

Section 1. Creating and Analyzing a Single River Mask

In this section, we will prepare an image and calculate some simple morphological attributes of a river. To do this, we will use a pre-classified image of surface water occurrence, identify which pixels represent the river and channel bars, and finally calculate the centerline, width, and bank characteristics.

Section 1.1. Isolate River from Water Surface Occurrence Map

Code Checkpoint A24a. The book's repository contains a script to use to begin this section. You will need to start with that script and paste code below into it.

The script includes our example area of interest (in the variable `aoi`) and two helper functions for reprojecting data to the local UTM coordinates. We force this projection and scale for many of our map layers because we are trying to observe and measure the river morphology. As data is viewed at different zoom levels, the shapes and apparent

connectivity of many water bodies will change. To allow a given dataset to be viewed with the same detail at multiple scales, we can force the data to be reprojected, as we do here.

The Joint Research Centre's surface water occurrence dataset (Pekel et al. 2016) classified the entire Landsat 5, 7, and 8 history and produced annual maps that identify seasonal and permanent water classes. Here, we will include both seasonal and permanent water classes (represented by pixel values of ≥2) as water pixels (with value = 1) and the rest as non-water pixels (with value = 0). In this section, we will look at only one image at a time by choosing the image from the year 2000 (Fig. A2.4.1a). In the code below, `bg` serves as a dark background layer for other map layers to be seen easily.

```
// IMPORT AND VISUALIZE SURFACE WATER MASK.
// Surface water occurrence dataset from the JRC (Pekel et al., 2016).
var jrcYearly = ee.ImageCollection('JRC/GSW1_3/YearlyHistory');

// Select the seasonal and permanent pixels image representing the
year 2000
var watermask = jrcYearly.filter(ee.Filter.eq('year', 2000)).first()
    .gte(2).unmask(0)
    .clip(aoi);

Map.centerObject(aoi);
Map.addLayer(ee.Image.constant(0), {
    min: 0,
    palette: ['black']
}, 'bg', false);
Map.addLayer(watermask, {}, 'watermask', false);
```

Next, we clean up the water mask by filling in small gaps by performing a closing operation (dilation followed by erosion). Areas of non-water pixels inside surface water bodies in the water mask may represent small channel bars, which we will fill in to create a simplified water mask. We identify these bars using a vectorization; however, you could do a similar operation with the `connectedPixelCount` method for bars up to 256 pixels in size (Fig. A2.4.1b). Filling in these small bars in the river mask improves the creation of a new centerline later in the lab.

```
// REMOVE NOISE AND SMALL ISLANDS TO SIMPLIFY THE TOPOLOGY.

// a. Image closure operation to fill small holes.
watermask = watermask.focal_max().focal_min();

// b. Identify small bars and fill them in to create a filled water
mask.
var MIN_SIZE = 2E3;
var barPolys = watermask.not().selfMask()
    .reduceToVectors({
        geometry: aoi,
        scale: 30,
        eightConnected: true
    })
    .filter(ee.Filter.lte('count', MIN_SIZE)); // Get small polys.
var filled = watermask.paint(barPolys, 1);

Map.addLayer(rpj(filled), {
    min: 0,
    max: 1
}, 'filled water mask', false);
```

Note here that we forced reprojection of the map layer using the helper function `rpj`. This means we have to be careful to keep our domain small enough to be processed at the set scale when doing the calculation on the fly in the Code Editor; otherwise, we will run out of memory. The reprojection may not be necessary when exporting the output using a task.

In the following step, we extract water bodies in the water mask that correspond to rivers. We will define a river mask (Fig. A2.4.1d) to be pixels that are connected to the river centerline according to the filled water mask. The channel mask (Fig. A2.4.1c) is defined also by connectivity but excludes the small bars, which will give us more accurate widths and areas for change detection in Sects. 1.2 and 1.3.

We can extract the river mask by checking the water pixels' connectivity to a provided river location database. Specifically, we use the Earth Engine method `cumulativeCost`

to identify connectivity between the filled water mask and the pixels corresponding to the river dataset. By inverting the filled mask, the cost to traverse water pixels is 0, and the cost over land pixels is 1. Pixels in the cost map with a value of 0 are entirely connected to the Surface Water and Ocean Topography (SWOT) Mission River Database (SWORD) centerline points by water, and pixels with values greater than 0 are separated from SWORD by land. The SWORD data, which were loaded as assets in the starter script, have some points located on land, either because the channel bifurcates or because the channel has migrated, so we must exclude those from our cumulative cost parameter `source`, or they will appear as single pixels of 0 in our cost map.

The `maxDistance` parameter must be set to capture maximum distance between centerline points and river pixels. In a single-threaded river with an accurate centerline, the ideal `maxDistance` value would be about half the river width. However, in reality, the centerlines are not perfect, and large islands may separate pixels from their nearest centerline. Unfortunately, increasing `maxDistance` has a large computational penalty, so some tweaking is required to get an optimal value. We can set `geodeticDistance` to **`false`** to regain some computational efficiency, because we are not worried about the accuracy of the distances.

```
// IDENTIFYING RIVERS FROM OTHER TYPES OF WATER BODIES.
// Cumulative cost mapping to find pixels connected to a reference
centerline.
var costmap = filled.not().cumulativeCost({
    source: watermask.and(ee.Image().toByte().paint(sword,
        1)),
    maxDistance: 3E3,
    geodeticDistance: false
});

var rivermask = costmap.eq(0).rename('riverMask');
var channelmask = rivermask.and(watermask);

Map.addLayer(sword, {
    color: 'red'
}, 'sword', false);
Map.addLayer(rpj(costmap), {
    min: 0,
```

```
    max: 1E3
}, 'costmap', false);
Map.addLayer(rpj(rivermask), {}, 'rivermask', false);
Map.addLayer(rpj(channelmask), {}, 'channelmask', false);
```

Code Checkpoint A24b. The book's repository contains a script that shows what your code should look like at this point.

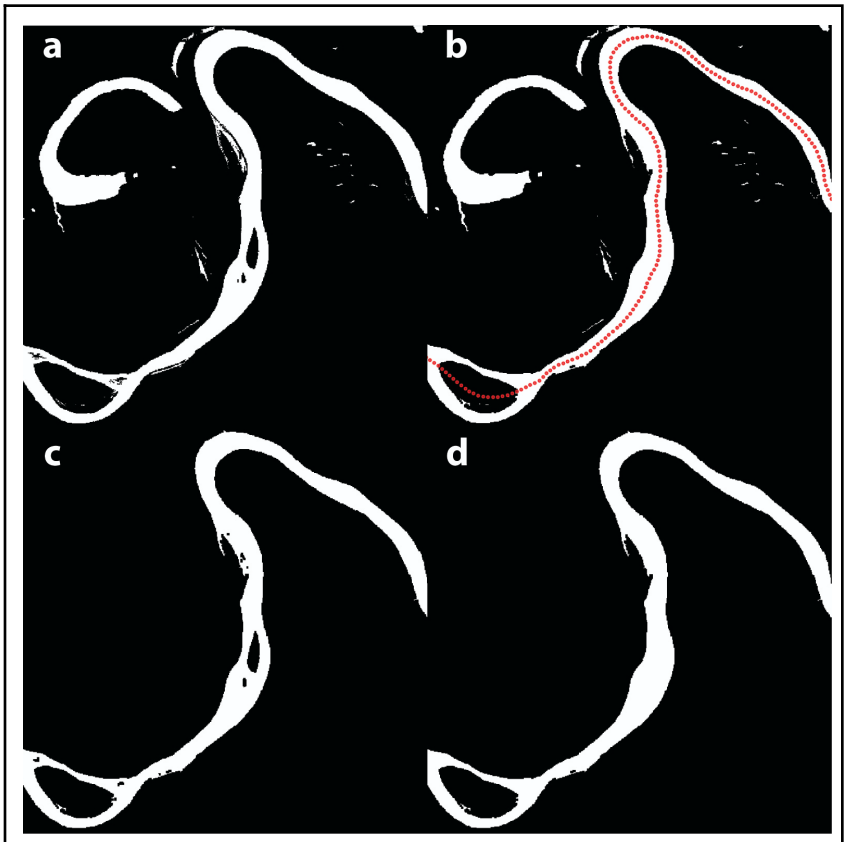


Fig. A2.4.1 (a) water mask; (b) filled water mask with prior centerline points; (c) channel mask; (d) river mask

Section 1.2. Obtain River Centerline and Width

After processing the image to create a river mask, we will use existing functions from RivWidthCloud to process the image further to obtain river centerlines and widths. Here we will call RivWidthCloud functions directly, taking advantage of the ability to use

exposed functions from another Earth Engine script (using the `require` functionality to load another script as a module). We will explain the usage and purpose of the RivWidthCloud functions used here.

There are three major steps involved in obtaining river widths from a given river mask:

1. Calculate one-pixel-width river centerlines.
2. Estimate the direction orthogonal to the flow direction for each centerline pixel.
3. Quantify river width on the channel mask along the orthogonal directions.

Extract River Centerline

We rely on morphological image analysis techniques to extract a river centerline. This process involves three steps:

1. Using distance transform to enhance pixels near the centerline of the river.
2. Using gradient to further isolate the centerline pixel having local minimal gradient values.
3. Cleaning the raw centerline by removing spurious centerlines.

First, a distance transform is applied to the river mask, resulting in a raster image where the value of each water pixel in the river mask is replaced by the closest distance to the shore. This step is done by using the `CalcDistanceMap` function from RWC. From Fig. A2.4.2a, we can see that, in the distance transform, the center of the river has the highest values.

```
// Import existing functions from RivWidthCloud.
var riverFunctions = require(
    'users/eeProject/RivWidthCloudPaper:functions_river.js');
var clFunctions = require(
    'users/eeProject/RivWidthCloudPaper:functions_centerline_width.js'
    );

//Calculate distance from shoreline using distance transform.

var distance = clFunctions.CalcDistanceMap(rivermask, 256, scale);
Map.addLayer(rpj(distance), {
    min: 0,
    max: 500
}, 'distance raster', false);
```

Second, to isolate the centerline of the river, we apply a gradient calculation to the distance raster. If we treat the distance raster as a digital elevation model (DEM), then the locations of the river centerline can be visualized as a ridgeline. They will thus have minimal gradient value. The gradient calculation is important, as it converts a local property of the centerline (local maximum distance) to a global property (global minimal gradient) to allow extraction of the centerline with a fixed gradient threshold (Fig. A2.4.2b). We use a 0.9 threshold (recommended for RivWidth (Pavelsky and Smith, 2008) and RWC) to extract the centerline pixels from the gradient image. However, the resulting initial centerline is not always one pixel wide. To ensure a one-pixel-wide centerline, iterative image skeletonization is applied to thin the initial centerline (Fig. A2.4.2c).

```
// Calculate gradient of the distance raster.
// There are three different ways (kernels) to calculate the gradient.
// By default, the function used the second approach.
// For details on the kernels, please see the source code for this
function.
var gradient = clFunctions.CalcGradientMap(distance, 2, scale);
Map.addLayer(rpj(gradient), {}, 'gradient raster', false);

// Threshold the gradient raster and derive 1px width centerline using
skeletonization.

var centerlineRaw = clFunctions.CalcOnePixelWidthCenterline(rivermask,
    gradient, 0.9);
var raw1pxCenterline = rpj(centerlineRaw).eq(1).selfMask();
Map.addLayer(raw1pxCenterline, {
    palette: ['red']
}, 'raw 1px centerline', false);
```

Third, the centerline from the previous step will have noise along the shoreline and will have spurious branches resulting from side channels or irregular channel forms that need to be pruned. The pruning function in RWC, `CleanCenterline`, works by first identifying end pixels of the centerline (i.e., centerline pixels with only one neighboring pixel) and then erasing pixels along the centerline pixels starting from the end pixels for a distance specified by `MAXDISTANCE_BRANCH_REMOVAL`. It will stop if the specified

distance is reached or the erasing encounters a joint pixel (i.e., pixels having more than two neighboring pixels). After pruning, the final centerline should look like Fig. A2.4.2d.

```
// Prune the centerline to remove spurious branches.
var MAXDISTANCE_BRANCH_REMOVAL = 500;
// Note: the last argument of the CleanCenterline function enables
removal of the pixels so that the resulting centerline will have 1px
width in an 8-connected way. Once it is done, it doesn't need to be
done the second time (thus it equals false)
var cl1px = clFunctions
    .CleanCenterline(centerlineRaw, MAXDISTANCE_BRANCH_REMOVAL, true);
var cl1px = clFunctions
    .CleanCenterline(cl1px, MAXDISTANCE_BRANCH_REMOVAL, false);
var final1pxCenterline = rpj(cl1px).eq(1).selfMask();
Map.addLayer(final1pxCenterline, {
    palette: ['red']
}, 'final 1px centerline', false);
```

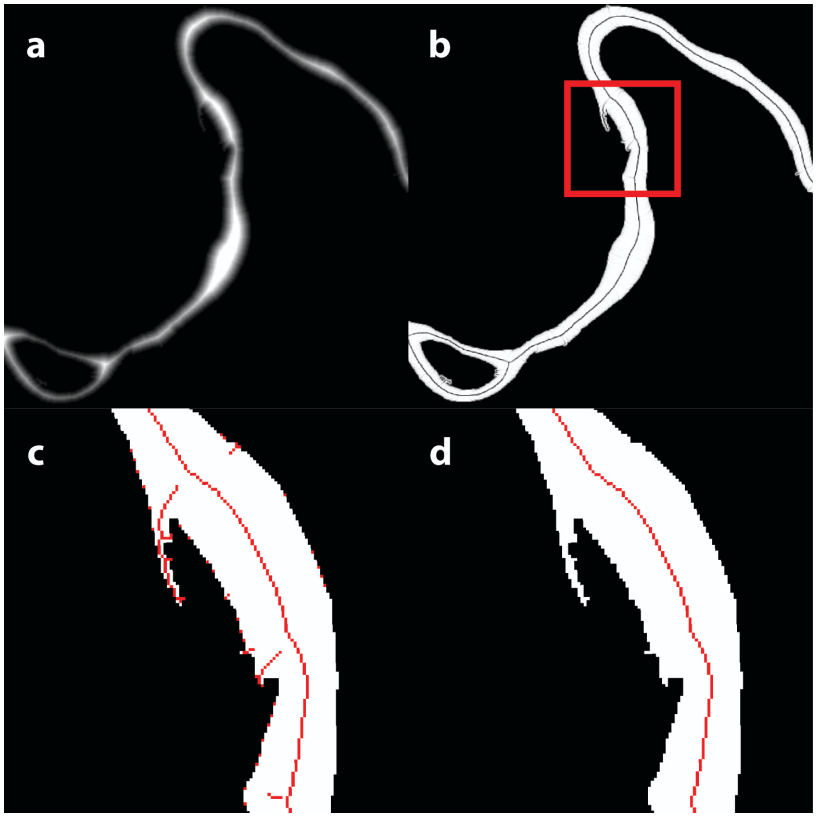


Fig. A2.4.2 Steps extracting river centerline: (a) distance transform of a river mask; (b) gradient of the distance map (a); (c) raw centerline after skeletonization; (d) centerline after pruning.

Estimate Cross-Sectional Direction

Now we will use the centerline we obtained from the previous step to help us measure the widths of the river. River width is often measured along the direction perpendicular to the flow, which we will approximate using the course of its centerline. To estimate cross-sectional directions, we convolve the centerline image with a customized kernel. The square 9 x 9 kernel has been designed so that each pixel on its rim has the radian value of the angle between the line connecting the rim pixel and the center of the kernel and the horizontal x-axis (radian angle 0). The convolution works by overlapping the center of the kernel with the centerline and calculating the average of the values of the rim pixels that overlap the centerline pixels, which corresponds to the cross-sectional direction of the particular centerline point under consideration. Here we use the function `CalculateAngle` to estimate the cross-sectional angles. The resulting raster will replace each centerline pixel with the value of the cross-sectional directions in degrees.

```
// Calculate perpendicular direction for the cleaned centerline.
var angle = clFunctions.CalculateAngle(cl1px);
var angleVis = {
    min: 0,
    max: 360,
    palette: ['#ffffd4', '#fed98e', '#fe9929', '#d95f0e',
        '#993404'
    ]
};
Map.addLayer(rpj(angle), angleVis, 'cross-sectional directions',
    false);
```

Quantify River Widths

To estimate river width, we will be using the RWC function `rwGen_waterMask`. This function can take any binary water mask image as input to calculate river widths, so long as the band name is 'waterMask' and contains the following three properties: 1) crs—UTM projection code, 2) scale—native spatial resolution, and 3) image_id—acting as an identifier for the output widths. This function works by first processing the input water mask to create all the intermediate images mentioned before (channel mask, river mask, centerline, and angle image). Then it creates a `FeatureCollection` of cross-sectional lines, each centered on one centerline pixel (from the centerline raster) along the direction estimated in the "Estimate Cross-Sectional Direction" section (from the angle raster) and with a length three times longer than the distance from the centerline point to the closest shoreline pixel (obtained from the distance raster). This `FeatureCollection` is then used in a `Image.reduceRegions` method as the `FeatureCollection` input. With a mean reducer, the result denotes the ratio between the actual river width and the length of the line segment (which is known). Thus, the final river width can be estimated by multiplying the ratio with the length of each line segment in the `FeatureCollection`. However, the scaling factor of 3 is chosen empirically, and can over- or underestimate the maximum extent of river width. This is because the width, scaled by 3, is the minimal distance from centerline pixels to the nearest shoreline pixels. When aligning line segments along the directions orthogonal to the river centerline, we might encounter situations when the length of these segments is too short to cover the width of the river (underestimation) or too long that they overlap with neighboring river reaches (overestimation). In both cases, the end(s) of the line segment

overlaps with a pixel identified as "water" in the channel mask. Thus, additional steps are taken to flag these measurements.

The `rwGen_waterMask` takes four arguments—maximum search distance (unit: meter) to label river pixels, maximum size of islands (unit: pixel) to be filled in to calculate river mask, distance (unit: meter) to be pruned to clean the raw centerline, and the area of interest to carry out the width calculation. The output of the `rwc` function is a `FeatureCollection` with each feature having the properties listed in Table A2.4.1.

Table A2.4.1 Output variables from the `rwc` function.

`longitude`	Longitude of the centerline point
`latitude`	Latitude of the centerline point
`width`	Wetted river width measured at the centerline point
`orthogonalDirection`	Angle of the cross-sectional direction at the centerline point
`flag_elevation`	Mean elevation across the river surface (unit: meter) based on MERIT DEM
`image_id`	Image ID of the input image
`crs`	The projection of the input image
`endsInWater`	Indicates inaccurate width due to the insufficient length of the cross-sectional segment that was used to measure the river width
`endsOverEdge`	Indicates width too close to the edge of the image such that the width can be inaccurate

```
// Estimate width.
var rwcFunction = require(
    'users/eeProject/RivWidthCloudPaper:rwc_watermask.js');
var rwc = rwcFunction.rwGen_waterMask(4000, 333, 500, aoi);
watermask = ee.Image(watermask.rename(['waterMask']).setMulti({
```

```
    crs: crs,
    scale: 30,
    image_id: 'aoi'
}));

var widths = rwc(watermask);
print('example width output', widths.first());
```

Section 1.3. Bank Morphology

In addition to a river's centerline and width, we can also extract information about the banks of the river, such as their aspect and total length. To identify the banks, we simply dilate the channel mask and compare it to the original channel mask. The difference in these images represents the land pixels adjacent to the channel mask.

```
var bankMask = channelmask.focal_max(1).neq(channelmask);
```

Next, we will calculate the aspect, or compass direction, of the bank faces. We use the `Image.cumulativeCost` method with the entire river channel as our source to create a new image (bankDistance) with increasing values away from the river channel, similar to an elevation map of river banks. In this image, the banks will 'slope' towards the river channel and we can take advantage of the terrain methods in EE. We will call the `Terrain.aspect` method on the bank distance and select the bank pixels by applying the bank mask. In the end, our bank aspect data will give us the direction from each bank pixel towards the center of the channel. These data could be useful for interpreting any directional preferences in erosion as a result of geological features or thawed permafrost soils from solar radiation.

```
var bankDistance = channelmask.not().cumulativeCost({
    source: channelmask,
    maxDistance: 1E2,
    geodeticDistance: false
});

var bankAspect = ee.Terrain.aspect(bankDistance)
    .multiply(Math.PI).divide(180)
    .mask(bankMask).rename('bankAspect');
```

Last, we calculate the length represented by each bank pixel by convolving the bank mask with a Euclidean distance kernel. Sections of bank oriented along the pixel edges will have a value of 30 m per pixel, whereas a diagonal section will have a value of $\sqrt{2}$ * 30 m per pixel.

```
var distanceKernel = ee.Kernel.euclidean({
    radius: 30,
    units: 'meters',
    magnitude: 0.5
});
var bankLength = bankMask.convolve(distanceKernel)
    .mask(bankMask).rename('bankLength');

var radianVis = {
    min: 0,
    max: 2 * Math.PI,
    palette: ['red', 'yellow', 'green', 'teal', 'blue', 'magenta',
        'red'
    ]
};
Map.addLayer(rpj(bankAspect), radianVis, 'bank aspect', false);
Map.addLayer(rpj(bankLength), {
    min: 0,
    max: 60
}, 'bank length', false);
```

Code Checkpoint A24c. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Multitemporal River Width

Refresh the Code Editor to begin with a new script for this section.

In Sect. 1.2, we walked through the process of extracting the river centerline and width from a given water mask. In that section, we intentionally unpacked the different steps used to extract river centerline and width so that readers can: (1) get an intuitive idea of how the image processes work step by step and see the resulting images at each stage;

(2) combine these functions to answer different questions (e.g., readers might only be interested in river centerlines instead of getting all the way to widths). In this section, we will walk you through how to use some high-level functions in RivWidthCloud to more efficiently implement these steps across multiple water mask images to extract time series of widths at a given location. To do this, we need to provide two inputs: a point of interest (longitude, latitude) and a collection of binary water masks. The code below re-introduces a helper function to convert between projections, then accesses other data and functionality.

```
var getUTMProj = function(lon, lat) {
    // Given longitude and latitude in decimal degrees,
    // return EPSG string for the corresponding UTM projection. See:
    //
https://apollomapping.com/blog/gtm-finding-a-utm-zone-number-easily
    // https://sis.apache.org/faq.html
    var utmCode = ee.Number(lon).add(180).divide(6).ceil().int();
    var output = ee.Algorithms.If({
        condition: ee.Number(lat).gte(0),
        trueCase: ee.String('EPSG:326').cat(utmCode
            .format('%02d')),
        falseCase: ee.String('EPSG:327').cat(utmCode
            .format('%02d'))
    });
    return (output);
};

// IMPORT AND VISUALIZE SURFACE WATER MASK
// Surface water occurrence dataset from the JRC (Pekel et al., 2016).
var jrcYearly = ee.ImageCollection('JRC/GSW1_3/YearlyHistory');
var poi = ee.Geometry.LineString([
    [110.77450764660864, 30.954167027937988],
    [110.77158940320044, 30.950633845897112]
]);

var rwcFunction = require(
    'users/eeProject/RivWidthCloudPaper:rwc_watermask.js');
```

Remember that the widths from Sect. 1.2 are stored in a `FeatureCollection` with multiple width values from different locations along a centerline. To extract the multitemporal river width for a particular location along a river, we only need one width measurement from each water mask. Here, we choose the width for the centerline pixel that is nearest to the given point of interest using the function `getNearestCl`. This function takes the width `FeatureCollection` from Sect. 1.2 as input and returns a feature corresponding to the width closest to the point of interest.

```
// Function to identify the nearest river width to a given location.
var GetNearestClGen = function(poi) {
    var temp = function(widths) {
        widths = widths.map(function(f) {
            return f.set('dist2cl', f.distance(poi,
                30));
        });

        return ee.Feature(widths.sort('dist2cl', true)
            .first());
    };
    return temp;
};
var getNearestCl = GetNearestClGen(poi);
```

Then we will need to use the `map` method on the input collection of water masks to apply the `rwc` to all the water mask images. This will result in a `FeatureCollection`, each feature of which will contain the width quantified from one image (or time stamp).

```
// Multitemporal width extraction.
var polygon = poi.buffer(2000);
var coords = poi.centroid().coordinates();
var lon = coords.get(0);
var lat = coords.get(1);
var crs = getUTMProj(lon, lat);
var scale = ee.Number(30);

var multiwidths = ee.FeatureCollection(jrcYearly.map(function(i) {
    var watermask = i.gte(2).unmask(0);
```

```
    watermask = ee.Image(watermask.rename(['waterMask'])
        .setMulti({
            crs: crs,
            scale: scale,
            image_id: i.getNumber('year')
        }));
    var rwc = rwcFunction.rwGen_waterMask(2000, 333, 300,
        polygon);
    var widths = rwc(watermask)
        .filter(ee.Filter.eq('endsInWater', 0))
        .filter(ee.Filter.eq('endsOverEdge', 0));

    return ee.Algorithms.If(widths.size(), getNearestCl(
        widths), null);
}, true));

var widthTs = ui.Chart.feature.byFeature(multiwidths, 'image_id', [
        'width'
    ])
    .setOptions({
        hAxis: {
            title: 'Year',
            format: '####'
        },
        vAxis: {
            title: 'Width (meter)'
        },
        title: 'River width time series upstream of the Three Gorges
Dam'
    });
print(widthTs);

Map.centerObject(polygon);
Map.addLayer(polygon, {}, 'area of width calculation');
```

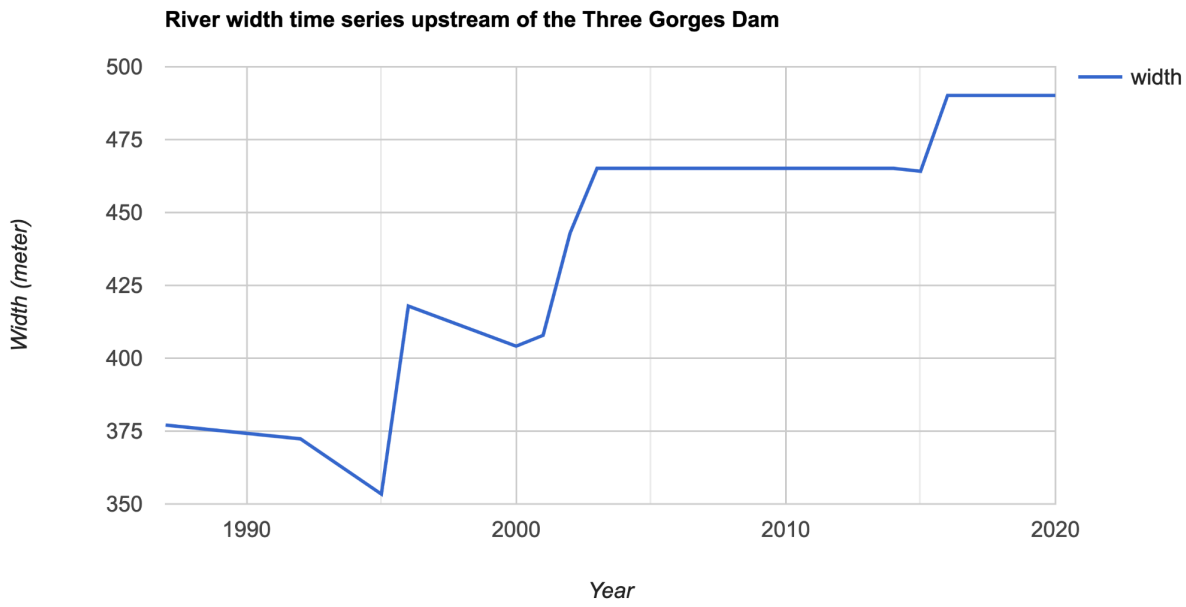


Fig. A2.4.3 River width time series upstream of the Three Gorges Dam in China. The series shows the abrupt increase in river width around the year 2003, when the dam was completed.

Code Checkpoint A24d. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Riverbank Erosion

In this section, we will apply the methods we developed in Sect. 1 to multiple images, calculate the amount of bank erosion, and summarize our results back onto our centerline. Before doing so, we will create a new script that wraps the masking and morphology code in Sects. 1.1 and 1.3 into a function called `makeChannelmask` that has one argument for the year. We return an image with bands for all of the masks and bank calculations, plus a property named `'year'` that contains the year argument. If you have time, you could try to create this function on your own and then compare with our implementation of it, in the next code checkpoint. Note that we would not expect that your code would look the same, but it should ideally have the same functionality.

Code Checkpoint A24e. The book's repository contains a script to use to begin this section. You will need to start with that script and paste code below into it.

Change Detection

We will use a section of the Madre de Dios River as our study area for this example because it migrates very quickly, more than 30 m per year in some locations. Our methods will work best if the two channel masks partially overlap everywhere along the length of the river; if there is a gap between the two masks, we will underestimate the amount of change and not be able to calculate the direction of change. As such, we will pick the years 2015 and 2020 for our example. However, in other locations, you may want to increase the time span in order to observe more change. We first create these two sets of channel masks and add them to the map (Fig. A2.4.4a).

```
var masks1 = makeChannelmask(2015);
var masks2 = makeChannelmask(2020);
Map.centerObject(aoi, 13);
var year1mask = rpj(masks1.select('channelmask').selfMask());
Map.addLayer(year1mask, {
    palette: ['blue']
}, 'year 1');
var year2mask = rpj(masks2.select('channelmask').selfMask());
Map.addLayer(year2mask, {
    palette: ['red']
}, 'year 2', true, 0.5);
```

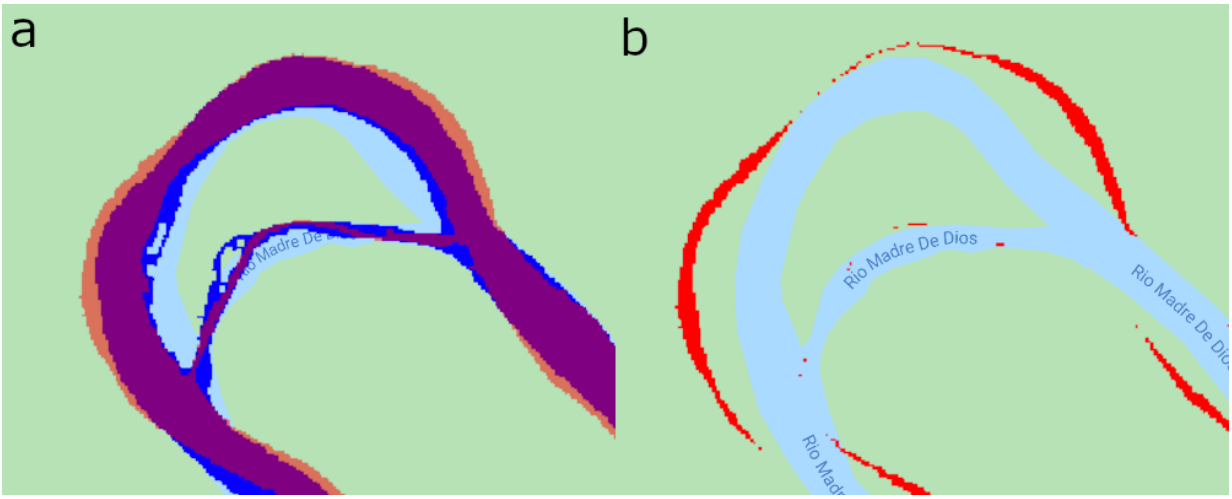


Fig. A2.4.4 A single meander bend of the Madre de Dios River in Bolivia, showing areas of erosion and accretion: (a) channel mask from 2015 in blue and channel mask from 2020 in red at 50% transparency; (b) pixels that represent erosion between 2015 and 2020

Next, we create an image to represent the eroded area (Fig. A2.4.4b). We can quickly calculate this by comparing the channel mask in year 2 to the inverse water mask from year 1. In alluvial river systems, avulsions and meander cutoffs can leave fragments of old channels near the river. If the river meanders back into these water bodies, we want to be careful not to count these as fully eroded, which is why we need to compare our river pixels in year 2 (channel mask) to the land pixels in year 1 (inverse water mask). If you were to compare only the channel masks from year to year, water in the floodplains that is captured by the channel migration would be falsely counted as erosion.

```
// Pixels that are now the river channel but were previously land.
var erosion = masks2.select('channelmask')
    .and(masks1.select('watermask').not()).rename('erosion');
Map.addLayer(rpj(erosion).selfMask(), {}, 'erosion', false);
```

Now we are going to approximate the direction of erosion. We will define the direction of erosion by the shortest path through the eroded area from each bank pixel in year 1 to any of the bank pixels in year 2. In reality, meandering rivers often translate their shape downvalley, which breaks our definition of the shortest path between banks. However,

the shortest path produces a reasonable approximation in most cases and is easy to calculate. We will again use `Image.cumulativeCost` to measure the distance using the erosion image as our cost surface. The erosion image has to be dilated by 1 pixel to compensate for the missing edge pixels in the gradient calculations, and masked in order to limit the cost paths to within the eroded area.

```
// Erosion distance assuming the shortest distance between banks.
var erosionEndpoints = erosion.focal_max(1).and(masks2.select(
    'bankMask'));
var erosionDistance = erosion.focal_max(1).selfMask()
    .cumulativeCost({
        source: erosionEndpoints,
        maxDistance: 1E3,
        geodeticDistance: true
    }).rename('erosionDistance');
Map.addLayer(rpj(erosionDistance),
    {
        min: 0,
        max: 300
    },
    'erosion distance',
    false);
```

Now we can use the same `Terrain.aspect` method that we used for the bank aspect to calculate the direction of the shortest path along our cost surface. You could also calculate this direction (and the bank aspect in Sect. 1.3) using the `Image.gradient` method and then calculating the tangent of the resulting *x* and *y* components.

```
// Direction of the erosion following slope of distance.
var erosionDirection = ee.Terrain.aspect(erosionDistance)
    .multiply(Math.PI).divide(180)
    .clip(aoi)
    .rename('erosionDirection');
erosionDistance = erosionDistance.mask(erosion);
Map.addLayer(rpj(erosionDirection),
```

```
    {
        min: 0,
        max: Math.PI
    },
    'erosion direction',
    false);
```

Connecting to the Centerline

We now have all of our change metrics calculated as images in Earth Engine. We could export these and make maps and figures using these data. However, when analyzing a lot of river data, we often want to look at long profiles of a river or tributary networks in a watershed. In order to do this, we will use reducers to summarize our raster data back onto our vector centerline. The first step is to identify which pixels should be assigned to which centerline points. We will start by calculating a single image representing the distance to any SWORD centerline point with the `FeatureCollection.distance` method. Next, we will use a convolution with the Laplacian kernel (Chap. F3.2) as an edge detection method on our distance raster. By convolving the distance to the nearest SWORD node with the Laplacian kernel, we are calculating the second derivative of distance, and can find the locations where the distance surface starts sloping towards another SWORD point.

```
// Distance to nearest SWORD centerline point.
var distance = sword.distance(2E3).clip(aoi);

// Second derivatives of distance.
// Finding the 0s identifies boundaries between centerline points.
var concavityBounds = distance.convolve(ee.Kernel.laplacian8())
    .gte(0).rename('bounds');

Map.addLayer(rpj(distance), {
    min: 0,
    max: 1E3
}, 'distance', false);
Map.addLayer(rpj(concavityBounds), {}, 'bounds', false);
```

Next, we need to create an image where each pixel's value is set to the unique node identifier of the nearest SWORD centerline point. We will create a two-band image, where the first band is the concavity boundaries found in the last step, and the second band has the unique node identifiers painted on their location. When we reduce this image using the `Image.reduceConnectedComponents` method, we set all pixels in each region with the corresponding node ID. Last, we need to dilate these pixels to fill in the boundary gaps using a call to the `Image.focalMode` method.

```
// Reduce the pixels according to the concavity boundaries,
// and set the value to SWORD node ID.  Note that focalMode is used
// to fill in the empty pixels that were the boundaries.
var swordImg = ee.Image(0).paint(sword, 'node_id').rename('node_id')
    .clip(aoi);
var nodePixels = concavityBounds.addBands(swordImg)
    .reduceConnectedComponents({
        reducer: ee.Reducer.max(),
        labelBand: 'bounds'
    }).focalMode({
        radius: 3,
        iterations: 2
    });
Map.addLayer(rpj(nodePixels).randomVisualizer(),
    {},
    'node assignments',
    false);
```

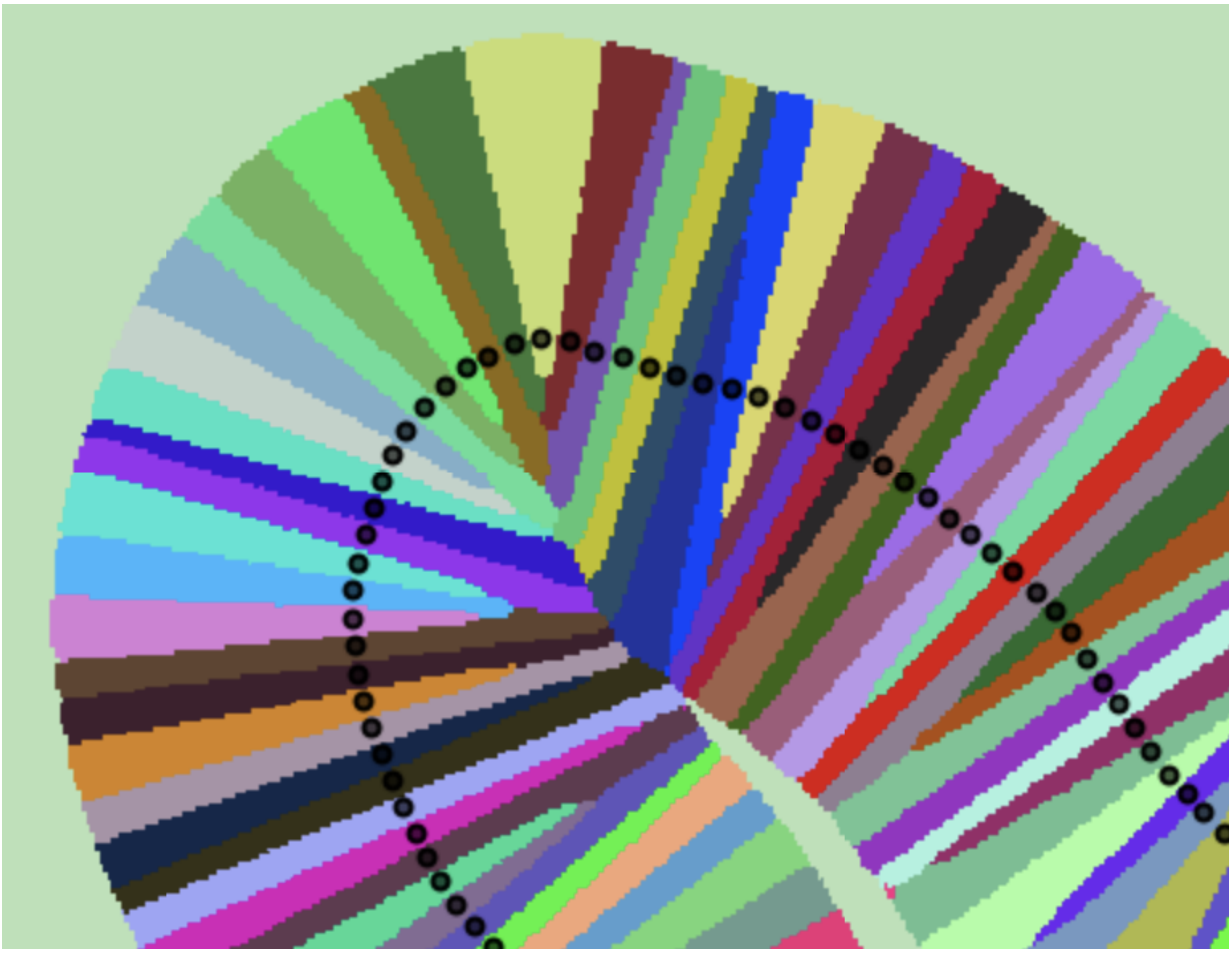


Fig. A2.4.5 A section of the Madre de Dios River where each pixel is assigned to its closest centerline node

Summarizing the Data

The final step in this section is to apply a reducer that uses our `nodePixels` image from the previous step to group our raster data. We will combine the `reducer.forEach` and `reducer.group` methods into our own custom function that we can use with different reducers to get our final results. The `reducer.forEach` method sets up a different reducer and output for each band in our image, which is necessary when we use the reducer.group method. The `reducer.group` method is conceptually similar to `reducer.reduceRegions`, except our regions are defined by an image band instead of by polygons. In some cases the group method is much faster than the

`reducer.reduceRegions` method, particularly if you were to have to convert your regions to polygons in order to provide the input to `reducer.reduceRegions`. The grouped reducers in our function return a list of dictionaries. However, it is much easier to work with feature collections, so we will map over the list and create a `FeatureCollection` before returning from the function.

```
// Set up a custom reducing function to summarize the data.
var groupReduce = function(dataImg, nodeIds, reducer) {
    // Create a grouped reducer for each band in the data image.
    var groupReducer = reducer.forEach(dataImg.bandNames())
        .group({
            groupField: dataImg.bandNames().length(),
            groupName: 'node_id'
        });

    // Apply the grouped reducer.
    var statsList = dataImg.addBands(nodeIds).clip(aoi)
        .reduceRegion({
            reducer: groupReducer,
            scale: 30,
        }).get('groups');

    // Convert list of dictionaries to FeatureCollection.
    var statsOut = ee.List(statsList).map(function(dict) {
        return ee.Feature(null, dict);
    });
    return ee.FeatureCollection(statsOut);
};
```

For some variables—such as the erosion, the channel mask, or the bank length—we want the total number of pixels or bank length, so we will use the `Reducer.sum` method with our grouped reducer function. For our aspect and directional variables, we need to use the `Reducer.circularMean` method to find the mean direction. The returned variables `sumStats` and `angleStats` are feature collections with properties for our reduced data and the corresponding node ID.

```
var dataMask = masks1.addBands(masks2).reduce(ee.Reducer
```

```
    .anyNonZero());

var sumBands = ['watermask', 'channelmask', 'bankLength'];
var sumImg = erosion
    .addBands(masks1, sumBands)
    .addBands(masks2, sumBands);
var sumStats = groupReduce(sumImg, nodePixels, ee.Reducer.sum());

var angleImg = erosionDirection
    .addBands(masks1, ['bankAspect'])
    .addBands(masks2, ['bankAspect']);
var angleStats = groupReduce(angleImg, nodePixels, ee.Reducer
    .circularMean());
```

Finally, we will join these two new feature collections to our original centerline data and print the results (Fig. A2.4.6).

```
var vectorData = sword.filterBounds(aoi).map(function(feat) {
    var nodeFilter = ee.Filter.eq('node_id', feat.get(
        'node_id'));
    var sumFeat = sumStats.filter(nodeFilter).first();
    var angleFeat = angleStats.filter(nodeFilter).first();
    return feat.copyProperties(sumFeat).copyProperties(
        angleFeat);
});

print(vectorData);
Map.addLayer(vectorData, {}, 'final data');
```

Code Checkpoint A24f. The book's repository contains a script that shows what your code should look like at this point.

```
▾ properties: Object (32 properties)
    geometry: Point
    bankAspect: -0.013657036314317621
    bankAspect_1: -0.009888223798589512
    bankLength: 1419.4112549695421
    bankLength_1: 1486.6904755831204
    channelmask: 489
    channelmask_1: 490
    cl_ids: 20118259
    cl_ids2: 20118265
    dist_out: 3006836.3250234
    erosion: 19
    erosionDirection: -0.004093294889776033
    ext_dist_coef: 20
    facc: 122415.923913638
    grod_id: 0
    hfalls_id: 0
    lake_id: NaN
    n_chan_max: 2
    n_chan_mod: 2
    node_id: 62266400080511
    node_length: 211.783182167348
    obstr_type: 0
    reach_id: 62266400081
    sin: 1.0475550651084
    watermask: 588
    watermask_1: 571
    wavelength: 9474.53707653225
    width: 760
    width_var: 2114.81632653061
    wse: 126.800003051758
    wse_var: 0
    wth_coef: 0.5
```

Fig. A2.4.6 The updated list of properties in our centerline dataset; new properties are outlined in black. The erosion and mask fields are in units of pixels, but you could convert to area using the `Image.pixelArea` method on the masks.

This workflow can be used to add many new properties to the river centerlines based on raster calculations. For example, we calculated the amount of erosion between these two years, but you could use very similar code to calculate the amount of accretion that occurred. Other interesting properties of the river, like the slope of the banks from a DEM, could be calculated and added to our centerline dataset.

Synthesis

Assignment 1. RivWidthCloud can estimate individual channel width in the case of multichannel rivers. Change the AOI to a multichannel river and observe the resulting centerline and width data. Note down things you think are different from the single-channel case.

Assignment 2. Answer the following question. When rivers experience both variable width over time and bank migration, how can we apply the methods in this chapter to distinguish these two types of changes?

Conclusion

In this chapter, we provide ways in which Earth Engine can be used to aid river planview morphological studies. In the first half of the chapter, we show how to distinguish river pixels from other types of water bodies, as well as how to extract river centerline, river width, bank aspect, and length. In the second half of the chapter, we give examples of how to apply these methods to multitemporal image collections to estimate changes in river widths for rivers that have stable channels, and to estimate bank erosion for rivers that tend to meander quickly. The analysis makes use of both raster- and vector-based methods provided by Earth Engine to help quantify river morphology. More importantly, these methods can be applied at scale.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Boothroyd RJ, Williams RD, Hoey TB, et al (2021) Applications of Google Earth Engine in fluvial geomorphology for detecting river channel change. Wiley Interdiscip Rev Water 8:e21496. https://doi.org/10.1002/wat2.1496

Constantine JA, Dunne T, Ahmed J, et al (2014) Sediment supply as a driver of river meandering and floodplain evolution in the Amazon Basin. Nat Geosci 7:899–903. https://doi.org/10.1038/ngeo2282

Pavelsky TM, Smith LC (2008) RivWidth: A software tool for the calculation of river widths from remotely sensed imagery. IEEE Geosci Remote Sens Lett 5:70–73. https://doi.org/10.1109/LGRS.2007.908305

Pekel JF, Cottam A, Gorelick N, Belward AS (2016) High-resolution mapping of global surface water and its long-term changes. Nature 540:418–422. https://doi.org/10.1038/nature20584

Rowland JC, Shelef E, Pope PA, et al (2016) A morphology independent methodology for quantifying planview river change and characteristics from remotely sensed imagery. Remote Sens Environ 184:212–228. https://doi.org/10.1016/j.rse.2016.07.005

Shields Jr FD, Simon A, Steffen LJ (2000) Reservoir effects on downstream river channel migration. Environ Conserv 27:54–66. https://doi.org/10.1017/S0376892900000072

Smith LC, Isacks BL, Bloom AL, Murray AB (1996) Estimation of discharge from three braided rivers using synthetic aperture radar satellite imagery: Potential application to ungaged basins. Water Resour Res 32:2021–2034. https://doi.org/10.1029/96WR00752

Yang X, Pavelsky TM, Allen GH, Donchyts G (2020) RivWidthCloud: An automated Google Earth Engine algorithm for river width extraction from remotely sensed imagery. IEEE Geosci Remote Sens Lett 17:217–221. https://doi.org/10.1109/LGRS.2019.2920225

Chapter A2.5: Water Balance and Drought

Authors
Ate Poortinga, Quyen Nguyen, Nyein Soe Thwal, Andréa Puzzi Nicolau

Overview

In this chapter, you will learn simple water balance calculations using remote-sensing-derived products related to precipitation and evapotranspiration. You will work at the river basin scale and perform time-series analysis, while comparing the data series with remote sensing vegetation and drought indices using the Earth Engine platform. You will also overlay the various indices with a land cover map to estimate potential drought impacts throughout the region.

Learning Outcomes

- Understanding the basics of remote-sensing-derived precipitation and evapotranspiration products.
- Calculating monthly aggregate statistics.
- Performing time-series analysis.
- Calculating vegetation and drought indices.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Create a graph using `ui.Chart` (Chap. F1.3).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Write a function and map it over an ImageCollection (Chap. F4.0).
- Summarize an ImageCollection with reducers (Chap. F4.0, Chap. F4.1).
- Aggregate data to build a time series (Chap. F4.2).
- Work with CHIRPS rainfall data (Chap. F4.2)
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).

Introduction to Theory

Water is vital for sustaining human life, ensuring food security, generating power, and supporting industrial processes in river basins. Both terrestrial and aquatic ecosystems are dependent on water to provide valuable ecosystem services, not only for the current generation but also for generations in the future. Managing the complex flow paths of water to and from these different water-use sectors requires a quantitative understanding of hydrological processes. Quantitative insights are necessary to help manage water consumption more efficiently by means of retention, withdrawals, and land use change. Managers need background data to help them optimize water allocation to sectors without further depleting the natural capital in the basin.

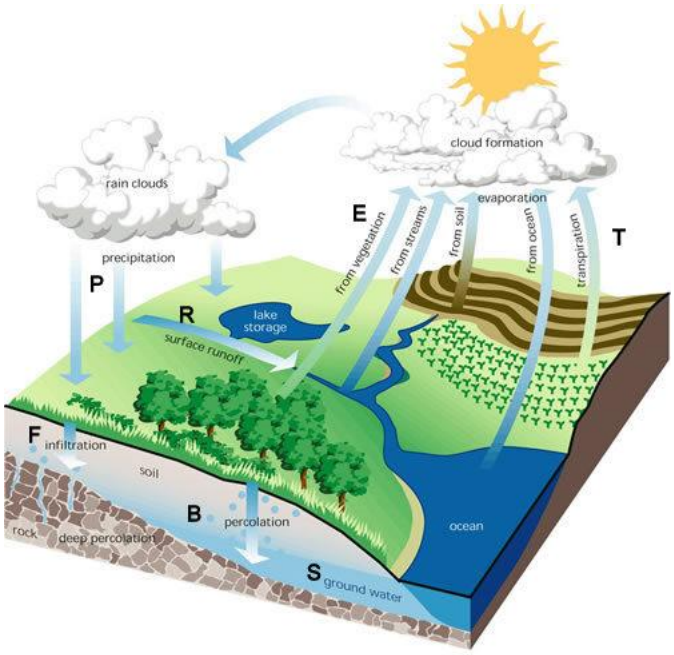


Fig. A2.5.1 The key components of the hydrological cycle

The water balance (Fig. A2.5.1) is the key concept in understanding the availability of water resources in a hydrological system. The water balance includes both input and extractions of water. In its simplest form, the water balance can be defined as Equation A2.5.1. Inputs to the hydrological system are defined by precipitation (P; rainfall and snow). Extractions for the system are from runoff (Q) and evapotranspiration (ET), with evapotranspiration denoting the sum of evaporation from the land surface plus

transpiration from plants. Water balance changes in groundwater and soil storage are indicated by ΔS.

$$P = Q + ET + \Delta S \qquad \text{(A2.5.1)}$$

A hydrological system, also referred to as river basin or drainage basin, is any land area where precipitation collects and drains off into a common outlet. The hydrological processes between upstream and downstream are interconnected: For example, extractions of water resources upstream will impact the amount of available downstream water resources. Similarly, upstream activities such as logging and swidden agriculture might impact the quality of downstream water resources. Fig. A2.5.2 shows the boundaries of the Lower Mekong River Basin, which covers parts of Laos, Thailand, Cambodia, Myanmar, and Vietnam. Water is a shared resource among the countries, although each country has its own legal framework to maintain and protect water supplies. A quantitative understanding of this shared resource is imperative to formulating effective management strategies.

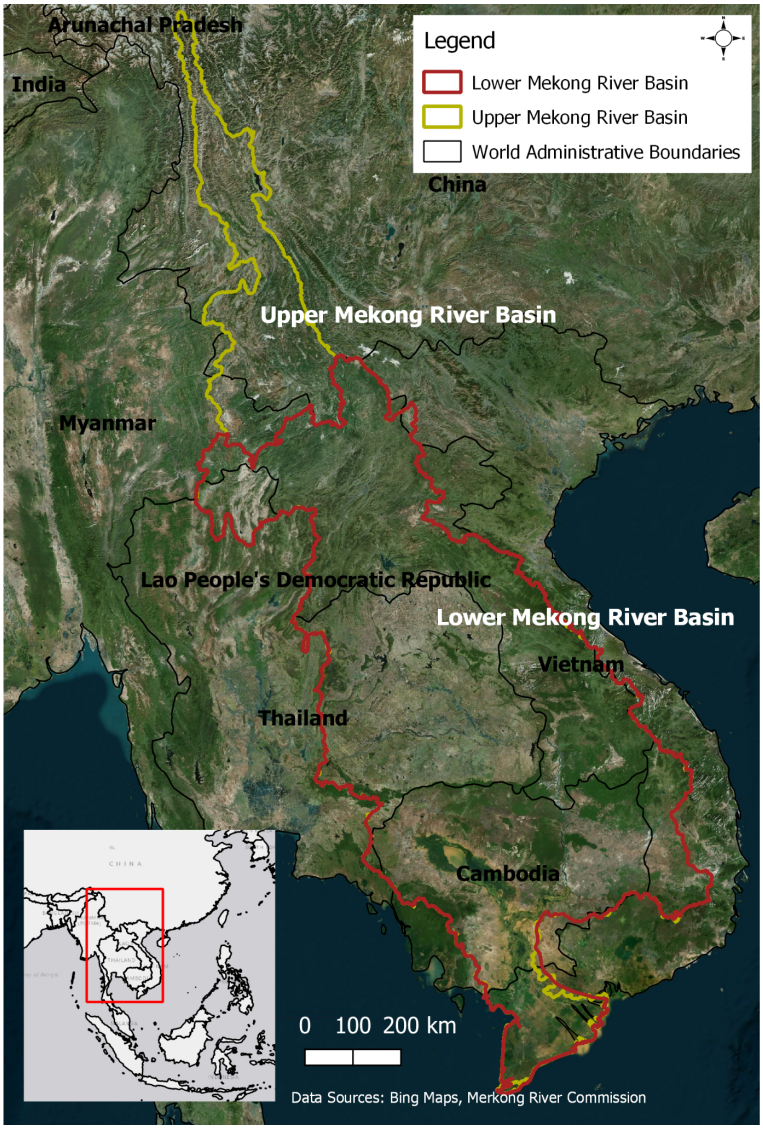


Fig. A2.5.2 The Upper and Lower Mekong River Basin

In the following exercises, we will calculate the main components of the water balance and link them with vegetation growth, drought information, and land cover information in the Lower Mekong Basin.

Practicum

Section 1. Calculating Monthly Precipitation

Precipitation has been measured for many centuries (Strangeways 2010). The traditional method is point measurement, which was standardized in the previous century to make measurements comparable in space and time. Fig. A2.5.3 shows a conventional weather station used to measure various weather-related parameters, including the amount of rainfall. Although statistical methods exist to calculate area averaged rainfall from weather stations, the limited number of data points remains a constraint, especially in developing countries and sparsely populated regions where the density of weather stations is low. Satellites can fill this information gap, as they observe the planet at a regular interval with calibrated sensors.

Fig. A2.5.3 Conventional weather station that measures various parameters, including precipitation

The Tropical Rainfall Measuring Mission (TRMM), a joint mission of the Japan Aerospace Exploration Agency (JAXA) and NASA, was a notable effort to monitor and study tropical rainfall (Kummerow et al. 1998). The satellite, which operated for 17 years, contained various instruments to measure clouds and cloud structures in order to

advance understanding of the global energy and water cycles. The Global Precipitation Measurement (GPM; Fig. A2.5.4) mission is the successor, with the primary aim of making frequent (every 2–3 hours) observations of Earth's precipitation (Hou et al. 2014). A wide variety of data products are available for both TRMM and GPM. Precipitation estimates are derived from the Precipitation Radar (PR), TRMM Microwave Imager (TMI), Visible Infrared Scanner (VIRS), Clouds and Earth's Radiant Energy System (CERES), and Lightning Imaging Sensor (LSI). Frequently used products of TRMM have a spatial resolution of 0.25 degrees (~25 km), whereas GPM has a higher resolution of 0.1 degrees (~10 km). Data is available on a three-hour time interval. The data can be obtained through NASA but also can be accessed from the Earth Engine data repository.

The Climate Hazards Group InfraRed Precipitation with Station (CHIRPS) data is a quasi-global rainfall dataset (Funk et al. 2015) covering more than 35 years. CHIRPS provides precipitation information at a 0.5 degrees (~5 km) spatial resolution. The dataset estimates precipitation by combining data from observing meteorological stations with satellite data. CHIRPS data is available at intervals from daily to annual and can be very valuable in hydrology studies, as it provides a long and consistent time series with precipitation estimates at a relatively high spatial resolution.

Fig. A2.5.4 The satellite for the Global Precipitation Measurement (GPM) mission

Below is the code and an exercise to calculate monthly precipitation using these satellite data.

We begin by importing our area of interest, the Lower Mekong River Basin (Fig. A2.5.5).

```
// Import the Lower Mekong boundary.
var mekongBasin = ee.FeatureCollection(
    'projects/gee-book/assets/A2-5/lowerMekongBasin');

// Center the map.
Map.centerObject(mekongBasin, 5);

// Add the Lower Mekong Basin boundary to the map.
Map.addLayer(mekongBasin, {}, 'Lower Mekong basin');
```

Fig. A2.5.5 The Lower Mekong Basin

In the next step, we set the start and end dates for our analysis. We create a list for both years and months, which we will later use to iterate over.

```
// Set start and end years.
```

```
var startYear = 2010;
var endYear = 2020;

// Create two date objects for start and end years.
var startDate = ee.Date.fromYMD(startYear, 1, 1);
var endDate = ee.Date.fromYMD(endYear + 1, 1, 1);

// Make a list with years.
var years = ee.List.sequence(startYear, endYear);

// Make a list with months.
var months = ee.List.sequence(1, 12);
```

We import the CHIRPS `ImageCollection` and select the imagery for the relevant dates, as presented in Chap. F4.2. Note that we used the pentad time series; each image in this collection contains the accumulated rainfall for five days. The daily product is also available in Earth Engine. The pentad dataset was used rather than the daily data product to reduce the number of computations needed to aggregate the data.

```
// Import the CHIRPS dataset.
var CHIRPS = ee.ImageCollection('UCSB-CHG/CHIRPS/PENTAD');

// Filter for the relevant time period.
CHIRPS = CHIRPS.filterDate(startDate, endDate);
```

The year and month lists are used in the function below to calculate the monthly rainfall. We use a server-side nested loop where we first `map` over the years (2010, 2011, … 2020) and then `map` over the months (1, 2, … 12). This returns an image with the total rainfall for each month. We set the year, month, and timestamp (`'system:time_start'`) for each image and flatten the image to turn the object into a single `ImageCollection`.

```
// We apply a nested loop where we first map over
// the relevant years and then map over the relevant
// months. The function returns an image with the total (sum)
// rainfall for each month. A flatten is applied to convert a
// feature collection of features into a single feature collection.
```

```
var monthlyPrecip = ee.ImageCollection.fromImages(
    years.map(function(y) {
        return months.map(function(m) {
            var w = CHIRPS.filter(ee.Filter
                    .calendarRange(y, y, 'year'))
                .filter(ee.Filter.calendarRange(m, m,
                    'month'))
                .sum();
            return w.set('year', y)
                .set('month', m)
                .set('system:time_start', ee.Date
                    .fromYMD(y, m, 1));

        });
    }).flatten()
);
```

Add a layer with the monthly mean precipitation to the map and calculate a chart with monthly mean precipitation (Fig. A2.5.6).

```
// Add the layer with monthly mean. Note that we clip for the Mekong
river basin.
var precipVis = {
    min: 0,
    max: 250,
    palette: 'white, blue, darkblue, red, purple'
};

Map.addLayer(monthlyPrecip.mean().clip(mekongBasin),
    precipVis,
    '2015 precipitation');

// Set the title and axis labels for the chart.
var title = {
    title: 'Monthly precipitation',
    hAxis: {
        title: 'Time'
```

```
    },
    vAxis: {
        title: 'Precipitation (mm)'
    },
};

// Plot the chart using the Mekong boundary.
var chartMonthly = ui.Chart.image.seriesByRegion({
        imageCollection: monthlyPrecip,
        regions: mekongBasin.geometry(),
        reducer: ee.Reducer.mean(),
        band: 'precipitation',
        scale: 5000,
        xProperty: 'system:time_start'
    }).setSeriesNames(['P'])
    .setOptions(title)
    .setChartType('ColumnChart');

// Print the chart.
print(chartMonthly);
```

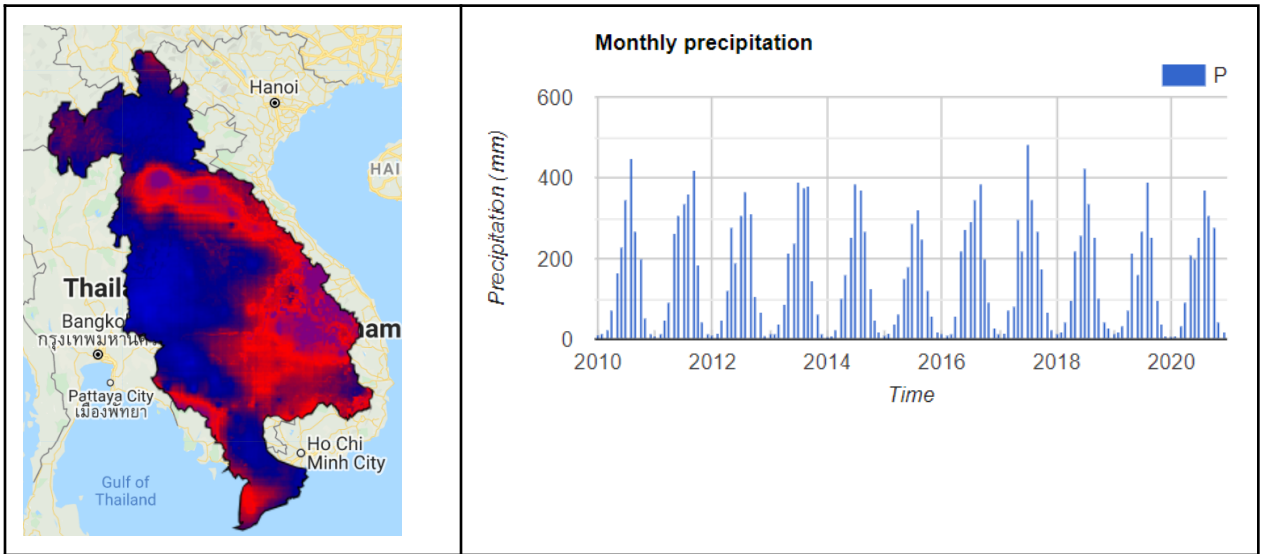


Fig. A2.5.6 Mean precipitation in the Lower Mekong Basin (left) and the monthly average precipitation (right)

Code Checkpoint A25a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Calculating Monthly Evapotranspiration

Measuring evapotranspiration at large scales is important for assessing climate and anthropogenic effects on natural and agricultural ecosystems (Kustas and Norman 1996). Methods exist to measure ET at a field scale, but those methods cannot be extrapolated to larger areas. Traditional ways of estimating ET have been to use reference ET, derived from various weather-related parameters, with a crop coefficient. However, there are large uncertainties due to, for example, spatial and temporal heterogeneity and data gaps. Satellite information can be very useful as it provides spatially and temporally dense information for ET estimation.

Different methods exist to map ET from remote sensing data, including simple empirical models that relate spectral reflectance with ET, vegetation index models, energy budget, and deterministic models (Courault et al. 2005). Fundamentally, however, ET is governed by the energy budget and driving variables such as surface temperature. The total amount of available net radiant energy is divided into a soil heat flux and the atmospheric convective fluxes, which are the sensible heat flux (H) and latent energy exchanges (LE). This essentially means that the temperature decreases when energy is used for ET. Indeed, there are different ways to further quantify the different energy fluxes in more detail, and there is a wide body of scientific literature that describes those methods.

There are different readily available ET products derived from the Moderate Resolution Imaging Spectroradiometer (MODIS), including Atmosphere-Land Exchange Inverse (ALEXI; Anderson et al. 1997, Mecikalski et al. 1999), the operational Simplified Surface Energy Balance (SSEB; Senay et al. 2013), CSIRO MODIS Rescaled Evapotranspiration (CMRSET; Guerschman et al. 2009), and MOD16. The MOD16 algorithm is based on the logic of the Penman-Monteith equation, which uses daily meteorological reanalysis data and eight-day remotely sensed vegetation property dynamics from MODIS as inputs. The MOD16 product is available in the Earth Engine assets, and we will use this product for ET estimation in the next exercise.

First, we start a new script and repeat the first two steps of the previous section by copying and pasting the following code:

```
// Import the Lower Mekong boundary.
var mekongBasin = ee.FeatureCollection(
    'projects/gee-book/assets/A2-5/lowerMekongBasin');

// Center the map.
Map.centerObject(mekongBasin, 5);

// Add the Lower Mekong Basin boundary to the map.
Map.addLayer(mekongBasin, {}, 'Lower Mekong basin');

// Set start and end years.
var startYear = 2010;
var endYear = 2020;

// Create two date objects for start and end years.
var startDate = ee.Date.fromYMD(startYear, 1, 1);
var endDate = ee.Date.fromYMD(endYear + 1, 1, 1);

// Make a list with years.
var years = ee.List.sequence(startYear, endYear);

// Make a list with months.
var months = ee.List.sequence(1, 12);
```

We import the MOD16 dataset and select the ET band, which represents total evapotranspiration.

```
// Import the MOD16 dataset.
var mod16 = ee.ImageCollection('MODIS/006/MOD16A2').select('ET');

// Filter for the relevant time period.
mod16 = mod16.filterDate(startDate, endDate);
```

We use the same function to calculate monthly values as in the previous section. Note that we multiply by 0.1 as a scaling factor. The scaling factor can be found in the description of the dataset in Earth Engine. Scaling factors are applied to reduce the required storage capacity by changing the data type.

```
// We apply a nested loop where we first map over
// the relevant years and then map over the relevant
// months. The function returns an image with the total (sum)
// evapotranspiration for each month. A flatten is applied to convert
a
// collection of collections into a single collection.
// We multiply by 0.1 because of the ET scaling factor.
var monthlyEvap = ee.ImageCollection.fromImages(
    years.map(function(y) {
        return months.map(function(m) {
            var w = mod16.filter(ee.Filter
                    .calendarRange(y, y, 'year'))
                .filter(ee.Filter.calendarRange(m, m,
                    'month'))
                .sum()
                .multiply(0.1);
            return w.set('year', y)
                .set('month', m)
                .set('system:time_start', ee.Date
                    .fromYMD(y, m, 1));

        });
    }).flatten()
);
```

We use the code below to visualize the results. Note that we changed the color of the bar chart and applied the reducer on a 500 m spatial resolution.

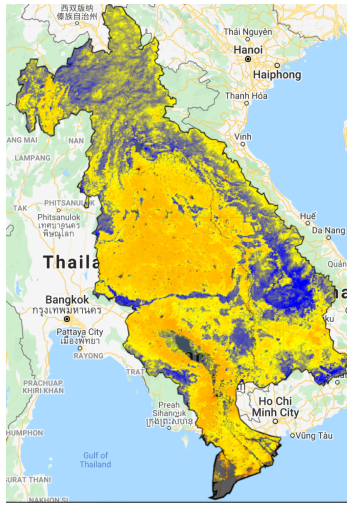
```
// Add the layer with monthly mean. Note that we clip for the Mekong
river basin.
var evapVis = {
    min: 0,
    max: 140,
    palette: 'red, orange, yellow, blue, darkblue'
};
```

```
Map.addLayer(monthlyEvap.mean().clip(mekongBasin),
    evapVis,
    'Mean monthly ET');

// Set the title and axis labels for the chart.
var title = {
    title: 'Monthly evapotranspiration',
    hAxis: {
        title: 'Time'
    },
    vAxis: {
        title: 'Evapotranspiration (mm)'
    },
    colors: ['red']
};

// Plot the chart using the Mekong boundary.
var chartMonthly = ui.Chart.image.seriesByRegion({
        imageCollection: monthlyEvap,
        regions: mekongBasin.geometry(),
        reducer: ee.Reducer.mean(),
        band: 'ET',
        scale: 500,
        xProperty: 'system:time_start'
    }).setSeriesNames(['ET'])
    .setOptions(title)
    .setChartType('ColumnChart');

// Print the chart.
print(chartMonthly);
```



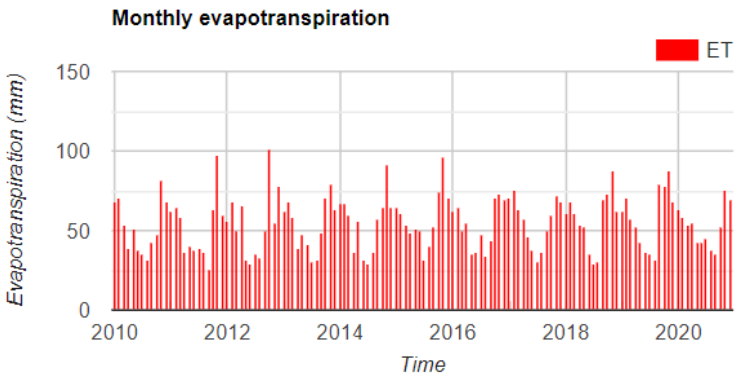


Fig. A2.5.7 Mean ET in the lower Mekong basin (left) and the monthly average ET (right)

Code Checkpoint A25b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Monthly Water Balance

We learned that the water balance is calculated using precipitation, evapotranspiration, runoff, and storage changes (Eq. A2.5.1). In the previous two sections, we calculated the monthly precipitation (P) on a 5 km spatial resolution and the monthly evapotranspiration (ET) on a 500 m spatial resolution. In Equation A2.5.2 we rearrange Equation 2.7.1 so that we can calculate the portion of Q and ΔS on a pixel level and aggregate that information to a basin level.

$$P - E = Q + \Delta S \qquad \text{(A2.5.2)}$$

In this section we will use the previous data to calculate the monthly water balance. First, we set the dates and import the relevant `ImageCollection`, as shown in the previous sections. Copy and paste the code below in a new script.

```
// Import the Lower Mekong boundary.
var mekongBasin = ee.FeatureCollection(
```

```
    'projects/gee-book/assets/A2-5/lowerMekongBasin');

// Center the map.
Map.centerObject(mekongBasin, 5);

// Add the Lower Mekong Basin boundary to the map.
Map.addLayer(mekongBasin, {}, 'Lower Mekong basin');

// Set start and end years.
var startYear = 2010;
var endYear = 2020;

// Create two date objects for start and end years.
var startDate = ee.Date.fromYMD(startYear, 1, 1);
var endDate = ee.Date.fromYMD(endYear + 1, 1, 1);

// Make a list with years.
var years = ee.List.sequence(startYear, endYear);

// Make a list with months.
var months = ee.List.sequence(1, 12);

// Import the CHIRPS dataset.
var CHIRPS = ee.ImageCollection('UCSB-CHG/CHIRPS/PENTAD');

// Filter for relevant time period.
CHIRPS = CHIRPS.filterDate(startDate, endDate);

// Import the MOD16 dataset.
var mod16 = ee.ImageCollection('MODIS/006/MOD16A2').select('ET');

// Filter for relevant time period.
mod16 = mod16.filterDate(startDate, endDate);
```

Now we use the function that we used earlier to calculate monthly ET and P to calculate the water balance.

```
// We apply a nested loop where we first map over
// the relevant years and then map over the relevant
// months. The function returns an image with P - ET
// for each month. A flatten is applied to convert an
// collection of collections into a single collection.
var waterBalance = ee.ImageCollection.fromImages(
    years.map(function(y) {
        return months.map(function(m) {

            var P = CHIRPS.filter(ee.Filter
                    .calendarRange(y, y, 'year'))
                .filter(ee.Filter.calendarRange(m, m,
                    'month'))
                .sum();

            var ET = mod16.filter(ee.Filter
                    .calendarRange(y, y, 'year'))
                .filter(ee.Filter.calendarRange(m, m,
                    'month'))
                .sum()
                .multiply(0.1);

            var wb = P.subtract(ET).rename('wb');

            return wb.set('year', y)
                .set('month', m)
                .set('system:time_start', ee.Date
                    .fromYMD(y, m, 1));

        });
    }).flatten()
);
```

Next we add the monthly mean water balance to the map and calculate the monthly water balance (Fig. A2.5.8). Note that negative numbers in the map indicate regions with an overall surplus of ET, whereas negative monthly water balances indicate a surplus ET for the whole region.

```
// Add layer with monthly mean. note that we clip for the Mekong river
basin.
var balanceVis = {
    min: -50,
    max: 200,
    palette: 'red, orange, yellow, blue, darkblue, purple'
};

Map.addLayer(waterBalance.mean().clip(mekongBasin),
    balanceVis,
    'Mean monthly water balance');

// Set the title and axis labels for the chart.
var title = {
    title: 'Monthly water balance',
    hAxis: {
        title: 'Time'
    },
    vAxis: {
        title: 'Evapotranspiration (mm)'
    },
    colors: ['green']
};

// Plot the chart using the Mekong boundary.
var chartMonthly = ui.Chart.image.seriesByRegion({
        imageCollection: waterBalance,
        regions: mekongBasin.geometry(),
        reducer: ee.Reducer.mean(),
        band: 'wb',
        scale: 500,
        xProperty: 'system:time_start'
    }).setSeriesNames(['WB'])
    .setOptions(title)
    .setChartType('ColumnChart');

// Print the chart.
```

```
print(chartMonthly);
```

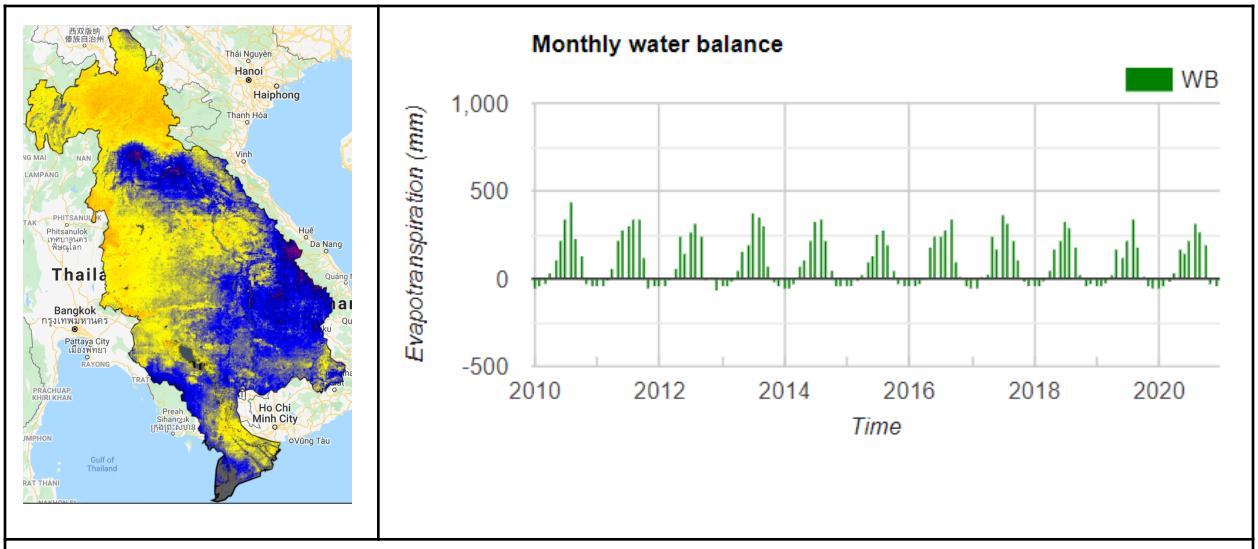


Fig. A2.5.8 Mean water balance in the lower Mekong basin (left) and the monthly average water balance (right)

Code Checkpoint A25c. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Vegetation and Drought Indices

Calculating vegetation indices is a common practice when working with remote sensing data. The Normalized Difference Vegetation Index (NDVI; Rouse et al. 1973) and Enhanced Vegetation Index (EVI; Chap. F3.1) (Huete et al. 1994) are among the most commonly used. Vegetation indices rely on the absorption and reflection spectra of chlorophyll, often including the red band, where absorption is high, and near infrared, where reflection is high. Vegetation indices are often used to measure crop health and density, but they can also be used to measure, for example, biophysical health over longer time periods (Poortinga et al. 2018). Vegetation indices can be calculated from the spectral reflectance, but readily available products can be used as well. The latter have been processed and contain, for example, corrections for outliers and artifacts.

Besides vegetation indices, there are many other indices to describe specific natural phenomena or detect specific land surface features. These indices often rely on simple band ratios or more sophisticated formulas containing multiple bands. Drought indices

are another category of important indicators as they enable us to depict spatiotemporal drought patterns in great detail. This can be particularly useful for remote areas where people rely on local agriculture for their livelihoods but production data is scarce. In the next exercise we will be using the Moisture Stress Index (MSI; Vogelmann 1985), as this index has been shown to be highly related to soil moisture. Eq. A2.5.3 shows that MSI is calculated from the shortwave infrared (SWIR) and near infrared (NIR) bands.

MSI = SWIR / NIR (A2.5.3)

In this section, we use the EVI product for the vegetation index from MODIS. This data is collected daily and can be used for calculation of the monthly vegetation index. You may note that we import the readily available EVI product and also import the spectral reflectance dataset.

We first import all the relevant datasets and define the period of interest, as in the other exercises, starting a new script.

```
// Import the Lower Mekong boundary.
var mekongBasin = ee.FeatureCollection(
    'projects/gee-book/assets/A2-5/lowerMekongBasin');

// Center the map.
Map.centerObject(mekongBasin, 5);

// Add the Lower Mekong Basin boundary to the map.
Map.addLayer(mekongBasin, {}, 'Lower Mekong basin');

// Set start and end years.
var startYear = 2010;
var endYear = 2020;

// Create two date objects for start and end years.
var startDate = ee.Date.fromYMD(startYear, 1, 1);
var endDate = ee.Date.fromYMD(endYear + 1, 1, 1);

// Make a list with years.
var years = ee.List.sequence(startYear, endYear);
```

```
// Make a list with months.
var months = ee.List.sequence(1, 12);

// Import the CHIRPS dataset.
var CHIRPS = ee.ImageCollection('UCSB-CHG/CHIRPS/PENTAD');

// Filter for relevant time period.
CHIRPS = CHIRPS.filterDate(startDate, endDate);

// Import the MOD16 dataset.
var mod16 = ee.ImageCollection('MODIS/006/MOD16A2').select('ET');

// Filter for relevant time period.
mod16 = mod16.filterDate(startDate, endDate);

// Import and filter the MOD13 dataset.
var mod13 = ee.ImageCollection('MODIS/006/MOD13A1');
mod13 = mod13.filterDate(startDate, endDate);

// Select the EVI.
var EVI = mod13.select('EVI');

// Import and filter the MODIS Terra surface reflectance dataset.
var mod09 = ee.ImageCollection('MODIS/006/MOD09A1');
mod09 = mod09.filterDate(startDate, endDate);
```

Two steps are needed to calculate the MSI. First, we need to remove the clouds and cloud shadows. This can be done by using the StateQA quality band that comes with the MOD09 product. After all the artifacts have been removed, we can calculate the MSI in the next function.

```
// We use a function to remove clouds and cloud shadows.
// We map over the mod09 image collection and select the StateQA band.
// We mask pixels and return the image with clouds and cloud shadows
masked.
mod09 = mod09.map(function(image) {
```

```
    var quality = image.select('StateQA');
    var mask = image.and(quality.bitwiseAnd(1).eq(
            0)) // No clouds.
        .and(quality.bitwiseAnd(2).eq(0)); // No cloud shadow.

    return image.updateMask(mask);
});

// We use a function to calculate the Moisture Stress Index.
// We map over the mod09 image collection and select the NIR and SWIR
bands
// We set the timestamp and return the MSI.
var MSI = mod09.map(function(image) {
    var nirband = image.select('sur_refl_b02');
    var swirband = image.select('sur_refl_b06');

    var msi = swirband.divide(nirband).rename('MSI')
        .set('system:time_start', image.get(
            'system:time_start'));
    return msi;
});
```

We use the same nested loop as the previous sections, but now we calculate all layers and return an image where the layers are included as bands. Note that EVI and MOD16 are multiplied by the scale factor. For P, ET, and WB we calculate the sum; for MSI and EVI we calculate the mean.

```
// We apply a nested loop where we first map over
// the relevant years and then map over the relevant
// months. The function returns an image with bands for
// water balance (wb), rainfall (P), evapotranspiration (ET),
// EVI and MSI for each month. A flatten is applied to
// convert an collection of collections
// into a single collection.
var ic = ee.ImageCollection.fromImages(
    years.map(function(y) {
        return months.map(function(m) {
```

```
        // Calculate rainfall.
        var P = CHIRPS.filter(ee.Filter
                .calendarRange(y, y, 'year'))
            .filter(ee.Filter.calendarRange(m, m,
                'month'))
            .sum();

        // Calculate evapotranspiration.
        var ET = mod16.filter(ee.Filter
                .calendarRange(y, y, 'year'))
            .filter(ee.Filter.calendarRange(m, m,
                'month'))
            .sum()
            .multiply(0.1);

        // Calculate EVI.
        var evi = EVI.filter(ee.Filter
                .calendarRange(y, y, 'year'))
            .filter(ee.Filter.calendarRange(m, m,
                'month'))
            .mean()
            .multiply(0.0001);

        // Calculate MSI.
        var msi = MSI.filter(ee.Filter
                .calendarRange(y, y, 'year'))
            .filter(ee.Filter.calendarRange(m, m,
                'month'))
            .mean();

        // Calculate monthly water balance.
        var wb = P.subtract(ET).rename('wb');

        // Return an image with all images as bands.
        return ee.Image.cat([wb, P, ET, evi, msi])
            .set('year', y)
            .set('month', m)
```

```
                .set('system:time_start', ee.Date
                    .fromYMD(y, m, 1));

        });
    }).flatten()
);
```

We display the monthly mean EVI and MSI and show the time series (Fig. A2.5.9).

```
// Add the mean monthly EVI and MSI to the map.
var eviVis = {
    min: 0,
    max: 0.7,
    palette: 'red, orange, yellow, green, darkgreen'
};

Map.addLayer(ic.select('EVI').mean().clip(mekongBasin),
    eviVis,
    'EVI');

var msiVis = {
    min: 0.25,
    max: 1,
    palette: 'darkblue, blue, yellow, orange, red'
};

Map.addLayer(ic.select('MSI').mean().clip(mekongBasin),
    msiVis,
    'MSI');

// Define the water balance chart and print it to the console.
var chartWB =
    ui.Chart.image.series({
        imageCollection: ic.select(['wb', 'precipitation', 'ET']),
        region: mekongBasin,
        reducer: ee.Reducer.mean(),
        scale: 5000,
```

```
        xProperty: 'system:time_start'
    })
    .setSeriesNames(['wb', 'P', 'ET'])
    .setOptions({
        title: 'water balance',
        hAxis: {
            title: 'Date',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        vAxis: {
            title: 'Water (mm)',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        lineWidth: 1,
        colors: ['green', 'blue', 'red'],
        curveType: 'function'
    });

// Print the water balance chart.
print(chartWB);

// Define the indices chart and print it to the console.
var chartIndices =
    ui.Chart.image.series({
        imageCollection: ic.select(['EVI', 'MSI']),
        region: mekongBasin,
        reducer: ee.Reducer.mean(),
        scale: 5000,
        xProperty: 'system:time_start'
    })
    .setSeriesNames(['EVI', 'MSI'])
```

```
    .setOptions({
        title: 'Monthly indices',
        hAxis: {
            title: 'Date',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        vAxis: {
            title: 'Index',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        lineWidth: 1,
        colors: ['darkgreen', 'brown'],
        curveType: 'function'
    });

// Print the indices chart.
print(chartIndices);
```

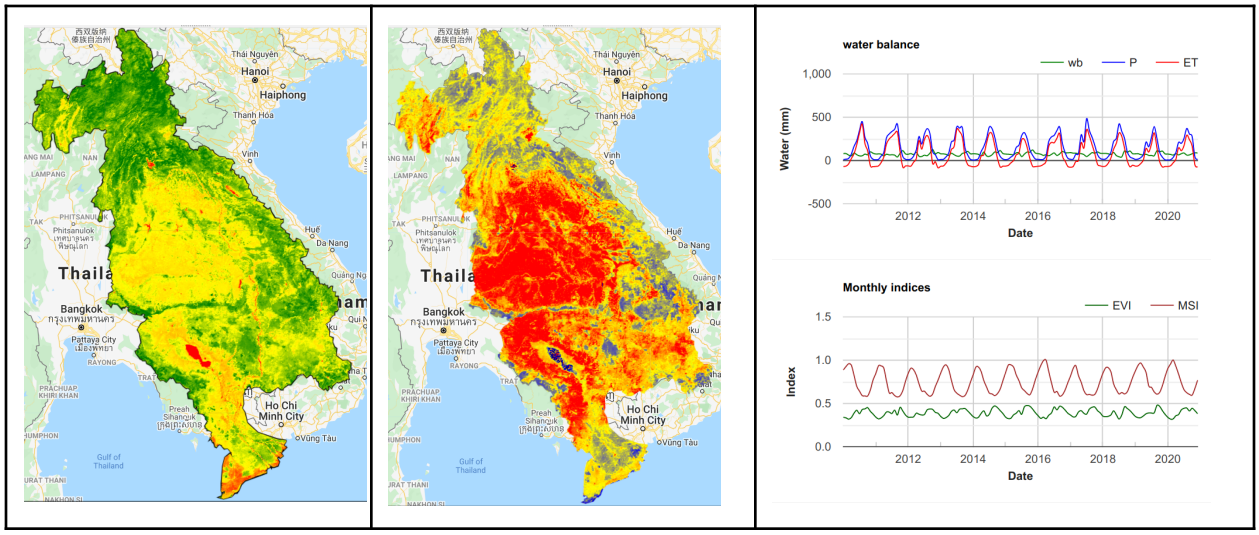


Fig. A2.5.9 Monthly mean EVI (left), monthly mean MSI (middle), and the time series for the water balance (top right) and indices (bottom right)

Code Checkpoint A25d. The book's repository contains a script that shows what your code should look like at this point.

Section 5. Partitioning Water Resources and Mapping Drought Impacts

Historical and near-real-time remote-sensing-derived data can be very useful to assess the impact on the ground. For example, overlaying information layers on water resources with land cover information enables us to partition water resources by land cover and investigate consumptive use of, for example, different agricultural commodities. Overlaying land cover with vegetation and drought indicators helps to investigate land cover categories that are most impacted by water shortage. It also helps to assess the impacts on crop production and food security (Poortinga et al. 2019), as well as potential environmental impacts, including impacts on biodiversity. However, in many cases maps are produced and updated only infrequently and are often not accompanied by appropriate accuracy assessment information and documentation (Saah et al. 2019). Therefore, SERVIR-Mekong developed a time series of yearly land cover maps covering the greater Mekong region, including the Lower Mekong Basin (Saah et al. 2020, Potapov et al. 2019, Poortinga et al. 2020). Fig. A2.5.10 shows the land cover map of 2018.

Section 5.1. Annual Classifications

In the following subsections, we illustrate annual land cover maps and a variety of interpretations of them. The annual maps can be accessed using script **A25s1 - Annual** in the book repository.

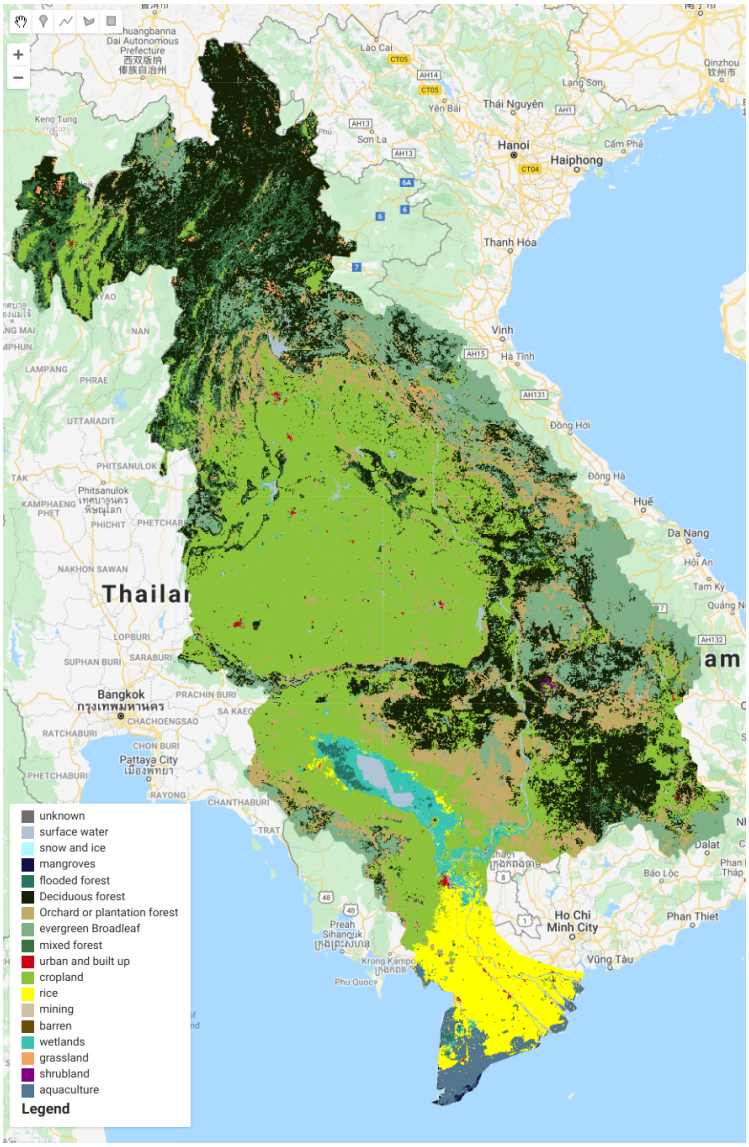


Fig. A2.5.10 Land cover map of the Lower Mekong Basin

Note: if you get an error related to "too many concurrent aggregations" in this last script link or the following ones, try re-running the script.

Section 5.2. Precipitation and Evapotranspiration

An example of partitioning water inputs and consumption is shown in Fig. A2.5.11. Here we calculate the mean P and ET for each land cover category using the land cover map. We can see that the largest portions of water are being used by cropland and agriculture; plantations also consume a large portion of the water. The pie charts can be created and viewed using script **A25s2 - PET** in the book repository.

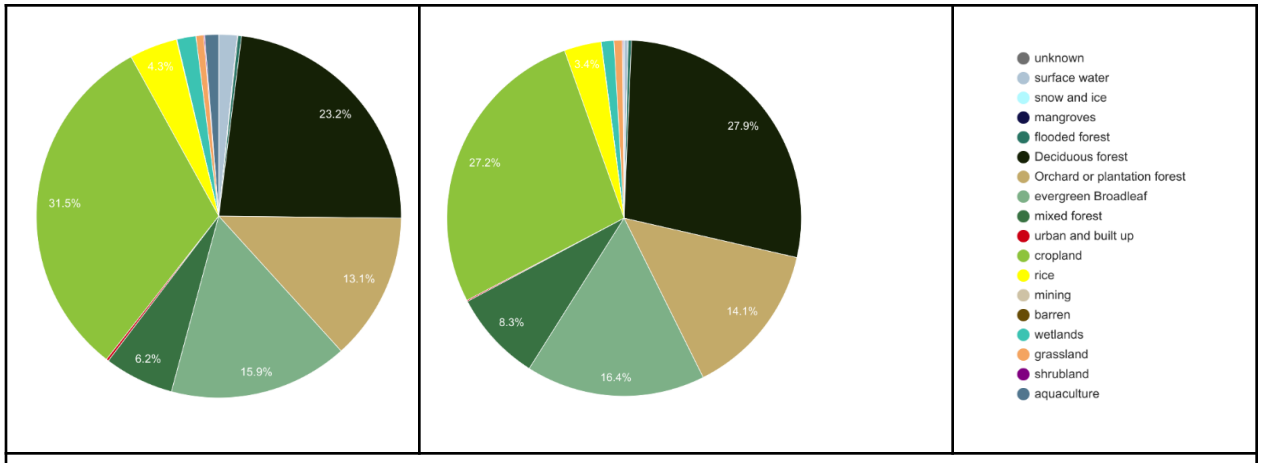


Fig. A2.5.11 The amount of P (left) and ET (right) per land cover category

Section 5.3. Monthly Water Balance

We can apply a similar overlaying method of partitioning for the water balance. Fig. A2.5.12 shows the monthly water balance for deciduous forest, evergreen broadleaf, cropland, and rice. It can be seen that a large portion of the water stored in the wet season is used by forest and cropland in the dry season. The area of paddy rice is smaller, so the total water consumption in the dry season of those categories is smaller. The monthly water balance charts can be found in script **A25s3 - Monthly** in the book repository.

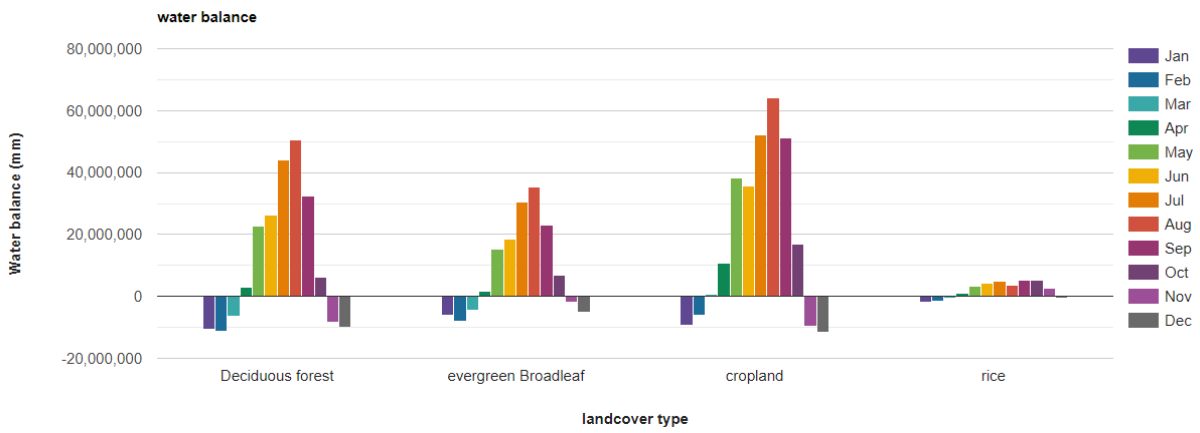
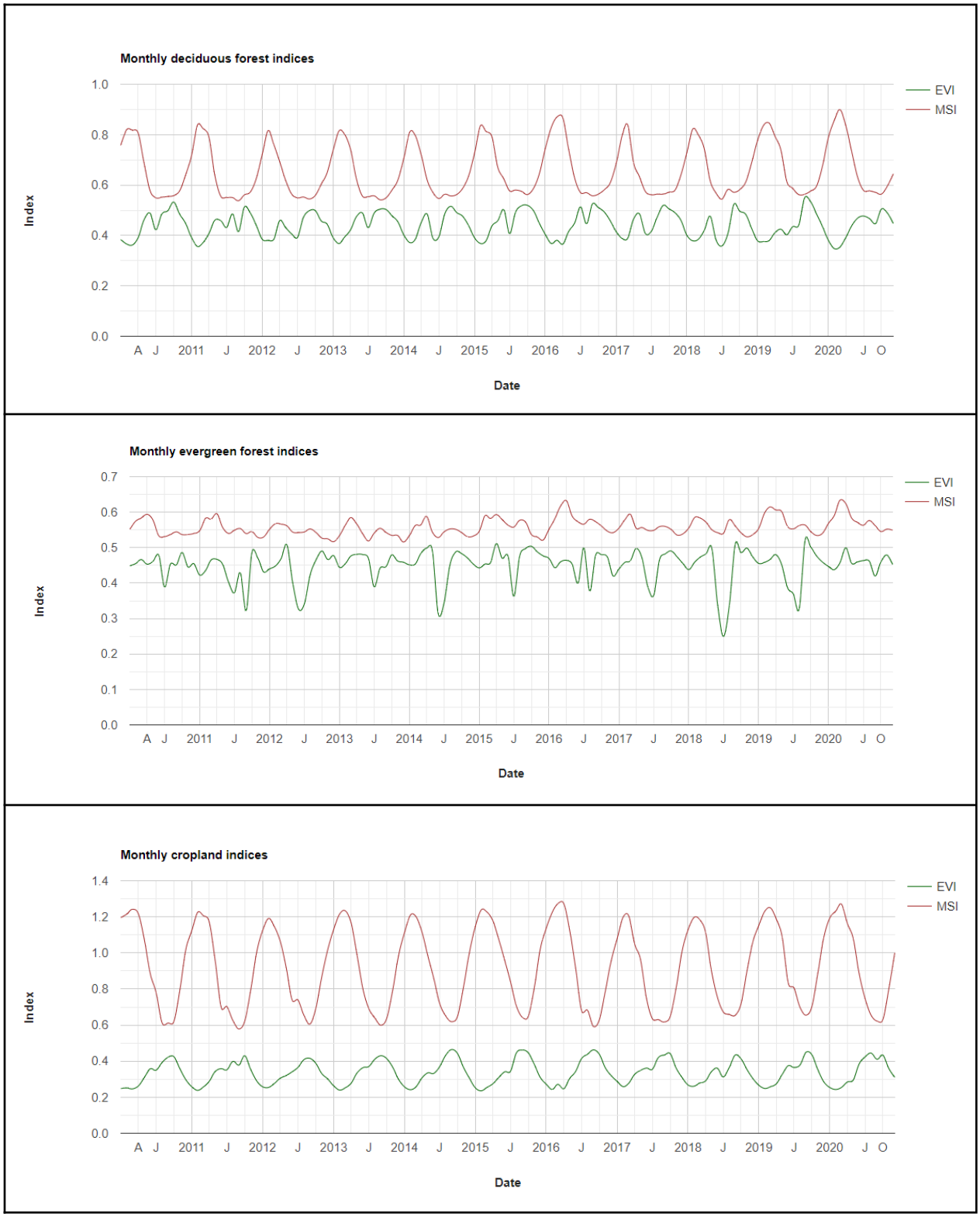


Fig. A2.5.12 Water balance for four different land cover categories: deciduous forest, evergreen broadleaf, cropland, and rice

Section 5.4. Per-class Water Balance Across Seasons

Partitioning can also be done per land cover category. In Fig. A2.5.13, we calculated the EVI and MSI for four land cover categories. There are very distinct patterns: we see little variation in the signal from the EVI, but large variations for the deciduous forest. For cropland, we found a signal that closely corresponds to the yearly dry and wet seasons, whereas we see multiple cropping seasons per year. The long time series enables us to investigate deviations from the mean, which in turn provides valuable information on, for example, potential drought impacts. The per-class water balance charts can be found in script **A25s4 - Per Class Balance** in the book repository.



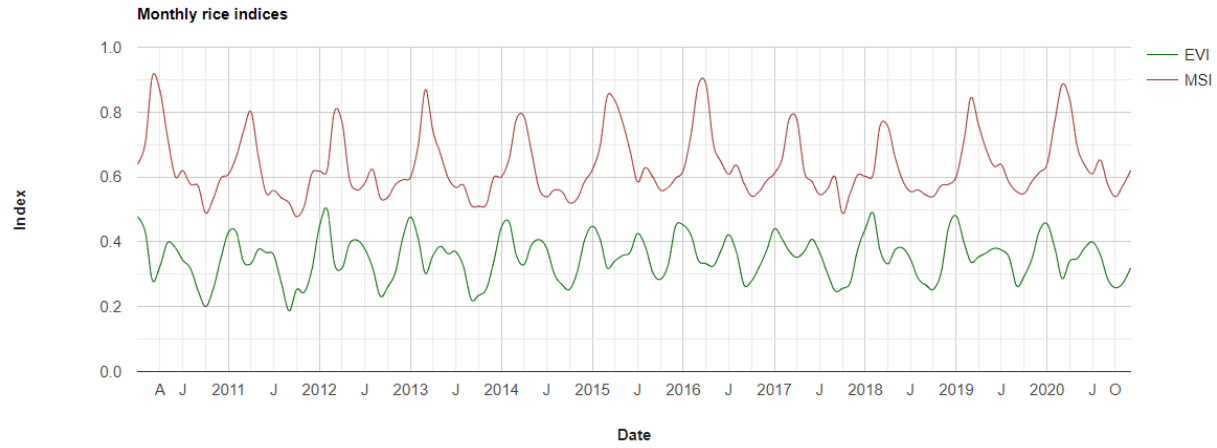


Fig. A2.5.13 EVI and MSI per land cover category

Synthesis

With what you learned in this chapter, you can analyze large-scale hydrological processes in a river basin. The approach can be applied for any river basin in the world using your own data or the open access data in the exercises.

Assignment 1. Test the approach in another part of the world using your own data or open-access data, or use your own training data for a more refined classification model.

Assignment 2. For further analysis, we encourage you to (1) replace MODIS with data from the Visible Infrared Imaging Radiometer Suite (VIIRS); (2) replace the CHIRPS data with the Integrated Multi-satellite Retrievals for GPM (IMERG) data and change the time intervals; and (3) use a different land cover map for partitioning the water resources.

Conclusion

A safe and sustainable supply of water is essential for drinking, washing, cleaning, cooking, and growing food. However, water is often a scarce resource that needs to be managed in a sustainable and equitable way. Satellite data products in Earth Engine can help us to map the quantity of water in space and time. It enables us to partition water according to its consumptive use and evaluate how this affects a wide variety of important functions within a water basin.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Anderson MC, Norman JM, Diak GR, et al (1997) A two-source time-integrated model for estimating surface fluxes using thermal infrared remote sensing. Remote Sens Environ 60:195–216. https://doi.org/10.1016/S0034-4257(96)00215-5

Courault D, Seguin B, Olioso A (2005) Review on estimation of evapotranspiration from remote sensing data: From empirical to numerical modeling approaches. Irrig Drain Syst 19:223–249. https://doi.org/10.1007/s10795-005-5186-0

Funk C, Peterson P, Landsfeld M, et al (2015) The climate hazards infrared precipitation with stations – a new environmental record for monitoring extremes. Sci Data 2:1–21. https://doi.org/10.1038/sdata.2015.66

Guerschman JP, Van Dijk AIJM, Mattersdorf G, et al (2009) Scaling of potential evapotranspiration with MODIS data reproduces flux observations and catchment water balance observations across Australia. J Hydrol 369:107–119. https://doi.org/10.1016/j.jhydrol.2009.02.013

Hou AY, Kakar RK, Neeck S, et al (2014) The global precipitation measurement mission. Bull Am Meteorol Soc 95:701–722. https://doi.org/10.1175/BAMS-D-13-00164.1

Huete A, Justice C, Liu H (1994) Development of vegetation and soil indices for MODIS-EOS. Remote Sens Environ 49:224–234. https://doi.org/10.1016/0034-4257(94)90018-3

Kummerow C, Barnes W, Kozu T, et al (1998) The tropical rainfall measuring mission (TRMM) sensor package. J Atmos Ocean Technol 15:809–817. https://doi.org/10.1175/1520-0426(1998)015<0809:TTRMMT>2.0.CO;2

Kustas WP, Norman JM (1996) Use of remote sensing for evapotranspiration monitoring over land surfaces. Hydrol Sci J 41:495–516. https://doi.org/10.1080/02626669609491522

Mecikalski JR, Diak GR, Anderson MC, Norman JM (1999) Estimating fluxes on continental scales using remotely sensed data in an atmospheric-land exchange model. J Appl Meteorol 38:1352–1369. https://doi.org/10.1175/1520-0450(1999)038<1352:EFOCSU>2.0.CO;2

Poortinga A, Aekakkararungroj A, Kityuttachai K, et al (2020) Predictive analytics for identifying land cover change hotspots in the Mekong region. Remote Sens 12:1472. https://doi.org/10.3390/RS12091472

Poortinga A, Clinton N, Saah D, et al (2018) An operational before-after-control-impact (BACI) designed platform for vegetation monitoring at planetary scale. Remote Sens 10:760. https://doi.org/10.3390/rs10050760

Poortinga A, Nguyen Q, Tenneson K, et al (2019) Linking Earth observations for assessing the food security situation in Vietnam: A landscape approach. Front Environ Sci 7:186. https://doi.org/10.3389/fenvs.2019.00186

Potapov P, Tyukavina A, Turubanova S, et al (2019) Annual continuous fields of woody vegetation structure in the Lower Mekong region from 2000-2017 Landsat time-series. Remote Sens Environ 232:111278. https://doi.org/10.1016/j.rse.2019.111278

Rouse Jr JW, Haas RH, Schell JA, Deering DW (1973) Paper a 20. In: Third Earth Resources Technology Satellite-1 Symposium: Section AB. Technical presentations. pp 309

Saah D, Tenneson K, Matin M, et al (2019) Land cover mapping in data scarce environments: Challenges and opportunities. Front Environ Sci 7:150. https://doi.org/10.3389/fenvs.2019.00150

Saah D, Tenneson K, Poortinga A, et al (2020) Primitives as building blocks for constructing land cover maps. Int J Appl Earth Obs Geoinf 85:101979. https://doi.org/10.1016/j.jag.2019.101979

Senay GB, Bohms S, Singh RK, et al (2013) Operational evapotranspiration mapping using remote sensing and weather datasets: A new parameterization for the SSEB approach. J Am Water Resour Assoc 49:577–591. https://doi.org/10.1111/jawr.12057

Strangeways I (2010) A history of rain gauges. Weather 65:133–138. https://doi.org/10.1002/wea.548

Vogelmann JE, Rock BN (1985) Spectral characterization of suspected acid deposition damage in red spruce (Picea Rubens) stands from Vermont. In: Proc. of the Airborne Imaging Spectrometer Data Anal. Workshop.

Chapter A2.6: Defining Seasonality: First Date of No Snow

Authors
Amanda Armstrong, Morgan Tassone, Justin Braaten

Overview
The purpose of this chapter is to demonstrate how to produce annual maps representing the first day within a year on which a given pixel reaches 0% snow cover. It also provides suggestions for summarizing and visualizing the results over time and space.

Learning Outcomes
- Generating and using a date band in image compositing.
- Applying temporal filtering to an `ImageCollection`.
- Identifying patterns of seasonal snowmelt.

Helps if you know how to:
- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Work with time-series data in Earth Engine (Part F4).
- Fit linear and nonlinear functions with regression in an ImageCollection time series (Chap. F4.6).

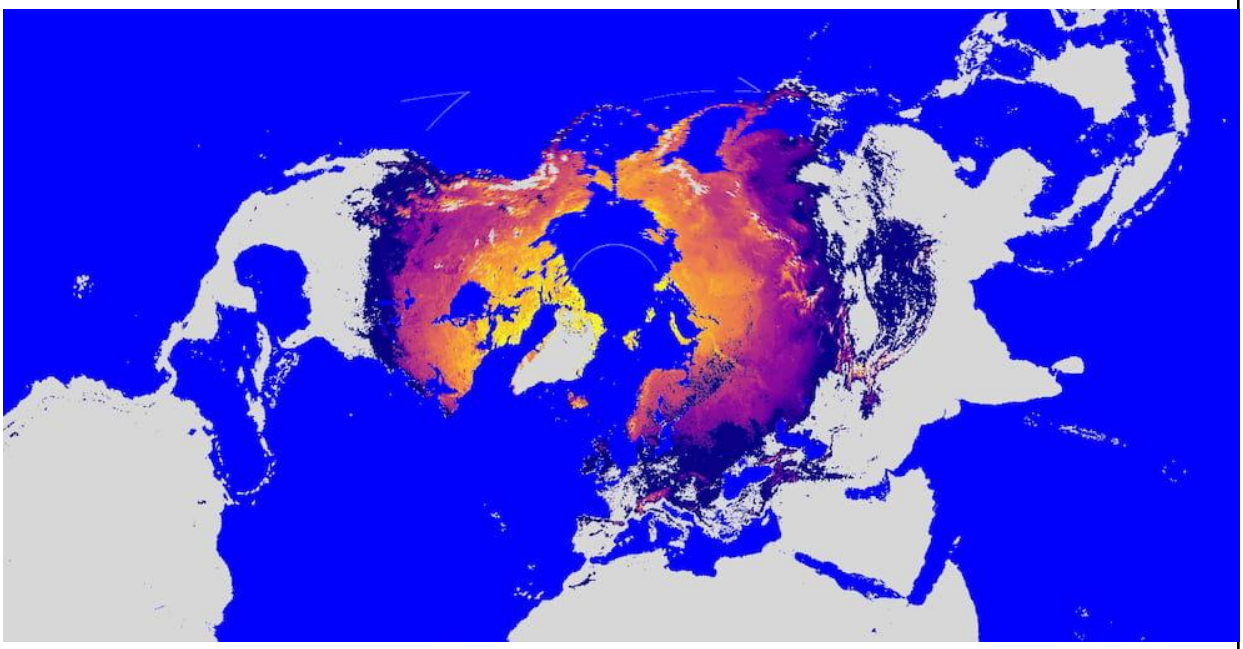


Fig. A2.6.1 Arctic polar stereographic projection showing the pattern of snowmelt timing in the Northern Hemisphere. The image shows the first day in 2018 on which each pixel no longer contained snow, as detected by the Moderate Resolution Imaging Spectroradiometer (MODIS) Snow Cover Daily Global product. The color grades from purple (earlier) to yellow (later).

Introduction to Theory

The timing of annual seasonal snowmelt (Fig. A2.6.1) and any potential change in that timing have broad ecological implications and thus impact human livelihoods, particularly in and around high-latitude and mountainous systems. The annual melting of accumulated winter snowfall, one of the most important phases of the hydrologic cycle within these regions, provides the dominant source of water for streamflow and groundwater recharge for approximately one sixth of the global population (Musselman et al. 2017, Barnhart et al. 2016, Bengtsson 1976). The timing of snowmelt in the Arctic and Antarctic influences the length of the growing season, and consistent snow cover throughout the winter insulates vegetation from harsh temperatures and wind (Duchesne

et al. 2012, Kudo et al. 1999). In mountainous regions, such as the Himalayas, snowmelt is a major source of freshwater downstream (Barnhart et al. 2016) and is essential in recharging groundwater.

This seasonal water resource is one of the fastest-changing hydrologic systems under Earth's warming climate, and these changes will broadly impact regional economies and ecosystem functioning, and increase the potential for flood hazards (Musselman et al. 2017, IPCC 2007, Beniston 2012, Allan and Casillo 2007, Barnett and Lettenmaier 2005). An analysis focusing on the Yamal Peninsula in the northwestern Siberian tundra found that the timing of snowmelt (calculated using the methods outlined here) was an important predictor of differences in ecosystem functioning across the landscape (Tassone et al., in prep).

The anticipated warmer temperatures will alter the type and onset of precipitation. Multiple regions, including the Rocky Mountains of North America, have already measured a reduction in snowpack volume, and warmer temperatures have shifted precipitation from snowfall to rain, causing snowmelt to occur earlier (Barnhart et al. 2016, Clow 2010, Harpold et al. 2012).

This tutorial demonstrates how to calculate the first day of no snow annually at the pixel level, providing the user with the ability to track the seasonal and interannual variability in the timing of snowmelt toward a better understanding of how the hydrological cycles of higher-latitude and mountainous regions are responding to climate change.

Practicum

Section 1. Identifying the First Day of 0% Snow Cover

This section covers building an `ImageCollection` where each image is a mosaic of pixels that describe the first day in a given year that 0% snow cover was recorded. Snow cover is defined by the MODIS Normalized Difference Snow Index (NDSI) Snow Cover Daily Global product. The general workflow is as follows.

1. Define the date range to consider for analysis.
2. Define a function that adds date information as bands to snow cover images.
3. Define an analysis mask.
4. For each year:
    a. Filter the `ImageCollection` to observations from the given year.
    b. Add date bands to the filtered images.

c. Identify the first day of the year without any snow per pixel.
d. Apply an analysis mask to the image mosaic.
e. Summarize the findings with a series of visualizations.

Section 1.1. Define the Date Range

First, we specify the day of year (DOY) on which to start the search for the first day with 0% snow cover. For applications in the Northern Hemisphere, you will likely want to start with January 1 (DOY 1). However, if you are studying snowmelt timing in the Southern Hemisphere (e.g., the Andes), where snowmelt can occur on dates on either side of January 1, it is more appropriate to start the year on July 1 (DOY 183), for instance. In this calculation, a year is defined as the 365 days beginning from the specified `startDoy`.

```
var startDoy = 1;
```

Then, we define the years to start and end tracking snow-cover fraction. All years in the range will be included in the analysis.

```
var startYear = 2000;
var endYear = 2019;
```

Section 1.2. Define the the Date Bands

Next, we will define a function to add several date bands to the images; the added bands will be used in a future step. Each image has a metadata timestamp property, but since we will be creating annual image mosaics composed of pixels from many different images, the date needs to be encoded per pixel as a value in an image band so that it is retained in the final mosaics. The function encodes:

- Calendar DOY (`calDoy`): enumerated DOY from January 1.
- Relative DOY (`relDoy`): enumerated DOY from a given `startDoy`.
- Milliseconds elapsed since the Unix epoch (`millis`).
- Year (`year`): Note that the year is tied to the `startDoy`. For example, if the `startDoy` is set at 183, the analysis will cross into the next calendar year, and the `year` given to all pixels will be the earlier year, even if a particular image was collected on or after January 1 of the subsequent year.

Additionally, two global variables are initialized (`startDate` and `startYear`) that will be redefined iteratively in a subsequent step.

```
var startDate;
var startYear;

function addDateBands(img) {
    // Get image date.
    var date = img.date();
    // Get calendar day-of-year.
    var calDoy = date.getRelative('day', 'year');
    // Get relative day-of-year; enumerate from user-defined startDoy.
    var relDoy = date.difference(startDate, 'day');
    // Get the date as milliseconds from Unix epoch.
    var millis = date.millis();
    // Add all of the above date info as bands to the snow fraction
image.
    var dateBands = ee.Image.constant([calDoy, relDoy, millis,
            startYear
        ])
        .rename(['calDoy', 'relDoy', 'millis', 'year']);
    // Cast bands to correct data type before returning the image.
    return img.addBands(dateBands)
        .cast({
            'calDoy': 'int',
            'relDoy': 'int',
            'millis': 'long',
            'year': 'int'
        })
        .set('millis', millis);
}
```

Section 1.3. Define an Analysis Mask

Here is the opportunity to define a mask for your analysis. This mask can be used to constrain the analysis to certain latitudes, land cover types, geometries, etc. In this case, we will (1) mask out water so that the analysis is confined to pixels over landforms only,

(2) mask out pixels that have very few days of snow cover, and (3) mask out pixels that are snow covered for a good deal of the year (e.g., glaciers).

Import the MODIS Land/Water Mask dataset, select the `'water_mask'` band, and set all land pixels to value 1.

```
var waterMask = ee.Image('MODIS/MOD44W/MOD44W_005_2000_02_24')
    .select('water_mask')
    .not();
```

Import the MODIS Snow Cover Daily Global 500 m product and select the `'NDSI_Snow_Cover'` band.

```
var completeCol = ee.ImageCollection('MODIS/006/MOD10A1')
    .select('NDSI_Snow_Cover');
```

Mask pixels based on the frequency of snow cover.

```
// Pixels must have been 10% snow covered for at least 2 weeks in
2018.
var snowCoverEphem = completeCol.filterDate('2018-01-01',
        '2019-01-01')
    .map(function(img) {
        return img.gte(10);
    })
    .sum()
    .gte(14);

// Pixels must not be 10% snow covered more than 124 days in 2018.
var snowCoverConst = completeCol.filterDate('2018-01-01',
        '2019-01-01')
    .map(function(img) {
        return img.gte(10);
    })
    .sum()
    .lte(124);
```

Combine the water mask and the snow cover frequency masks.

```
var analysisMask = waterMask.multiply(snowCoverEphem).multiply(
    snowCoverConst);
```

Section 1.4. Identify the First Day of the Year Without Snow Per Pixel, Per Year

Make a list of the years to process. The input variables were defined in Sect. 1.1.

```
var years = ee.List.sequence(startYear, endYear);
```

Map the following function over the list of years. For each year, identify the first day with 0% snow cover.

1. Define the start and end dates to filter the dataset for the given year.
2. Filter the `ImageCollection` by the date range.
3. Add the date bands to each image in the filtered collection.
4. Sort the filtered collection by date. (Note: To determine the first day with snow accumulation in the fall, reverse-sort the filtered collection.)
5. Make a mosaic using the `min` reducer to select the pixel with 0 (minimum) snow cover. Since the collection is sorted by date, the first image with 0 snow cover is selected. This operation is conducted per-pixel to build the complete image mosaic.
6. Apply the analysis mask to the resulting mosaic.

An `ee.List` of images is returned.

```
var annualList = years.map(function(year) {
    // Set the global startYear variable as the year being worked on
so that
    // it will be accessible to the addDateBands mapped to the
collection below.
    startYear = year;
    // Get the first day-of-year for this year as an ee.Date object.
    var firstDoy = ee.Date.fromYMD(year, 1, 1);
    // Advance from the firstDoy to the user-defined startDay;
subtract 1 since
    // firstDoy is already 1. Set the result as the global startDate
variable so
```

```
    // that it is accessible to the addDateBands mapped to the
collection below.
    startDate = firstDoy.advance(startDoy - 1, 'day');
    // Get endDate for this year by advancing 1 year from startDate.
    // Need to advance an extra day because end date of filterDate()
function
    // is exclusive.
    var endDate = startDate.advance(1, 'year').advance(1,
        'day');
    // Filter the complete collection by the start and end dates just
defined.
    var yearCol = completeCol.filterDate(startDate, endDate);
    // Construct an image where pixels represent the first day within
the date
    // range that the lowest snow fraction is observed.
    var noSnowImg = yearCol
        // Add date bands to all images in this particular collection.
        .map(addDateBands)
        // Sort the images by ascending time to identify the first day
without
        // snow. Alternatively, you can use .sort('millis', false) to
        // reverse sort (find first day of snow in the fall).
        .sort('millis')
        // Make a mosaic composed of pixels from images that represent
the
        // observation with the minimum percent snow cover (defined by
the
        // NDSI_Snow_Cover band); include all associated bands for the
selected
        // image.
        .reduce(ee.Reducer.min(5))
        // Rename the bands - band names were altered by previous
operation.
        .rename(['snowCover', 'calDoy', 'relDoy', 'millis',
            'year'
        ])
        // Apply the mask.
```

```
        .updateMask(analysisMask)
        // Set the year as a property for filtering by later.
        .set('year', year);

    // Mask by minimum snow fraction - only include pixels that reach
0
    // percent cover. Return the resulting image.
    return noSnowImg.updateMask(noSnowImg.select('snowCover')
        .eq(0));
});
```

Convert the `ee.List` of images to an `ImageCollection`.

```
var annualCol = ee.ImageCollection.fromImages(annualList);
```

Code Checkpoint A26a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Data Summary and Visualization

The following is a series of examples for how to display and explore the "first DOY with no snow" dataset you just generated.

- These examples refer to the calendar date (`calDoy` band) when displaying and incorporating date information in calculations. If you are using a date range that begins on any day other than January 1 (DOY 1) you may want to replace `calDoy` with `relDoy` in all cases below.
- Results may appear different as you zoom in and out of the **Map** because of tile aggregation, which is described in the Earth Engine documentation. It is best to view **Map** data interactively with a relatively high zoom level. Additionally, for any analysis where a function provides a scale parameter (e.g., region reduction, exporting results), it is best to define it with the native resolution of the dataset (500 m).
- MODIS cloud masking can influence results. If there are a number of sequentially masked image pixel observations (e.g., clouds, poor quality), the actual date of the first observation with 0% snow cover may be earlier than identified in the image time series. Regional patterns may be less influenced by this bias than

local results. For local results, please inspect image masks to understand their influence on the dates near snowmelt timing.

Section 2.1. Single-Year Map

Filter a single year from the collection (2018 in the example below) and display the image to the **Map** to see spatial patterns of snowmelt timing. Setting the `min` and `max` parameters of the `visArgs` variable to a narrow range around expected snowmelt timing is important for getting a good color stretch.

```
// Define a year to visualize.
var thisYear = 2018;

// Define visualization arguments.
var visArgs = {
    bands: ['calDoy'],
    min: 150,
    max: 200,
    palette: [
        '0D0887', '5B02A3', '9A179B', 'CB4678', 'EB7852',
        'FBB32F', 'F0F921'
    ]
};

// Subset the year of interest.
var firstDayNoSnowYear = annualCol.filter(ee.Filter.eq('year',
    thisYear)).first();

// Display it on the map.
Map.setCenter(-95.78, 59.451, 5);
Map.addLayer(firstDayNoSnowYear, visArgs,
    'First day of no snow, 2018');
```

Running this code produces something similar to Fig. A2.6.2. The color represents the DOY when 0% snow cover was first observed per pixel (blue is earlier, yellow is later).

One can notice a number of interesting patterns. Frozen lakes have been shown to decrease air temperatures in adjacent pixels, resulting in delayed snowmelt (Rouse et al. 1997, Salomonson and Appel 2004, Wang and Derksen 2007). Additionally, the

protected estuaries of the Northwest Passages have earlier dates of no snow compared to the landscapes exposed to the currents and winds of the Northern Atlantic. Latitude, elevation, and proximity to ocean currents are the strongest determinants in this region.

Note that pixels representing glaciers that did not get removed by the analysis mask can produce anomalies in the data. Since glaciers are generally snow covered, the DOY with the least snow cover according to the MODIS Snow Cover Daily Global product is presented in the **Map**. In Fig. A2.6.2, this is evident in the abrupt transition within alpine areas of Baffin Island (white pixels represent glaciers in this case).

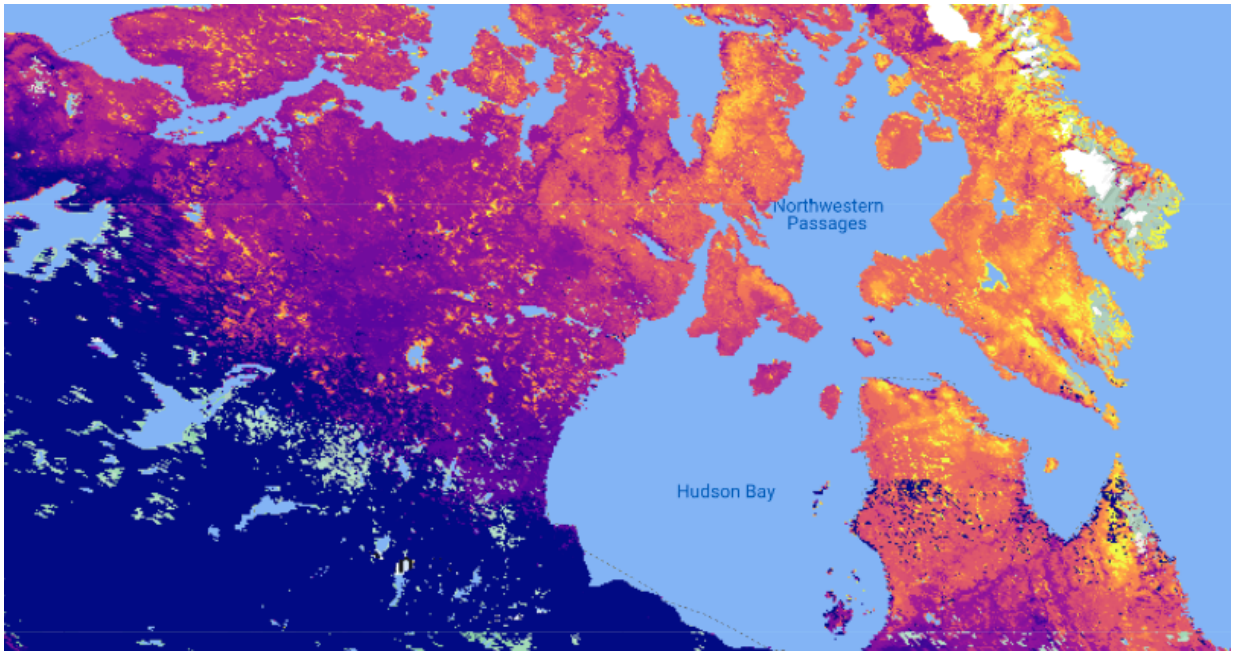


Fig. A2.6.*2* Thematic map representing the first DOY with 0% snow cover. Color grades from blue to yellow as the DOY increases.

Section 2.2. Year-to-Year Difference Map

Compare year-to-year difference in snowmelt timing by selecting two years of interest from the collection and subtracting them. Here, we're calculating the difference in snowmelt timing between 2005 and 2015.

```
// Define the years to difference.
var firstYear = 2005;
var secondYear = 2015;

// Calculate difference image.
var firstImg = annualCol.filter(ee.Filter.eq('year', firstYear))
    .first().select('calDoy');
var secondImg = annualCol.filter(ee.Filter.eq('year', secondYear))
    .first().select('calDoy');
var dif = secondImg.subtract(firstImg);

// Define visualization arguments.
var visArgs = {
    min: -15,
    max: 15,
    palette: ['b2182b', 'ef8a62', 'fddbc7', 'f7f7f7', 'd1e5f0',
        '67a9cf', '2166ac'
    ]
};

// Display it on the map.
Map.setCenter(95.427, 29.552, 8);
Map.addLayer(dif, visArgs, '2015-2005 first day no snow dif');
```

Running this code produces something similar to Fig. A2.6.3. The color represents the difference, in each pixel, between the 2005 and 2015 DOY when 0% snow cover was first observed. Red represents a negative change (an earlier no-snow date in 2015), blue represents a positive change (a later no-snow date in 2015), and white represents a negligible or no change in the no-snow dates for 2005 and 2015.

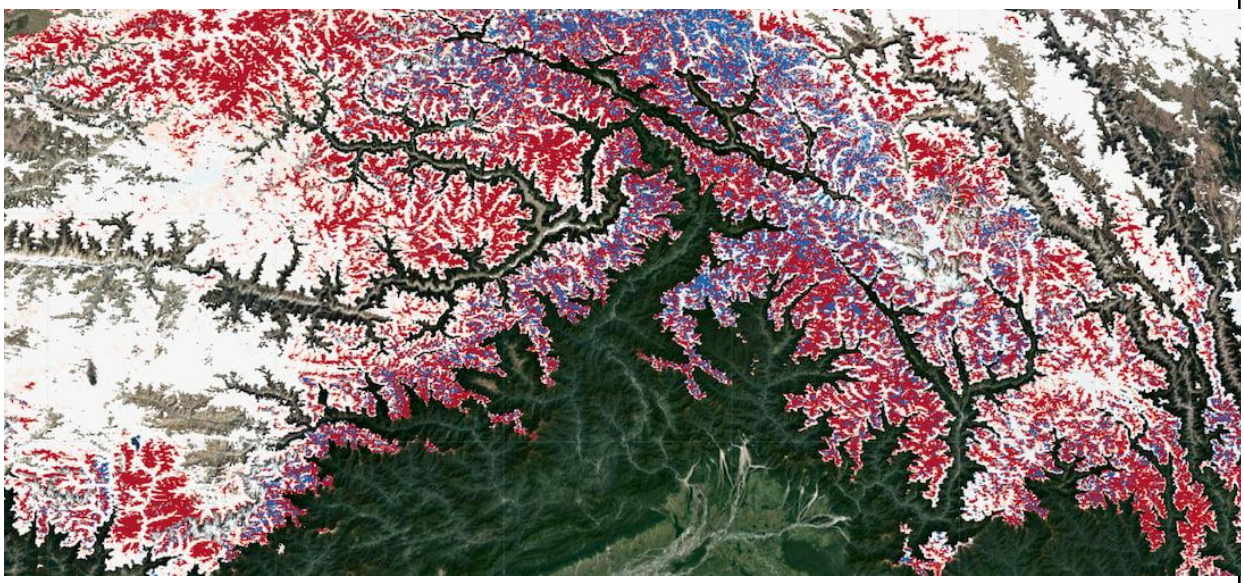


Fig. A2.6.3 Year-to-year (2005–2015) difference map of the Himalayas on the Nepal–China border. Color grades from red to blue, with red indicating an earlier date of no snow in 2015 and blue indicating a later date of no snow in 2015. White areas indicate little or no change.

Section 2.3. Trend Analysis Mapping

It is also possible to identify trends in the shifting first DOY with no snow by calculating the slope through a pixel's time-series points. Here, the slope for each pixel is calculated with year as the *x* variable and the first DOY with no snow as the *y* variable.

```
// Calculate slope image.
var slope = annualCol.sort('year').select(['year', 'calDoy'])
    .reduce(ee.Reducer.linearFit()).select('scale');

// Define visualization arguments.
var visArgs = {
    min: -1,
```

```
    max: 1,
    palette: ['b2182b', 'ef8a62', 'fddbc7', 'f7f7f7',
        'd1e5f0', '67a9cf', '2166ac'
    ]
};

// Display it on the map.
Map.setCenter(11.25, 59.88, 6);
Map.addLayer(slope, visArgs, '2000-2019 first day no snow slope');
```

The result is a map (Fig. A2.6.4) where red represents a negative slope (progressively earlier first DOY with no snow), white represents a slope of 0, and blue represents a positive slope (progressively later first DOY with no snow). In southern Norway and Sweden, the trend in the first DOY with no snow between 2002 and 2019 appears to be influenced by various factors. Coastal areas exhibit progressively earlier first DOY of no snow than inland areas; however, high variability in slopes can be observed around fjords and in mountainous regions. This map reveals the complexity of seasonal snow dynamics in these areas.

Note that goodness-of-fit calculations are not measured here, nor is significance considered. While a slope can indicate regional trends, more local trends should be investigated using a time-series chart (see the next section). Interannual variability can be influenced by masked pixels, as described above.

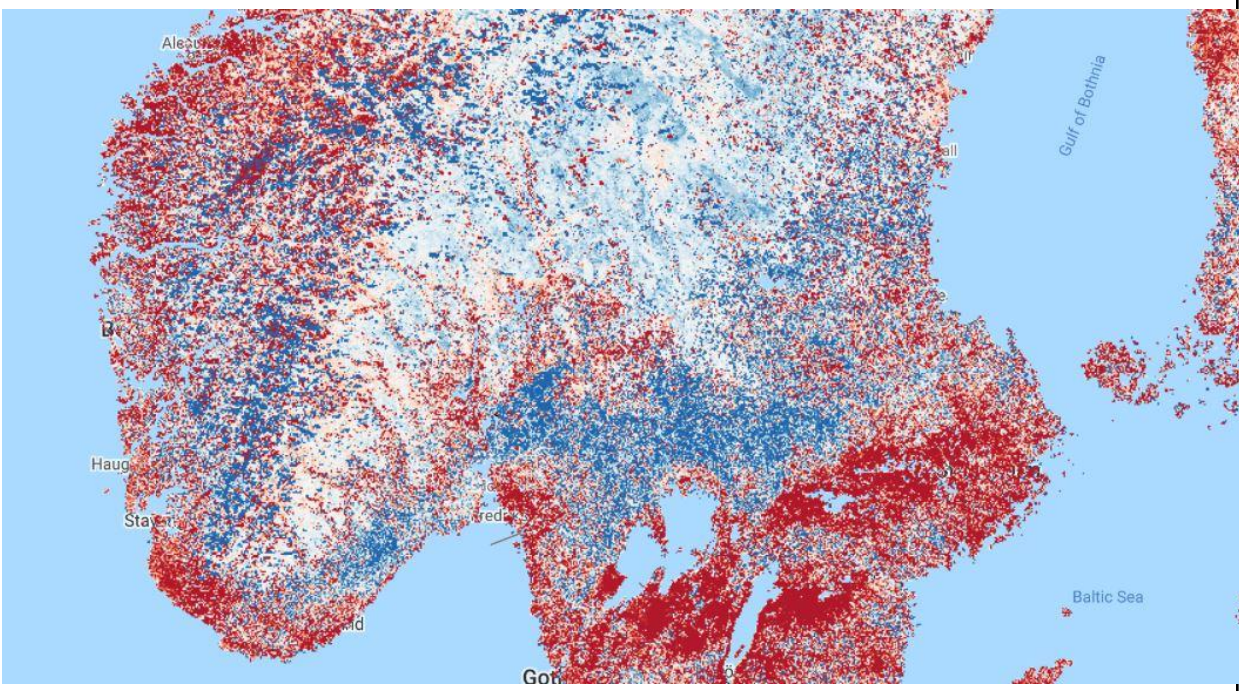


Fig. A2.6.4 Map representing the slope of the first DOY with no snow cover between 2000 and 2019. The slope represents the overall trend. Color grades from red (negative slope) to blue (positive slope). White indicates areas of little to no change.

Section 2.4. Time-Series Chart

To visually understand the temporal patterns of the first DOY with no snow through time, we can display our results in a time-series chart. In this case, we've defined a circle with a radius of 500 m around a point of interest and calculated the annual mean first DOY with no snow for pixels within that circle.

```
// Define an AOI.
var aoi = ee.Geometry.Point(-94.242, 65.79).buffer(1e4);
Map.addLayer(aoi, null, 'Area of interest');
```

```
// Calculate annual mean DOY of AOI.
var annualAoiMean = annualCol.select('calDoy').map(function(img) {
    var summary = img.reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: aoi,
        scale: 1e3,
        bestEffort: true,
        maxPixels: 1e14,
        tileScale: 4,
    });
    return ee.Feature(null, summary).set('year', img.get(
        'year'));
});

// Print chart to console.
var chart = ui.Chart.feature.byFeature(annualAoiMean, 'year',
        'calDoy')
    .setOptions({
        title: 'Regional mean first day of year with no snow cover',
        legend: {
            position: 'none'
        },
        hAxis: {
            title: 'Year',
            format: '####'
        },
        vAxis: {
            title: 'Day-of-year'
        }
    });
print(chart);
```

Code Checkpoint A26b. The book's repository contains a script that shows what your code should look like at this point.

As is evident in the displayed results (Fig. A2.6.5), the first DOY with no snow was mostly stable from 2000 to 2012; following this, it has become more erratic.

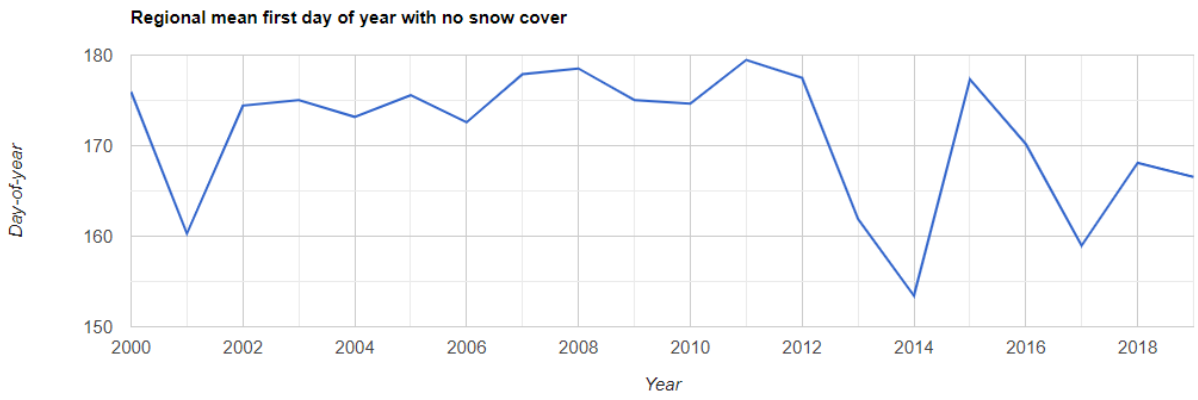


Fig. A2.6.5 Annual mean first DOY with no snow time series for pixels within a small region of interest

Synthesis

Assignment 1. In Sect. 2.4, a time-series chart for a region of interest was generated. Suppose you wanted to compare several regions in the same chart; how would you change the code to achieve this? For instance, try plotting the first DOY of no snow at points (−69.271, 65.532) and (−104.484, 65.445) in the same chart. Some helpful functions include `ee.Image.reduceRegions`, `ee.FeatureCollection.flatten`, and `ui.Chart.feature.groups`.

Assignment 2. The objective of this chapter was to identify the first DOY of 0% snow cover. However, Sect. 1.4 provides a suggestion for altering the code to identify the last day of 0% snow cover. Can you modify the code to achieve this result?

Assignment 3. How could you determine regions that are often masked during the time of year when snowmelt occurs? (The results from such regions might not be reliable.) Hint: Think about how you can use the `mask` function on images, and investigate the `'NDSI_Snow_Cover_Class'` and `'Snow_Albedo_Daily_Tile_Class'` bands.

Conclusion

In this chapter, we provided a method to identify the annual, per-pixel, first DOY with no snow from MODIS snow cover image data. The result can be used to investigate patterns of seasonal snowmelt spatially and temporally. The method relied on adding a date band to each image in the collection, temporal sorting, and `ImageCollection` reduction. We demonstrated several different ways to analyze the results using map and chart interpretation, image differencing, and linear regression.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Allan JD, Castillo MM (2007) Stream Ecology: Structure and Function of Running Waters. Springer Nature

Barnhart TB, Molotch NP, Livneh B, et al (2016) Snowmelt rate dictates streamflow. Geophys Res Lett 43:8006–8016. https://doi.org/10.1002/2016GL069690

Bengtsson L (1976) Snowmelt estimated from energy budget studies. Nord Hydrol 7:3–18. https://doi.org/10.2166/nh.1976.0001

Beniston M (2012) Impacts of climatic change on water and associated economic activities in the Swiss Alps. J Hydrol 412–413:291–296. https://doi.org/10.1016/j.jhydrol.2010.06.046

Clow DW (2010) Changes in the timing of snowmelt and streamflow in Colorado: A response to recent warming. J Clim 23:2293–2306. https://doi.org/10.1175/2009JCLI2951.1

Duchesne L, Houle D, D'Orangeville L (2012) Influence of climate on seasonal patterns of stem increment of balsam fir in a boreal forest of Québec, Canada. Agric For Meteorol 162–163:108–114. https://doi.org/10.1016/j.agrformet.2012.04.016

Harpold A, Brooks P, Rajagopal S, et al (2012) Changes in snowpack accumulation and ablation in the intermountain west. Water Resour Res 48. https://doi.org/10.1029/2012WR011949

Kudo G, Nordenhäll U, Molau U (1999) Effects of snowmelt timing on leaf traits, leaf production, and shoot growth of alpine plants: Comparisons along a snowmelt gradient in northern Sweden. Ecoscience 6:439–450. https://doi.org/10.1080/11956860.1999.11682543

Musselman KN, Clark MP, Liu C, et al (2017) Slower snowmelt in a warmer world. Nat Clim Chang 7:214–219. https://doi.org/10.1038/nclimate3225

Rouse WR, Douglas MSV, Hecky RE, et al (1997) Effects of climate change on the freshwaters of arctic and subarctic North America. Hydrol Process 11:873–902. https://doi.org/10.1002/(SICI)1099-1085(19970630)11:8<873::AID-HYP510>3.0.CO;2-6

Salomonson VV, Appel I (2004) Estimating fractional snow cover from MODIS using the normalized difference snow index. Remote Sens Environ 89:351–360. https://doi.org/10.1016/j.rse.2003.10.016

Solomon S, Manning M, Marquis M, et al (2007) Climate change 2007- The Physical Science Basis: Working Group I Contribution to the Fourth Assessment Report of the IPCC. Cambridge University Press

Tassone M, Epstein HE (2020) Drivers of spatial and temporal variability in vegetation productivity on the Yamal Peninsula, Siberia, Russia. In: AGU Fall Meeting Abstracts. pp B084-04

Wang L, Derksen C, Brown R (2008) Detection of pan-Arctic terrestrial snowmelt from QuikSCAT, 2000-2005. Remote Sens Environ 112:3794–3805. https://doi.org/10.1016/j.rse.2008.05.017

Outline

Below is an outline of the entire section, including every section header.