Codes Used for Data Collect on & Preprocessing

////////////////////////////////// Downloading Sentinel 2 Bands //////////////////////////////////

```
// /  Function to mask clouds using the Sentinel-2 QA band ///

function maskS2clouds(image) {
 var qa = image.select('QA60');

  // Bits 10 and 11 are clouds and cirrus, respectively.
  var cloudBitMask = 1 << 10;
  var cirrusBitMask = 1 << 11;

  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));

  return image.updateMask(mask).divide(10000);
}

var S2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED')
                .filterDate('2023-09-01', '2023-10-30')
                // Pre-filter to get less cloudy granules.
                .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 0.5))
                .map(maskS2clouds);

// Function to calculate and add an NDVI band
var addNDVI = function(image) {
return image.addBands(image.normalizedDifference(['B8', 'B4']));
};

// Add NDVI band to image collection
var S2 = S2.map(addNDVI);
// Extract NDVI band and create NDVI median composite image
var NDVI = S2.select(['nd']);
var NDVI = NDVI.median();

// Function to calculate and add an NDWI band
var addNDWI = function(image) {
```

```
return image.addBands(image.normalizedDifference(['B3', 'B8']));
};

// Add NDWI band to image collection
var S2 = S2.map(addNDWI);

// Extract NDWI band and create NDWI median composite image
var NDWI = S2.select(['nd_1']);
var NDWI = NDWI.median();

// Extract bands B2 (blue), B3 (green), B4 (red), and B8 (NIR); create
// median composite images of each
var B2 = S2.select(['B2']);
var B2 = B2.median();
var B3 = S2.select(['B3']);
var B3 = B3.median();
var B4 = S2.select(['B4']);
var B4 = B4.median();
var B8 = S2.select(['B8']);
var B8 = B8.median();

// Clip input variable images
var NDVI = NDVI.clip(geometry);
var NDWI = NDWI.clip(geometry);
var B2 = B2.clip(geometry);
var B3 = B3.clip(geometry);
var B4 = B4.clip(geometry);
var B8 = B8.clip(geometry);

// Create palettes for display of NDVI and NDWI
var ndvi_pal = ['#d73027', '#f46d43', '#fdae61', '#fee08b', '#d9ef8b',
'#a6d96a', '#66bd63', '#1a9850'];
var ndwi_pal = ['#ece7f2', '#d0d1e6', '#a6bddb', '#74a9cf', '#3690c0',
'#0570b0', '#045a8d', '#023858'];

// Display NDVI and NDWI results on map
Map.addLayer(NDVI, {min:-0.5, max:0.9, palette: ndvi_pal}, 'NDVI');
Map.addLayer(NDWI, {min:-1, max:1, palette: ndwi_pal}, 'NDWI');
```

```
// Export results to Google Drive
Export.image.toDrive({
image: NDWI,
description: 'NDWI',
scale: 10,
region: geometry,
maxPixels: 10E10
});

Export.image.toDrive({
image: NDVI,
description: 'NDVI',
scale: 10,
region: geometry,
maxPixels: 10E10
});

Export.image.toDrive({
image: B2,
description: 'B2',
scale: 10,
region: geometry,
maxPixels: 10E10
});

Export.image.toDrive({
image: B3,
description: 'B3',
scale: 10,
region: geometry,
maxPixels: 10E10
});

Export.image.toDrive({
image: B4,
description: 'B4',
scale: 10,
```

```
region: geometry,
maxPixels: 10E10
});

Export.image.toDrive({
image: B8,
description: 'B8',
scale: 10,
region: geometry,
maxPixels: 10E10
});
```

//////////////////////////////////// Downloading Sentinel 1 Bands ////////////////////////////////////

```
var imgS1 = ee.ImageCollection('COPERNICUS/S1_GRD')
      .filterDate('2023-09-01', '2023-10-30')
      .filter(ee.Filter.eq('instrumentMode', 'IW'))
      .filterBounds(geometry)
      .map(function(image) {
        var edge = image.lt(-30.0);
        var maskedImage = image.mask().and(edge.not());
        return image.updateMask(maskedImage);
      });

print('Number of Images in Collection', imgS1.size());

var bandNames = imgS1.first().bandNames();
 print('Band Names for Image:', bandNames);

// Function to mask out edges of images using angle
// (mask out angles <= 30.63993)
var maskAngGT30 = function(image) {
var ang = image.select(['angle']);
return image.updateMask(ang.gt(30.63993));
};

// Function to mask out edges of images using using angle
// (mask out angles >= 45.23993)
var maskAngLT452 = function(image) {
var ang = image.select(['angle']);
return image.updateMask(ang.lt(45.23993));
};

// Apply angle masking functions to image collection
var imgS1 = imgS1.map(maskAngGT30);
var imgS1 = imgS1.map(maskAngLT452);

// Function to apply angle correction (for VV & VH Bands Individually - One at a Time)
function toGamma01(image) {
```

```
return image.select('VV').subtract(image.select('angle').clip(geometry)
// return image.select('VH').subtract(image.select('angle').clip(geometry)
.multiply(Math.PI/180.0).cos().log10().multiply(10.0));
}

// Apply angle correction
var imgS1 = imgS1.map(toGamma01);

//------------------------------------------------------------------------------// Sigma Lee speckle filtering //------------------------------------------------------------------------------//

function toNatural(img) {
return ee.Image(10.0).pow(img.select(0).divide(10.0));
}
function toDB(img) {
return ee.Image(img).log10().multiply(10.0);
}
// The RL speckle filter from
// https://code.earthengine.google.com/2ef38463ebaf5ae133a478f173fd0ab5
// by Guido Lemoine
function RefinedLee(img) {
// img must be in natural units, i.e. not in dB!
// Set up 3x3 kernels
var weights3 = ee.List.repeat(ee.List.repeat(1,3),3);
var kernel3 = ee.Kernel.fixed(3,3, weights3, 1, 1, false);
var mean3 = img.reduceNeighborhood(ee.Reducer.mean(), kernel3);
var variance3 = img.reduceNeighborhood(ee.Reducer.variance(), kernel3);
// Use a sample of the 3x3 windows inside a 7x7 windows to determine gradients
// and directions
var sample_weights = ee.List([[0,0,0,0,0,0,0], [0,1,0,1,0,1,0],
[0,0,0,0,0,0,0], [0,1,0,1,0,1,0], [0,0,0,0,0,0,0], [0,1,0,1,0,1,0],
[0,0,0,0,0,0,0]]);
var sample_kernel = ee.Kernel.fixed(7,7, sample_weights, 3,3, false);
// Calculate mean and variance for the sampled windows and store as 9 bands
var sample_mean = mean3.neighborhoodToBands(sample_kernel);
var sample_var = variance3.neighborhoodToBands(sample_kernel);
// Determine the 4 gradients for the sampled windows
var gradients = sample_mean.select(1).subtract(sample_mean.select(7)).abs();
gradients = gradients.addBands(sample_mean.select(6).subtract(sample_mean
```

```
.select(2)).abs());
gradients = gradients.addBands(sample_mean.select(3).subtract(sample_mean
.select(5)).abs());
gradients = gradients.addBands(sample_mean.select(0).subtract(sample_mean
.select(8)).abs());
// And find the maximum gradient amongst gradient bands
var max_gradient = gradients.reduce(ee.Reducer.max());
// Create a mask for band pixels that are the maximum gradient
var gradmask = gradients.eq(max_gradient);

// duplicate gradmask bands: each gradient represents 2 directions
gradmask = gradmask.addBands(gradmask);
// Determine the 8 directions
var directions = sample_mean.select(1).subtract(sample_mean.select(4))
.gt(sample_mean.select(4).subtract(sample_mean.select(7))).multiply(1);
directions = directions.addBands(sample_mean.select(6).subtract(sample_mean
.select(4)).gt(sample_mean.select(4).subtract(sample_mean.select(2)))
.multiply(2));
directions = directions.addBands(sample_mean.select(3).subtract(sample_mean
.select(4)).gt(sample_mean.select(4).subtract(sample_mean.select(5)))
.multiply(3));
directions = directions.addBands(sample_mean.select(0).subtract(sample_mean
.select(4)).gt(sample_mean.select(4).subtract(sample_mean.select(8)))
.multiply(4));
// The next 4 are the not() of the previous 4
directions = directions.addBands(directions.select(0).not().multiply(5));
directions = directions.addBands(directions.select(1).not().multiply(6));
directions = directions.addBands(directions.select(2).not().multiply(7));
directions = directions.addBands(directions.select(3).not().multiply(8));
// Mask all values that are not 1-8
directions = directions.updateMask(gradmask);
// "collapse" the stack into a singe band image (due to masking, each pixel
// has just one value (1-8) in it's directional band, and is otherwise masked)
directions = directions.reduce(ee.Reducer.sum());
// Generate stats
var sample_stats = sample_var.divide(sample_mean.multiply(sample_mean));
// Calculate localNoiseVariance
var sigmaV = sample_stats.toArray().arraySort().arraySlice(0,0,5)
```

```
.arrayReduce(ee.Reducer.mean(), [0]);
// Set up the 7*7 kernels for directional statistics
var rect_weights = ee.List.repeat(ee.List.repeat(0,7),3)
.cat(ee.List.repeat(ee.List.repeat(1,7),4));
// Set weights
var diag_weights = ee.List([[1,0,0,0,0,0,0], [1,1,0,0,0,0,0], [1,1,1,0,0,0,0],
[1,1,1,1,0,0,0], [1,1,1,1,1,0,0], [1,1,1,1,1,1,0], [1,1,1,1,1,1,1]]);
var rect_kernel = ee.Kernel.fixed(7,7, rect_weights, 3, 3, false);
var diag_kernel = ee.Kernel.fixed(7,7, diag_weights, 3, 3, false);
// Create stacks for mean and variance using the original kernels.
// Mask with relevant direction.
var dir_mean = img.reduceNeighborhood(ee.Reducer.mean(), rect_kernel)
.updateMask(directions.eq(1));
var dir_var = img.reduceNeighborhood(ee.Reducer.variance(), rect_kernel)
.updateMask(directions.eq(1));
dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
diag_kernel).updateMask(directions.eq(2)));
dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
diag_kernel).updateMask(directions.eq(2)));

// and add the bands for rotated kernels
for (var i=1; i<4; i++) {
dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
rect_kernel.rotate(i)).updateMask(directions.eq(2*i+1)));
dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
rect_kernel.rotate(i)).updateMask(directions.eq(2*i+1)));
dir_mean = dir_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(),
diag_kernel.rotate(i)).updateMask(directions.eq(2*i+2)));
dir_var = dir_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(),
diag_kernel.rotate(i)).updateMask(directions.eq(2*i+2)));
}
// "collapse" the stack into a single band image (due to masking, each pixel
// has just one value in it's directional band, and is otherwise masked)
dir_mean = dir_mean.reduce(ee.Reducer.sum());
dir_var = dir_var.reduce(ee.Reducer.sum());
// A finally generate the filtered value
var varX = dir_var.subtract(dir_mean.multiply(dir_mean).multiply(sigmaV))
.divide(sigmaV.add(1.0));
```

```
var b = varX.divide(dir_var);
var result = dir_mean.add(b.multiply(img.subtract(dir_mean)));
return(result.arrayFlatten([['sum']]));
}
//----------------------------------------------------------------------------// Sigma Lee speckle filtering End //----------------------------------------------------------------------------//

// Apply three functions as part of Sigma Lee filtering
var imgS1 = imgS1.map(toNatural);
var imgS1 = imgS1.map(RefinedLee);
var imgS1 = imgS1.map(toDB);

// var bandNames = imgS1.first().bandNames();
//   print('Band Names for Image:', bandNames);

// Extract mean VV and mean VH input variables
var Output = imgS1.mean();

// // Map orig mean VV and VH
Map.addLayer(Output,{min:-16.0, max:-0.53},'Filtered VV');
// Map.addLayer(Output,{min:-16.0, max:-1},'Filtered VH');


Export.image.toDrive({
image: Output,
description: 'VV_Speckle_Filtered',
scale: 10,
region: geometry,
maxPixels: 10E10
});

// Export.image.toDrive({
// image: Output,
// description: 'VH_Speckle_Filtered',
// scale: 10,
// region: geometry,
// maxPixels: 10E10
// });
```