
Deforestation Viewed from Multiple Sensors

(A3.5)

Authors

Xiaojing Tang

Overview

Combining data from multiple sensors is the best way to increase data density and hence detect change faster. The purpose of this chapter is to demonstrate a simple method of combining Landsat, Sentinel-2, and Sentinel-1 data for monitoring tropical forest disturbance. You will learn how to import, preprocess, and fuse optical and synthetic aperture radar (SAR) remote sensing data. You will also learn how to monitor change using time-series models.

Learning Outcomes

- Combining optical and SAR images for change detection.
- Fitting a time-series model to an `ImageCollection`.
- Using established time-series models to detect anomalies in new images.
- Performing change detection with a rolling monitoring window.
- Monitoring forest disturbance in near real time.

Helps if you know how to:

- Understand the characteristics and preprocessing of Landsat and Sentinel images (Part F1).
- Understand regressions in Earth Engine (Chap. F3.0).
- Work with array images (Chap. F3.1, Chap. F4.6).
- Understand the concept of spectral unmixing (Chap. F3.2).
- 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).
- Export and import results as Earth Engine assets (Chap. F5.0).

Introduction to Theory

Deforestation and forest degradation are large sources of carbon emissions and negatively impact biodiversity, food security, and human well-being. The ability to quickly and accurately detect forest disturbance events is essential for preventing future forest loss and mitigating the negative effects. Combining optical and radar data has the potential to achieve faster detection of forest disturbance than using an individual system. The main challenge is the methodological approach for fusing the different datasets. The Fusion Near Real-Time (FNRT) algorithm (Tang et al. 2022) is a monitoring algorithm for tropical forest disturbance that combines data from Landsat, Sentinel-2, and Sentinel-1. For each sensor system, the FNRT algorithm fits time-series models over data from a three-year training period prior to the one-year monitoring period. FNRT then produces a change score for each new observation collected during the monitoring period based on the residuals and the root-mean-square error (RMSE) of the time-series model. The change scores from all three different sensors are then combined to form one dense time series for change detection. In this chapter, we will learn how to run a simplified version of FNRT to monitor forest disturbance in 2020 for a test area located in the Brazilian Amazon.

Practicum

This lab is designed for advanced users of Earth Engine. We assume that you already know how to import and preprocess large quantities of Landsat, Sentinel-2, and Sentinel-1 data as image collections. The code for importing and preprocessing input data will be provided in the example script, but it will not be discussed in detail in this chapter. To learn more about FNRT, refer to Tang et al. (2022).

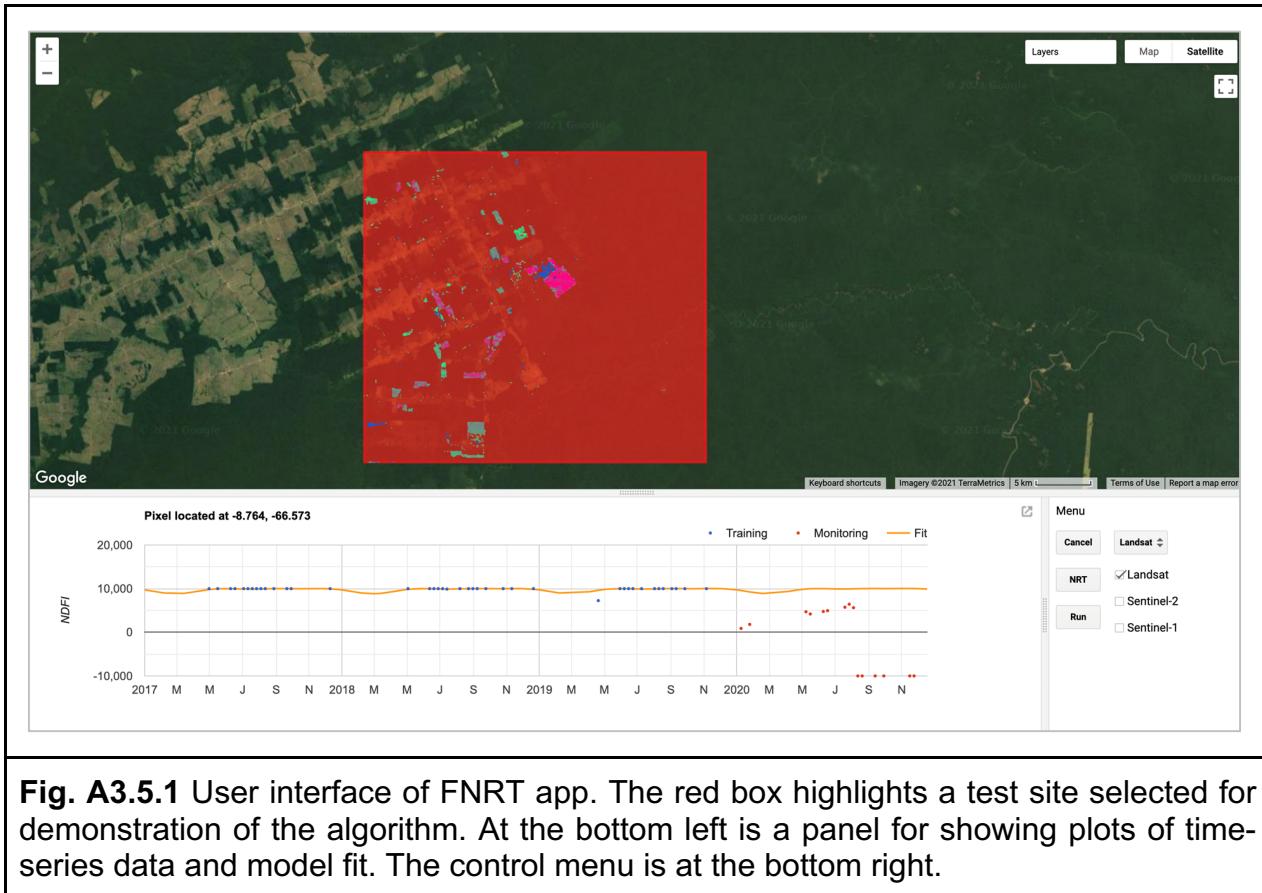
Section 1. Understand How FNRT Works

If you have not already done so, you can add the book's code repository to the Code Editor by entering

https://code.earthengine.google.com/?accept_repo=projects/gee-edu/book (or the short URL bit.ly/EEFA-repo) into your browser. The book's scripts will then be available in the script manager panel to view, run, or modify. If you have trouble finding the repo, you can visit bit.ly/EEFA-repo-help for help.

For this lab, we will use a graphical user interface to help us understand conceptually how FNRT combines data from different sensors and detects forest disturbance. Fig. A3.5.1 shows the user interface of the app.

Code Checkpoint A35a. The book's repository contains information about accessing the app.

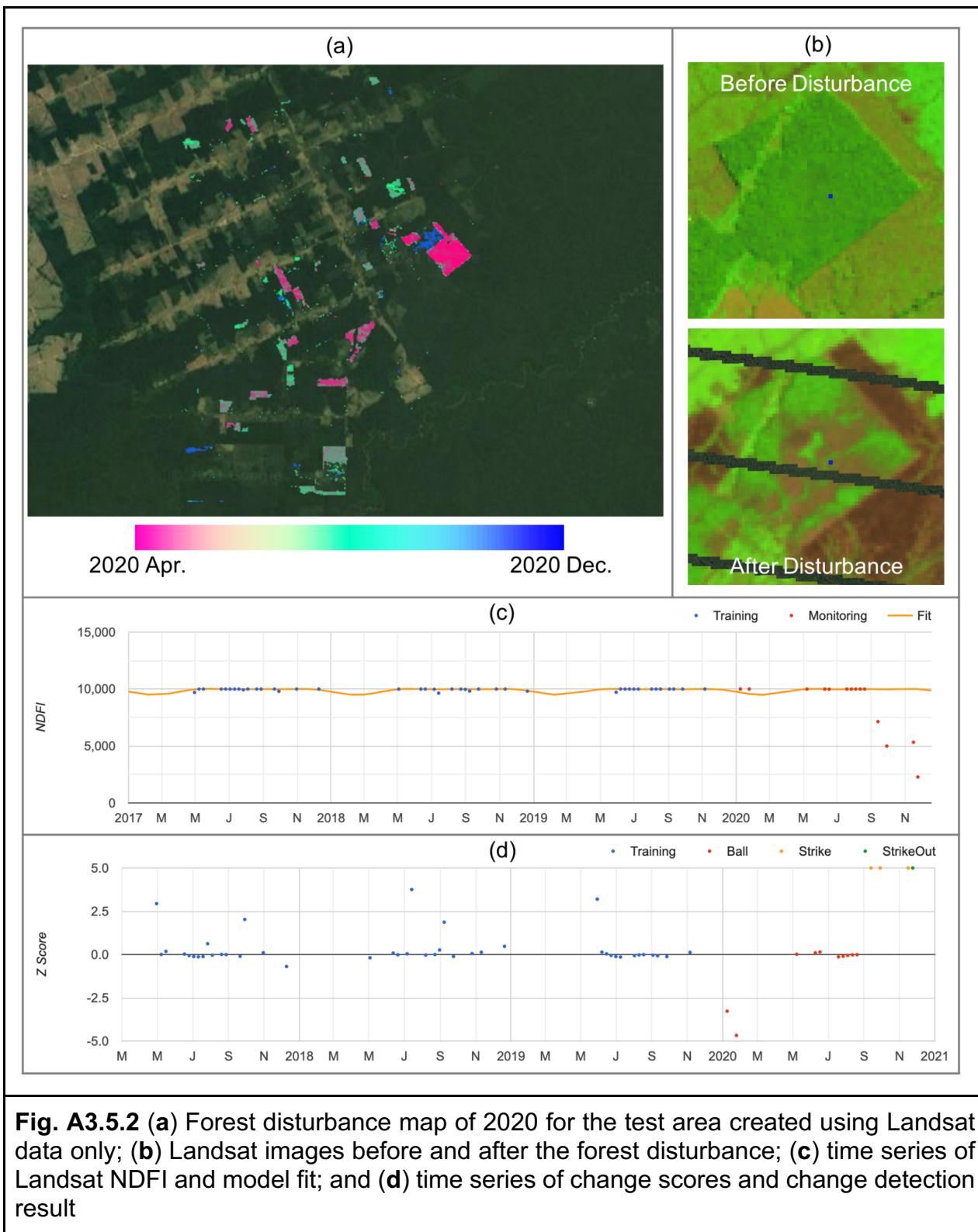


There are three main tools that can be used to explore the FNRT algorithm. Note that only one plotting tool can be activated at a time, and each tool can be deactivated by clicking the button again:

1. The **Fit** button activates the “fit time-series model” tool, which allows users to click anywhere in the map area, load the time-series data, and fit a harmonic model for the pixel covering the clicked location, using data from the sensor selected in the widget just to the right of the **Fit** button.
2. The **Monitor** button activates the “near real-time monitoring” tool, which allows users to click anywhere in the map area and plot the results of FNRT for the pixel covering the clicked location, using data from all the sensors that were checked via the checkbox widgets to the right of the **Monitor** button.
3. The **Run** button runs FNRT for the entire test area using data from the sensors that are checked, and loads the resulting forest disturbance map to the map area.

Let’s start by looking at some examples of time series so we have a better understanding of the model. We first need to know where to look for changes. Let’s try to have only “Landsat” checked, and click **Run**. This will run FNRT with only Landsat data and quickly produce a forest disturbance map for 2020 (Fig. A3.5.2a). Now click **Fit** to activate the “Fit time-series model” tool, and then click on a pixel that shows forest disturbance activity according to the map. Wait a few seconds to let the app generate the time-series

plot in the time series panel. Once it is completed, you should see a plot similar to Fig. A3.5.2c.



The model-fit plot (Fig. A3.5.2c) shows data from Landsat for the training period (blue) and monitoring period (red). It also shows a harmonic time-series model fit to the training data (orange line). The model fit extends into the monitoring period, which represents the model prediction. Notice that the satellite observations (red points) are very close to the model fit before the forest disturbance occurs. After the disturbance, the observations deviate away from the model fit. The algorithm monitors change by keeping track of differences between the observations and the model predictions.

Now click **Monitor** to activate the “near real-time monitoring” tool, and click the same pixel in the map. The app will then make a plot similar to (d) in Fig. A3.5.2. The near real-time monitoring plot (Fig. A3.5.2d) shows the change scores (Z-score) for the training and the monitoring period. A large change score indicates that something looks different in the satellite image. But sometimes that can be caused by a cloud or a cloud shadow that slipped through the masking process (e.g., the large change scores in the training period). Therefore, the monitoring algorithm looks for multiple large scores within a short monitoring window as an indicator of real forest disturbance.

The monitoring algorithm works through all the observations available during the monitoring period chronologically. Using baseball as an analogy, we flag each change score above a threshold as a “strike” and each change score below the threshold as a “ball.” If the algorithm finds multiple strikes in a short monitoring window, a “strikeout” would then be called on the pixel and the date of the first strike would be recorded as the date of change (Fig. A3.5.2d).

Question 1. How many strikes were detected before a forest disturbance was confirmed (strikeout)?

When performing multi-sensor data fusion, FNRT would fit a separate time-series model and calculate change scores using data from each sensor (Fig. A3.5.3a–c). The change scores calculated using data from the three sensors are then combined, and change monitoring is applied to the combined scores (Fig. A3.5.4d).

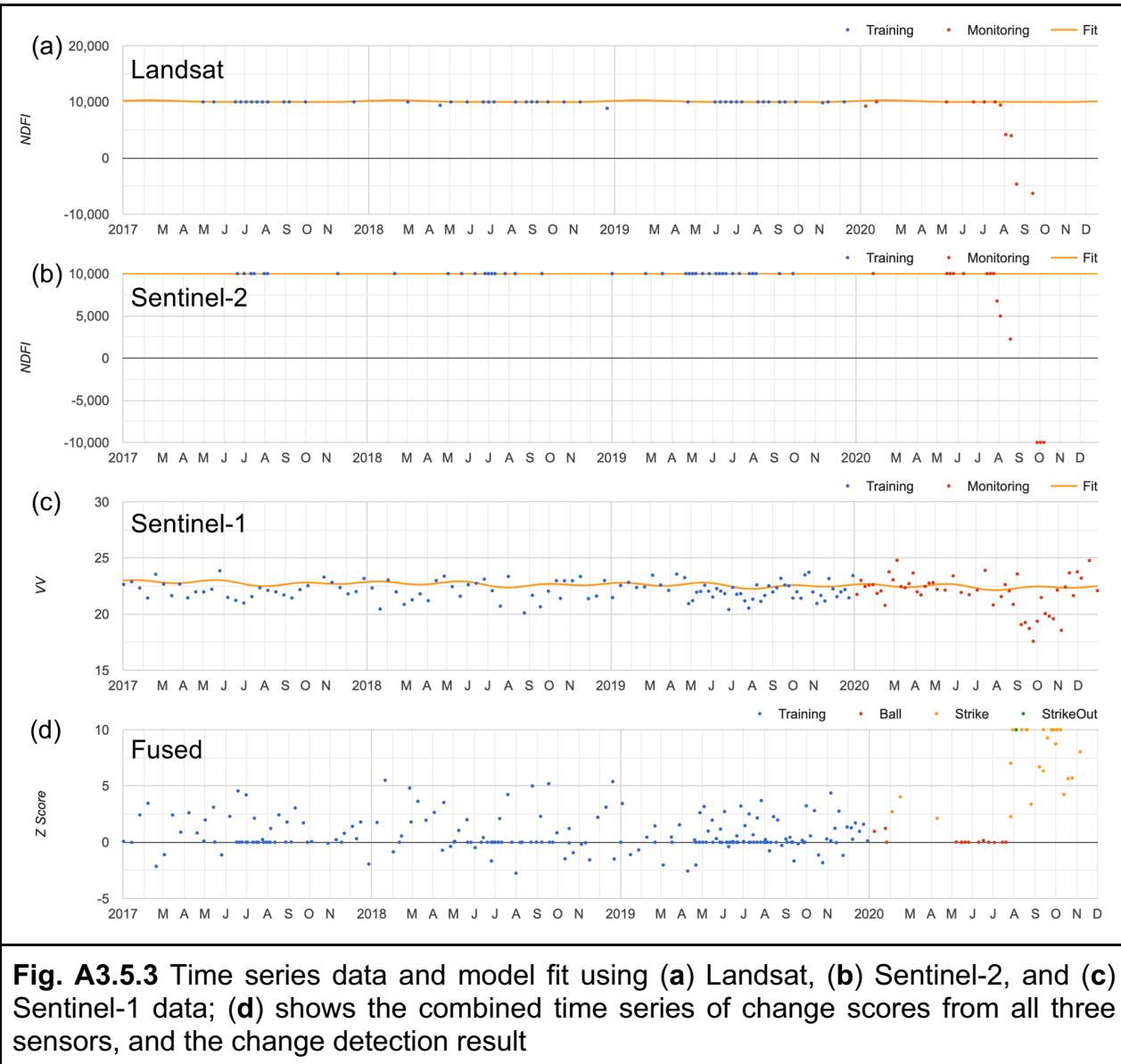


Fig. A3.5.3 Time series data and model fit using (a) Landsat, (b) Sentinel-2, and (c) Sentinel-1 data; (d) shows the combined time series of change scores from all three sensors, and the change detection result

The FNRT app also allows us to run change monitoring for the entire test area by clicking the **Run** button. A forest disturbance map showing the date of the disturbances is then added to the map. The users can use the checkbox widgets to choose which data stream is included in the process of making the map (e.g., Fig. A3.5.4). In general, optical data is more sensitive to forest disturbance than SAR data. Combining multiple data streams can help detect the disturbance earlier, compared to using data from each individual sensor.

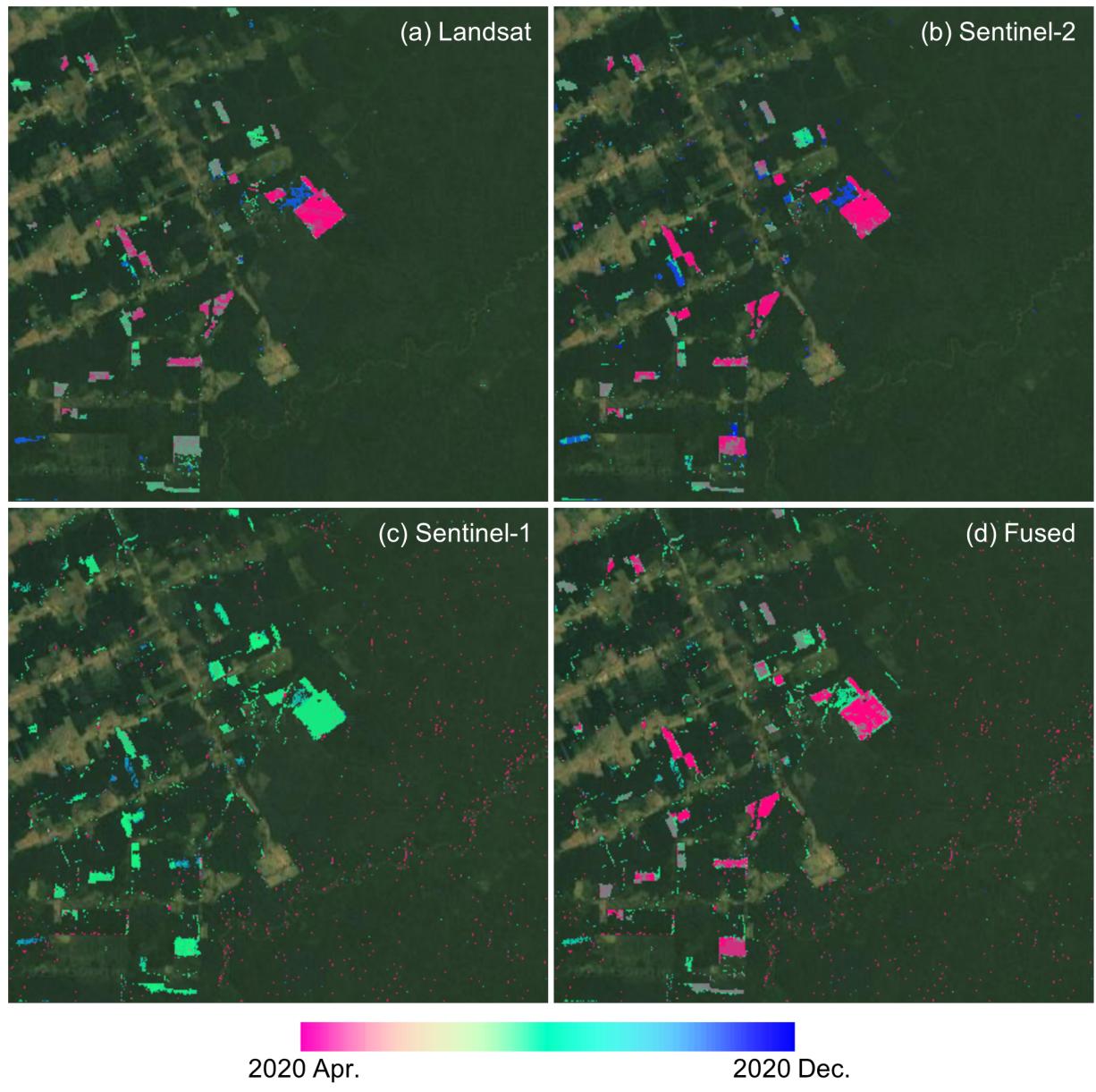


Fig. A3.5.4 Forest disturbance map made using data from (a) Landsat, (b) Sentinel-2, (c) Sentinel-1, and (d) all three sensors.

Question 2. Can you find forest disturbances that were detected earlier through the use of data from all three sensors rather than using just Landsat data?

Now that we understand conceptually how FNRT works, let's create a simple script to implement it.

Section 2. Define Study Area and Model Parameters

Section 2.1. Study Area

There are several ways to define a study area: 1) import an existing [FeatureCollection](#) from an Earth Engine asset; 2) use the drawing tool of the Code Editor; and 3) create a polygon by entering the coordinates. Here we select a box, approximately 20 km by 20 km, located in the Brazilian Amazon as our study area.

```
var testArea = ee.Geometry.Polygon(  
  [  
    [  
      [-66.73156878460787, -8.662236005089952],  
      [-66.73156878460787, -8.916025640576244],  
      [-66.44867083538912, -8.916025640576244],  
      [-66.44867083538912, -8.662236005089952]  
    ]  
  ]);  
  
Map.centerObject(testArea);
```

Note that because of the complexity of the FNRT algorithm, it requires quite a lot of computational resources and can result in a “User Memory Limit” error if applied to too large of an area. Divide the study area into multiple tiles if you are trying to monitor a large area.

Section 2.2. Model Parameters

A few model parameters need to be defined and can be tuned to optimize the algorithm for specific regions or monitoring conditions.

```
// Start and end of the training and monitoring period.  
var trainPeriod = ee.Dictionary({  
  'start': '2017-01-01',  
  'end': '2020-01-01'  
});  
var monitorPeriod = ee.Dictionary({  
  'start': '2020-01-01',  
  'end': '2021-01-01'  
});  
  
// Near-real-time monitoring parameters.  
var nrtParam = {
```

```

z: 2,
m: 5,
n: 4
};

// Sensor specific parameters.
var l1Param = {
  band: 'NDFI',
  minRMSE: 0.05,
  strikeOnly: false
};
var s2Param = {
  band: 'NDFI',
  minRMSE: 0.05,
  strikeOnly: false
};
var s1Param = {
  band: 'VV',
  minRMSE: 0.01,
  strikeOnly: true
};

```

For demonstration of the algorithm, we define the year 2020 as our monitoring period (`monitorPeriod`). We then recommend using the three years prior to the monitoring period as the training period (`trainPeriod`) to establish baseline time-series models.

A few parameters can be used to control the sensitivity of the algorithm: a z-threshold (`z`) defines the minimum change score required to flag a possible change observation; an m-size (`m`) defines the size of the monitoring window, or how many consecutive observations we check for change; and an n-change (`n`) defines the number of possible change observations within the monitoring window required to confirm a forest disturbance.

For each sensor we also need to select the band (`band`) to use for monitoring; a minimum RMSE (`minRMSE`) to prevent overly sensitive detection caused by unusually good model fit; and a strike-only flag (`strikeOnly`) to allow a data stream to contribute only to confirmation of a change.

Code Checkpoint A35b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Import and Preprocess Data

We need to import data from each sensor for the training period and the monitoring period as two separate image collections. Each image in the `ImageCollection` needs to be preprocessed. For optical data, we need to apply spectral unmixing and calculate the Normalized Difference Fraction Index (NDFI) (Souza et al. 2005). Since spectral unmixing would be applied to both Landsat and Sentinel-2 data, let's first implement a common "unmixing" function.

Several studies (e.g., Bullock et al. 2020, Chen et al. 2021) have shown that spectral unmixing and NDFI work very well in detecting tropical forest disturbance. Here we use the same endmembers used in Souza et al. (2005) for the spectral unmixing, and calculate NDFI based on Equations 4 and 5 in Souza et al. (2005).

```
var unmixing = function(col) {

    // Define endmembers and cloud fraction threshold.
    var gv = [500, 900, 400, 6100, 3000, 1000];
    var npv = [1400, 1700, 2200, 3000, 5500, 3000];
    var soil = [2000, 3000, 3400, 5800, 6000, 5800];
    var shade = [0, 0, 0, 0, 0, 0];
    var cloud = [9000, 9600, 8000, 7800, 7200, 6500];
    var cfThreshold = 0.05;

    return col.map(function(img) {
        // Select the spectral bands and perform unmixing
        var unmixed = img.select(['Blue', 'Green', 'Red',
            'NIR',
            'SWIR1', 'SWIR2'
        ])
        .unmix([gv, shade, npv, soil, cloud], true,
            true)
        .rename(['GV', 'Shade', 'NPV', 'Soil',
            'Cloud'
        ]);
    });

    // Calculate Normalized Difference Fraction Index.+
    var NDFI = unmixed.expression(
        '10000 * ((GV / (1 - SHADE)) - (NPV + SOIL)) / ' +
        '((GV / (1 - SHADE)) + (NPV + SOIL))', {
            'GV': unmixed.select('GV'),
            'SHADE': unmixed.select('Shade'),
    });
}
```

```

        'NPV': unmixed.select('NPV'),
        'SOIL': unmixed.select('Soil')
    }).rename('NDFI');

    // Mask cloudy pixel.
    var maskCloud = unmixed.select('Cloud').lt(
        cfThreshold);
    // Mask all shade pixel.
    var maskShade = unmixed.select('Shade').lt(1);
    // Mask pixel where NDFI cannot be calculated.
    var maskNDFI = unmixed.expression(
        '(GV / (1 - SHADE)) + (NPV + SOIL)', {
            'GV': unmixed.select('GV'),
            'SHADE': unmixed.select('Shade'),
            'NPV': unmixed.select('NPV'),
            'SOIL': unmixed.select('Soil')
        }).gt(0);

    // Scale fractions to 0-10000 and apply masks.
    return img
        .addBands(unmixed.select(['GV', 'Shade',
            'NPV', 'Soil'
        ])
            .multiply(10000))
        .addBands(NDFI)
        .updateMask(maskCloud)
        .updateMask(maskNDFI)
        .updateMask(maskShade);
    });
}

```

For Landsat data, we need to load the Landsat 7 and Landsat 8 Surface Reflectance product as an `ImageCollection`. We can filter the collection based on our study area and time period. Each image in the `ImageCollection` is masked based on the QA band; then spectral unmixing is applied and NDFI calculated. The `loadLandsatData` function is provided for importing and preprocessing Landsat data.

Preprocessing of Sentinel-2 data is similar to that of Landsat data because they are both optical remote sensing data and are almost identical in many ways. We need to load the Sentinel-2 top-of-atmosphere product (surface reflectance data do not have long enough time series for our purpose) and the Sentinel-2 cloud probability product as image

collections, and filter based on our study area and time period. We then join the two collections. Each image in the collection is then masked based on both the QA band and the cloud probability.

For Sentinel-1 data, we use the Sentinel-1 SAR GRD product. Preprocessing of Sentinel-1 data includes: 1) calculation of backscattering; 2) calculation of the ratio of VH and VV; 3) calculation of three-pixel spatial mean; 4) choice of the orbital pass with the most data for the region of interest; and 5) radiometric slope correction using volume model. Please refer to Mullissa et al. (2021) for more detail on the preprocessing of SAR data.

Here we assume you have already learned how to load and preprocess large quantities of Landsat, Sentinel-2, and Sentinel-1 data. We will not discuss these steps in detail here in this lab. Instead, three functions (`loadLandsatData`, `loadS2Data`, and `loadS1Data`) are provided in a separate script with which we can import using the `require` function. Now we can load all the required input data. We also need to apply the spectral unmixing and calculate NDFI for the Landsat and Sentinel-2 data.

```
var input = require(
  'projects/gee-edu/book:Part A - Applications/A3 - Terrestrial
  Applications/A3.5 Deforestation Viewed from Multiple
  Sensors/modules/Inputs'
);
var lstTraining = unmixing(input.loadLandsatData(testArea,
  trainPeriod));
var lstMonitoring = unmixing(input.loadLandsatData(testArea,
  monitorPeriod));
var s2Training = unmixing(input.loadS2Data(testArea, trainPeriod));
var s2Monitoring = unmixing(input.loadS2Data(testArea,
  monitorPeriod));
var s1Training = input.loadS1Data(testArea, trainPeriod);
var s1Monitoring = input.loadS1Data(testArea, monitorPeriod);
```

In addition to the remote sensing data, we also need to create a forest mask to limit our monitoring to forested areas. Here we will use the Hansen Global Forest Change dataset. We first use the “tree canopy cover” layer to find areas that had larger than 50% canopy cover in 2000. We then remove all the forest loss area during 2001–2019 and add in all the forest gain from 2000–2012 to create a mask of forested area by the beginning of our monitoring period.

```

var hansen = ee.Image('UMD/hansen/global_forest_change_2020_v1_8')
    .unmask();
var forestMask = hansen.select('treecover2000')
    .gt(50)
    .add(hansen.select('gain'))
    .subtract(hansen.select('loss'))
    .add(hansen.select('lossyear'))
        .eq(20)
    .gt(0)
    .clip(testArea);

```

Before we move on to the next section, let's do some simple checking. Print out the image collections and check whether the correct data is loaded. You can also load the forest mask and visually examine the quality of the mask.

```

var maskVis = {
  min: 0,
  max: 1,
  palette: ['blue', 'green']
};
Map.addLayer(forestMask, maskVis, 'Forest Mask');
print('1stTraining', 1stTraining);
print('1stMonitoring', 1stMonitoring);
print('s2Training', s2Training);
print('s2Monitoring', s2Monitoring);
print('s1Training', s1Training);
print('s1Monitoring', s1Monitoring);

```

Question 3. How many images are available from each sensor for our monitoring period? Are you surprised by the numbers? (Note that one sensor may have more data because the study area happened to be in the side-lap zone where two orbit passes overlap.) What bands are included in the data of each sensor?

Code Checkpoint A35c. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Establish Baseline Time Series Model

Before we can monitor change in near real time, we need to establish baseline time-series models using the data of the training period, so that we know what to expect from new observations in the monitoring period. Due to the different physical nature of optical

and radar data, it is easier to fit a separate time-series model to data from each sensor. Here we use a harmonic time-series model similar to the one used in the Continuous Change Detection and Classification (CCDC) algorithm (Zhu and Woodcock 2014). Because the timestamps of all the input images are stored in the unit of milliseconds while the harmonic model prefers a unit of years, we first need to define a function to convert any date object into the unit of fractional year (e.g. 2015-03-01 to 2015.1612).

```
var toFracYear = function(date) {
  var year = date.get('year');
  var fYear = date.difference(
    ee.Date.fromYMD(year, 1, 1), 'year');
  return year.add(fYear);
};
```

Here we define a function (`fitHarmonicModel`) to fit a harmonic model to an `ImageCollection` of time-series data. We first construct the dependent variables according to Equation 1 in Zhu and Woodcock (2014) and add them to each of the input images as new bands (using `addDependents` function). Then we use a reducer (`ee.Reducer.robustLinearRegression`) to fit the model to the data using robust linear regression. The function returns the model coefficients and the model RMSE.

```
var fitHarmonicModel = function(col, band) {
  // Function to add dependent variables to an image.
  var addDependents = function(img) {
    // Transform time variable to fractional year.
    var t = ee.Number(toFracYear(
      ee.Date(img.get('system:time_start')), 1));
    var omega = 2.0 * Math.PI;
    // Construct dependent variables image.
    var dependents = ee.Image.constant([
      1, t,
      t.multiply(omega).cos(),
      t.multiply(omega).sin(),
      t.multiply(omega * 2).cos(),
      t.multiply(omega * 2).sin(),
      t.multiply(omega * 3).cos(),
      t.multiply(omega * 3).sin()
    ])
    .float()
    .rename(['INTP', 'SLP', 'COS', 'SIN',
      'COS2', 'SIN2', 'COS3', 'SIN3'])
```

```

        ]);
    return img.addBands(dependents);
};

// Function to add dependent variable images to all images.
var prepareData = function(col, band) {
    return ee.ImageCollection(col.map(function(img) {
        return addDependents(img.select(band))
            .select(['INTP', 'SLP', 'COS',
                     'SIN',
                     'COS2', 'SIN2', 'COS3',
                     'SIN3',
                     band
                ])
            .updateMask(img.select(band)
                        .mask());
    }));
};

var col2 = prepareData(col, band);
// Fit model to data using robust linear regression.
var ccd = col2
    .reduce(ee.Reducer.robustLinearRegression(8, 1), 4)
    .rename([band + '_coefs', band + '_rmse']);

// Return model coefficients and model rmse.
return ccd.select(band + '_coefs').arrayTranspose()
    .addBands(ccd.select(band + '_rmse'));
};

```

With this function (`fitHarmonicModel`) we can fit the harmonic time-series model to the training data of all three sensors. It is always a good practice to also add some metadata to the results for future reference. The model fitting process will take a few minutes and quite a lot of computer memory. Therefore it is recommended that we run them as tasks and save the results as assets before we proceed to the next step.

```

// Fit harmonic models to training data of all sensors.
var lstModel = fitHarmonicModel(lstTraining, lstParam.band)
    .set({
        region: 'test',
        sensor: 'Landsat'
    }

```

```

    });
var s2Model = fitHarmonicModel(s2Training, s2Param.band)
.set({
  region: 'test',
  sensor: 'Sentinel-2'
});
var s1Model = fitHarmonicModel(s1Training, s2Param.band)
.set({
  region: 'test',
  sensor: 'Sentinel-1'
});

// Define function to save the results.
var saveModel = function(model, prefix) {
  Export.image.toAsset({
    image: model,
    scale: 30,
    assetId: prefix + '_CCD',
    description: 'Save_' + prefix + '_CCD',
    region: testArea,
    maxPixels: 1e13,
    pyramidingPolicy: {
      '.default': 'sample'
    }
  );
};

// Run the saving function.
saveModel(lstModel, 'LST');
saveModel(s2Model, 'S2');
saveModel(s1Model, 'S1');

```

Run the script. The model fitting results in export tasks to create assets. Instead of exporting the results for this exercise, however, you should proceed to the next section. In that section we will continue with a precomputed asset that is the same as what the above tasks would create.

Code Checkpoint A35d. The book's repository contains a script that shows what your code should look like at this point.

Section 5. Create Predicted Values for the Monitoring Period

We will now start exploring the pre-exported results mentioned in the previous section. Place this code below the checkpoint of the previous section:

```
var models = ee.ImageCollection('projects/gee-book/assets/A3-5/ccd');
var l1tModel = models
    .filterMetadata('sensor', 'equals', 'Landsat').first();
var s2Model = models
    .filterMetadata('sensor', 'equals', 'Sentinel-2').first();
var s1Model = models
    .filterMetadata('sensor', 'equals', 'Sentinel-1').first();
```

In this section we will use the time-series models that we produced in Section 4 to create a predicted image (also commonly referred to as a synthetic image; see Zhu et al. 2015) for any given date. The predicted image can give us a good estimate of the reflectance (or index such as NDFI) value of a place assuming no change has occurred. We can then compare the predicted image to the image collected by the satellite on the same date. Areas that look very different would likely be areas that have changed. Note that all the coefficients of the harmonic model were saved as a single-band array. So the very first step here is to define a function `dearrayModel` to convert the array image into a multiband image with each coefficient in one band.

```
var dearrayModel = function(model, band) {
  band = band + '_';

  // Function to extract a non-harmonic coefficients.
  var genCoefImg = function(model, band, coef) {
    var zeros = ee.Array(0).repeat(0, 1);
    var coefImg = model.select(band + coef)
        .arrayCat(zeros, 0).float()
        .arraySlice(0, 0, 1);
    return ee.Image(coefImg
        .arrayFlatten([
          [ee.String('S1_')
            .cat(band).cat(coef)]
        ]));
  };

  // Function to extract harmonic coefficients.
```

```

var genHarmImg = function(model, band) {
  var harms = ['INTP', 'SLP', 'COS', 'SIN',
    'COS2', 'SIN2', 'COS3', 'SIN3'
  ];
  var zeros = ee.Image(ee.Array([
    ee.List.repeat(0, harms.length)
  ]))
  .arrayRepeat(0, 1);
  var coefImg = model.select(band + 'coefs')
  .arrayCat(zeros, 0).float()
  .arraySlice(0, 0, 1);
  return ee.Image(coefImg
    .arrayFlatten([
      [ee.String(band).cat('coef')], harms
    ]));
};

// Extract harmonic coefficients and rmse.
var rmse = genCoefImg(model, band, 'rmse');
var coef = genHarmImg(model, band);
return ee.Image.cat(rmse, coef);
};

```

Second, we need to define a function that creates a predicted image for all the dates for which a real image is available during the monitoring period.

```

var createPredImg = function(modelImg, img, band, sensor) {
  // Reformat date.
  var date = toFracYear(ee.Date(img.get('system:time_start')));
  var dateString = ee.Date(img.get('system:time_start'))
    .format('yyyyMMdd');
  // List of coefficients .
  var coefs = ['INTP', 'SLP', 'COS', 'SIN',
    'COS2', 'SIN2', 'COS3', 'SIN3'
  ];
  // Get coefficients images from model image.
  var coef = ee.Image(coefs.map(function(coef) {
    return modelImg.select("." + concat(coef));
  })).rename(coefs);
  var t = ee.Number(date);
  var omega = 2.0 * Math.PI;

```

```

// Construct dependent variables.
var pred = ee.Image.constant([
    1, t,
    t.multiply(omega).cos(),
    t.multiply(omega).sin(),
    t.multiply(omega * 2).cos(),
    t.multiply(omega * 2).sin(),
    t.multiply(omega * 3).cos(),
    t.multiply(omega * 3).sin()
])
.float();

// Matrix multiply dependent variables with coefficients.
return pred.multiply(coef).reduce('sum')
    // Add original image and rename bands.
    .addBands(img, [band]).rename(['predicted', band])
    // Preserve some metadata.
    .set({
        'sensor': sensor,
        'system:time_start': img.get('system:time_start'),
        'dateString': dateString
    });
}

```

The `createPredImg` function calculates a predicted image based on the date of the real image and the model coefficients. It returns a new image with both the predicted image and the real image.

We can then apply the `createPredImg` function to each image in the image collections of the monitoring data. We can define another function to `map` over an `ImageCollection` to apply `createPredImg` to all images in the collection.

```

var addPredicted = function(data, modelImg, band, sensor) {
    return ee.ImageCollection(data.map(function(img) {
        return createPredImg(modelImg, img, band,
            sensor);
    }));
}

```

Let's apply `addPredicted` to monitoring data of all three sensors. Note this process is still sensor-specific. Let's print out the result for Landsat to examine the structure of the output.

```

// Convert models to non-array images.
var l1ModelImg = dearrayModel(l1Model, l1Param.band);
var s2ModelImg = dearrayModel(s2Model, s2Param.band);
var s1ModelImg = dearrayModel(s1Model, s1Param.band);

// Add predicted image to each real image.
var l1Predicted = addPredicted(l1Monitoring, l1ModelImg,
    l1Param.band, 'Landsat');
var s2Predicted = addPredicted(s2Monitoring, s2ModelImg,
    s2Param.band, 'Sentinel-2');
var s1Predicted = addPredicted(s1Monitoring, s1ModelImg,
    s1Param.band, 'Sentinel-1');

print('l1Predicted', l1Predicted);

```

Question 4. How many images are in the `ImageCollection` of the predicted images? Is it the same number as the original `ImageCollection` of the monitoring data?

Code Checkpoint A35e. The book's repository contains a script that shows what your code should look like at this point.

Section 6. Calculate Change Scores

Assuming a good model fit, the predicted image would be very close to the real image of the same date for areas with no change. So what we need to do now is examine each pair of real and predicted images and look for areas where they differ from each other. And if we merge the three data streams, we would be able to significantly increase the data density and potentially detect changes faster. But before we can fuse the three data streams together, we need to produce a normalized measurement of the likelihood of change.

The FNRT algorithm calculates a change score (Tang et al. 2019) very similar to a z-score. First, it calculates the residuals defined as the difference between the predicted value and the observed value. The residuals are then scaled with the RMSE of the time-series model to get the change score. If a change score is above the preset threshold (`z` in the parameters), then it is flagged as “possible change” (or a “strike” in baseball terminology). The implementation of this part is quite simple:

```

// Function to calculate residuals.
var addResiduals = function(data, band) {
  return ee.ImageCollection(data.map(function(img) {
    return img.select('predicted')
  })

```

```

        // Restrict predicted value to be under 10000
        .where(img.select('predicted').gt(10000),
               10000)
        // Calculate the residual
        .subtract(img.select(band))
        .rename('residual')
        // Save some metadata
        .set({
            'sensor': img.get('sensor'),
            'system:time_start': img.get(
                'system:time_start'),
            'dateString': img.get(
                'dateString')
        });
    }));
};

// Function to calculate change score and flag change.
var addChangeScores = function(data, rmse, minRMSE,
threshold, strikeOnly) {
    // If strikeOnly then we need a mask for balls.
    var mask = ee.Image(0);
    if (strikeOnly) {
        mask = ee.Image(1);
    }

    return ee.ImageCollection(data.map(function(img) {
        // Calculate change score
        var z = img.divide(rmse.max(minRMSE));
        // Check if score is above threshold
        var strike = z.multiply(z.gt(threshold));
        // Create the output image.
        var zStack = ee.Image.cat(z, strike).rename([
            'z', 'strike'
        ])
        .set({
            'sensor': img.get('sensor'),
            'system:time_start': img.get(
                'system:time_start')
        });
        // Mask balls if strikeOnly.
    }));
}

```

```

        return zStack.updateMask(strike.gt(0).or(
            mask));
    });
}

```

Note that here we restrict the predicted value to be under 10000 because that is the theoretical maximum for NDFI.

We also need to have a minimum RMSE (minRMSE) when scaling the residuals. This is because when using indices such as NDFI, it is sometimes possible to have perfect model fits resulting in very low or even zero RMSE, which would then cause the model to be overly sensitive to subtle variations in the monitoring data. The minRMSE is calculated for each pixel as a percentage of the average values of all observations in the training period.

For this particular implementation of FNRT, the determination of a strike is directional, meaning that we are looking for changes only in one direction. This is because we already expect that a forest disturbance will cause a decrease in NDFI or VV backscattering. Modification would be required if monitoring in another direction (e.g., if using SWIR reflectance) or bidirectional monitoring is desired. Let's apply the two new functions to our three data streams:

```

// Add residuals to collection of predicted images.
var l1Residuals = addResiduals(l1Predicted, l1Param.band);
var s2Residuals = addResiduals(s2Predicted, s2Param.band);
var s1Residuals = addResiduals(s1Predicted, s1Param.band);

// Add change score to residuals.
var l1Scores = addChangeScores(
    l1Residuals, l1ModelImg.select('.*rmse'),
    l1Predicted.select(l1Param.band).mean()
    .abs().multiply(l1Param.minRMSE),
    nrtParam.z, l1Param.strikeOnly);
var s2Scores = addChangeScores(
    s2Residuals, s2ModelImg.select('.*rmse'),
    s2Predicted.select(s2Param.band).mean()
    .abs().multiply(s2Param.minRMSE),
    nrtParam.z, s2Param.strikeOnly);
var s1Scores = addChangeScores(
    s1Residuals, s1ModelImg.select('.*rmse'),
    s1Predicted.select(s1Param.band).mean())

```

```
.abs().multiply(s1Param.minRMSE),  
nrtParam.z, s1Param.strikeOnly);  
  
print('1stScores', 1stScores);
```

Code Checkpoint A35f. The book's repository contains a script that shows what your code should look like at this point.

Question 5. What bands are included in the change scores images?

Section 7. Multisensor Data Fusion and Change Detection

Now we are finally ready to fuse the three data streams together. This step is quite straightforward: we just need to `merge` the three image collections of the change scores, and then `sort` them by their timestamps.

```
var fused = 1stScores.merge(s2Scores).merge(s1Scores)  
  .sort('system:time_start');
```

The change monitoring function uses a binary array to keep track of the number of strikes within a monitoring window. The size of the monitoring window is defined by the model parameters (`nrtParam.m`). The default monitoring window size is five.

We first initiate the monitoring window with a binary array of 0 with a size of five bits (00000) for each pixel. When we iterate through all the change score images chronologically, we shift the binary array left for those pixels that were not masked in the current change score image (00000 → 0000). We then append the current change flag (strike or not) to the shifted binary array (0000 → 00001). The process continues, as we iterate through all the change scores images (e.g., 00001 → 00011 → 00110 → 01101 → 11011). Every time a new value is appended to the binary array, we also check how many strikes (1s) are there. If the number of strikes exceeds the required number (`nrtParam.n`) to flag a change, then a change is flagged and the date of the change recorded.

```
var monitorChange = function(changeScores, nrtParam) {  
  // Initialize an empty image.  
  var zeros = ee.Image(0).addBands(ee.Image(0))  
    .rename(['change', 'date']);  
  // Determine shift size based on size of monitoring window.  
  var shift = Math.pow(2, nrtParam.m - 1) - 1;
```

```

// Function to monitor.
var monitor = function(img, result) {
    // Retrieve change image at last step.
    var change = ee.Image(result).select('change');
    // Retrieve change date image at last step.
    var date = ee.Image(result).select('date');
    // Create a shift image to shift the change binary array
    // left for one space so that new one can be appended.
    var shiftImg = img.select('z').mask().eq(0)
        .multiply(shift + 1).add(shift);
    change = change.bitwiseAnd(shiftImg)
        .multiply(shiftImg.eq(shift).add(1))
        .add(img.select('strike').unmask().gt(0));
    // Check if there are enough strike in the current
    // monitoring window to flag a change.
    date = date.add(change.bitCount().gte(nrtParam.n)
        // Ignore pixels where change already detected.
        .multiply(date.eq(0)))
        // Record change date where change is flagged.
        .multiply(ee.Number(toFracYear(
            ee.Date(img.get(
                'system:time_start')), 1))));
    // Combine change and date layer for next iteration.
    return (change.addBands(date));
};

// Iterate through the time series and look for change.
return ee.Image(changeScores.iterate(monitor, zeros))
    // Select change date layer and selfmask.
    .select('date').rename('Alerts').selfMask();
};

```

Now we apply the monitoring function (`monitorChange`) to the fused data (`fused`) using the default model parameters (`nrtParam`). We can then add the result, the forest disturbance map, to the map panel to visually examine it.

```

var alerts = monitorChange(fused, nrtParam).updateMask(forestMask);
print('alerts', alerts);

// Define a visualization parameter.
var altVisParam = {

```

```

min: 2020.4,
max: 2021,
palette: ['FF0080', 'EC1280', 'DA2480', 'C83680', 'B64880',
'A35B80', '916D80', '7F7F80', '6D9180', '5BA380',
'48B680', '36C880', '24DA80', '12EC80', '00FF80',
'00EB89', '00D793', '00C49D', '00B0A7', '009CB0',
'0089BA', '0075C4', '0062CE', '004ED7', '003AE1',
'0027EB', '0013F5', '0000FF'
]
};

Map.centerObject(testArea, 10);
Map.addLayer(alerts, altVisParam, 'Forest Disturbance Map (2020)');
Map.setOptions('SATELLITE');

```

Code Checkpoint A35g. The book's repository contains a script that shows what your code should look like at this point.

If we run the whole script now, we can create a map of forest disturbance that occurred in 2020 for our test area. The visualization of the map shows us the timing of all the forest disturbance detected by the FNRT algorithm (Fig. A3.5.5). You will notice a few large disturbance events. There may also be some tiny speckles, which are likely errors of commission. Try adjusting the monitoring parameters (`nrtParam`) to see if you can improve the map. Another common practice is to apply a spatial filter to filter out scattered single pixels of change (Bullock et al. 2020).



2020 Apr.

2020 Dec.

Fig. A3.5.5 Map of forest disturbance occurring in 2020 for a selected test area located on the frontline of deforestation in the Brazilian Amazon

Question 6. Can you run the change detection without Sentinel-1 input or with just Landsat input? Was forest disturbance detected faster when we used data from all three sensors?

Synthesis

Now that you have learned how to implement the simplified version of FNRT for the test area that we selected, let's try to use it to do some real monitoring of tropical forest disturbance.

Assignment 1. Modify the code and apply it in a different area of your choice. Note that you will need to keep the area small in order to be able to run everything easily in the Code Editor.

Assignment 2. Try to monitor more recent changes by changing the monitoring period to 2021. Note you would need to change the training period to 2018–2020, and also update the forest mask.

Assignment 3. If you want to try a more challenging task, you can integrate your results into the FNRT app that we used in Sect. 1.

Conclusion

In this chapter we experimented with a simple way to combine data from three different sensors (Landsat, Sentinel-2, and Sentinel-1) for the purpose of detecting tropical forest disturbance as early as possible. We started with loading and preprocessing the input data. We learned how to fit a harmonic model to time-series data, and how to make predicted observations based on a model fit. We also learned how to look for changes with a moving monitoring window over a time series of change scores. The full FNRT algorithm is more complex than the “lite” version that we implemented in this lab. If interested, please refer to Tang et al. (2022) for more details.

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

Bullock EL, Woodcock CE, Olofsson P (2020) Monitoring tropical forest degradation using spectral unmixing and Landsat time series analysis. *Remote Sens Environ* 238:110968. <https://doi.org/10.1016/j.rse.2018.11.011>

Chen S, Woodcock CE, Bullock EL, et al (2021) Monitoring temperate forest degradation on Google Earth Engine using Landsat time series analysis. *Remote Sens Environ* 265:112648. <https://doi.org/10.1016/j.rse.2021.112648>

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>

Souza Jr CM, Roberts DA, Cochrane MA (2005) Combining spectral and spatial information to map canopy damage from selective logging and forest fires. *Remote Sens Environ* 98:329–343. <https://doi.org/10.1016/j.rse.2005.07.013>

Tang X, Bratley KE, Cho K, et al (2022) Near real-time monitoring of tropical forest disturbance by fusion of Landsat, Sentinel-2, and Sentinel-1 data. *Remote Sens Environ*, in review

Tang X, Bullock EL, Olofsson P, et al (2019) Near real-time monitoring of tropical forest disturbance: New algorithms and assessment framework. *Remote Sens Environ* 224:202–218. <https://doi.org/10.1016/j.rse.2019.02.003>

Zhu Z, Woodcock CE (2014) Continuous change detection and classification of land cover using all available Landsat data. *Remote Sens Environ* 144:152–171. <https://doi.org/10.1016/j.rse.2014.01.011>

Zhu Z, Woodcock CE, Holden C, Yang Z (2015) Generating synthetic Landsat images based on all available Landsat data: Predicting Landsat surface reflectance at any given time. *Remote Sens Environ* 162:67–83. <https://doi.org/10.1016/j.rse.2015.02.009>