Bài tập 2: Tiền xử lý ảnh và tạo ảnh Sentinel tổ hợp

Giới Thiệu

Bài thực hành này nhằm mục tiêu là tạo ảnh tổ hợp sau khi đã tiền xử lý cho ảnh Sentinel 2. Các bước thực hành sẽ tập trung vào các bước nhập dữ liệu, hiệu chỉnh phổ nhằm loại bỏ ảnh hưởng của bóng mây, mây, bóng địa hình, hướng góc phổ phản xạ. Bài thực hành này sử dụng phương pháp tích hợp các hợp phần xử lý (mỗi hợp phần là một mô đun). Đây là cách giải quyết các bài toán phức tạp có hướng nâng cao kỹ năng lập trình cho các học viên.

Kết quả bài thực hành

Kết quả bài thực hành là tạo ra một ảnh Sentinel 2 tổ hợp năm 2017 tại khu vực Tây Nguyên

Mục Lục

Bước 1: Nhập dữ liệu Sentinel 2 3

Bước 2: Loại bỏ pixel bị ảnh hưởng bóng mây 5

Bước 3. Tạo mặt nạ mây trên tập ảnh Sentinel 2 6

Bước 4. Hiệu chỉnh giá trị phổ phản xạ trên hàm phân bố 2 chiều tán xạ trên điểm ảnh 9

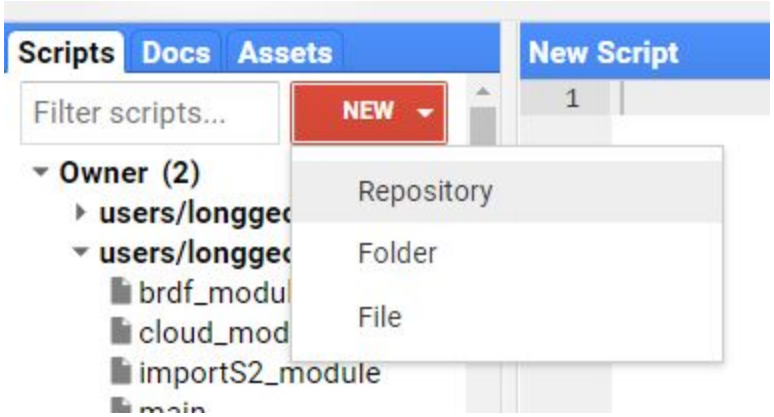
Bước 5. Hiệu chỉnh ảnh hưởng địa hình 19

"Ę \ úç .adi]. eb̆ n 3^gmbg^e û

Trong bài tập này, chúng ta sẽ làm quen với kiểu lập trình chạy một chương trình theo các mô đun. Sử dụng hàm exports để xuất kết quả trong script khác

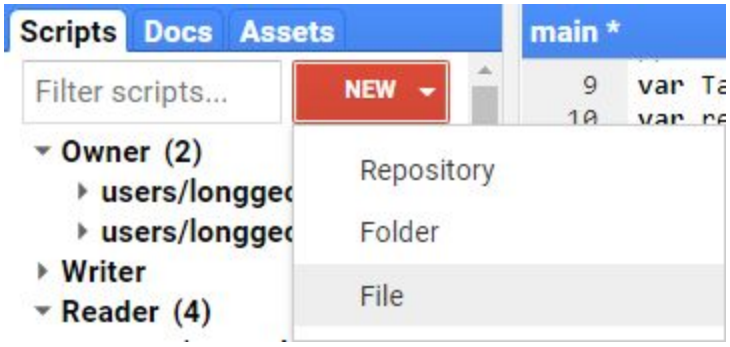
1.1. Tạo mới một repository

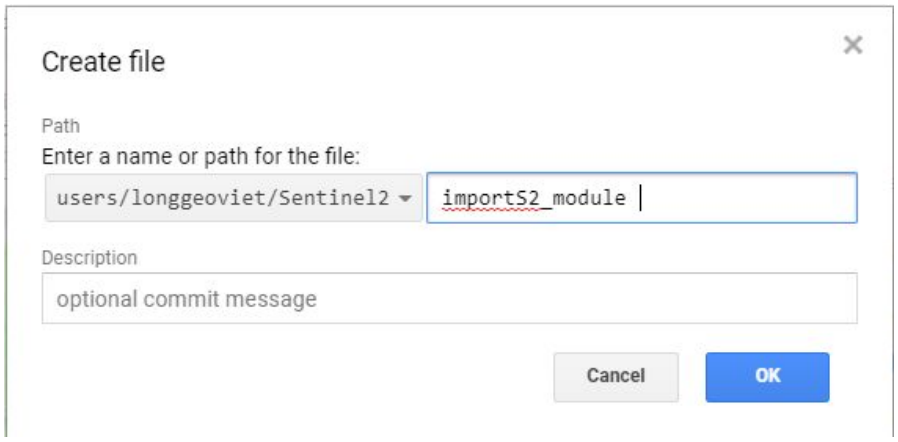
Một repository như một thư mục chứa các tệp thực thi của chương trình, để quản lý các mô đun chương trình được hiểu quả, cần đặt tất cả các tệp trong cùng một repository. Vào Scripts>New>Repository. Đặt tên repository mới là Sentinel2



1.2. Tạo mô đun nhập dữ liệu

1. Tạo mới một script đặt tên là **importS2_module**





2. Dán đoạn code dưới đây vào **importS2_module** rồi lưu lại

```
var inBands =
ee.List(['QA60','B1','B2','B3','B4','B5','B6','B7','B8','B8A','B9','B10','B11',
'B12']);
var outBands =
ee.List(['QA60','cb','blue','green','red','re1','re2','re3','nir','re4','waterV
apor','cirrus','swir1','swir2']);

var CloudCoverMax = 30;

exports.importData =
function(studyArea,startDate,endDate,startJulian,endJulian) {

// Khai b§o t¥p d¨ li¨u
var s2s = ee.ImageCollection('COPERNICUS/S2')
.filterDate(startDate,endDate)
//.l¨c ¨nh
.filterBounds(studyArea)
.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE',CloudCoverMax))
.filter(ee.Filter.lt('CLOUD_COVERAGE_ASSESSMENT',CloudCoverMax));

function scaleBands(img){
var prop = img.toDictionary()
var t =
img.select(['B1','B2','B3','B4','B5','B6','B7','B8','B8A','B9','B10','B11','B12
']).divide(10000);
```

```
t =
t.addBands(img.select(['QA60'])).set(prop).copyProperties(img,['system:time_sta
rt','system:footprint'])
return ee.Image(t);
}

s2s = s2s.map(scaleBands);
s2s = s2s.select(inBands,outBands) ;

return s2s;
};
```

1.3. Tạo file thực thi chính của chương trình

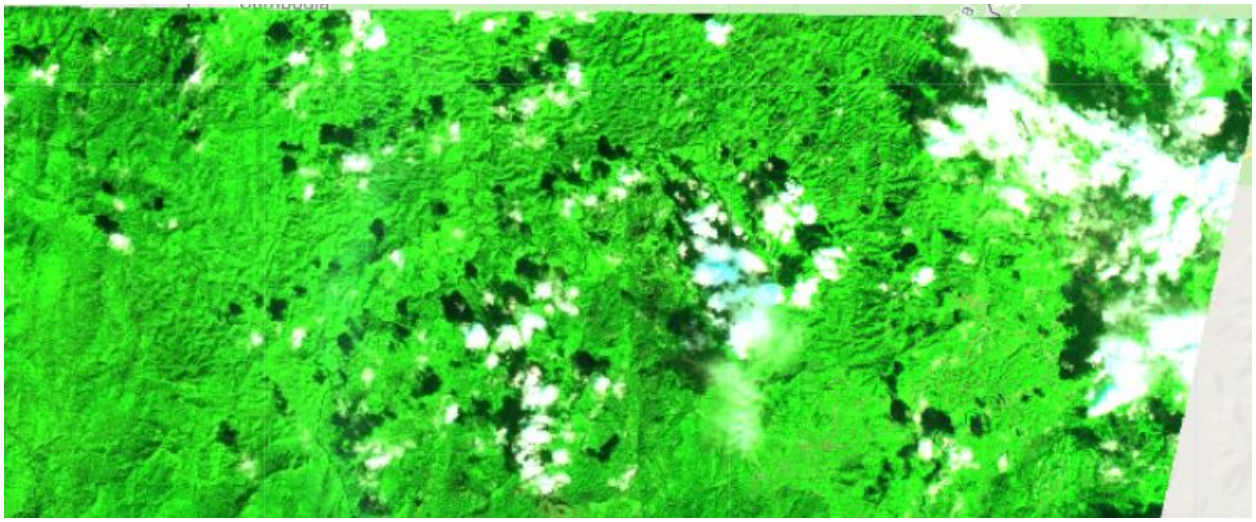
1. Tạo mới một cript đặt tên là main đây sẽ là file thân chương trình, khớp nối các mô đun lại để thực thi
2. Dán đoạn code dưới đây vào file **main**. Chú ý cần phải sửa **username** để tương ứng tới tài khoản người dùng

```
// Nh×p m¹ dun importS2
var importS2 = require("users/username/Sentinel2:importS2_module");

// Nh×p c§c bi°n kh¹ng gian v¨ th`i gian
var TayNguyen =
ee.FeatureCollection("users/longgeoviet/CHL_Boundary").filter(ee.Filter.eq("nam
e","TN"));
var region = TayNguyen.geometry();

var startDate = "2017-01-01";
var endDate = "2018-01-01";

print("getting images");
var s2 = importS2.importData(region,startDate,endDate);
print("found ",s2.size(),"images");
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"first
image");
print(ee.Image(s2.first()));
```



"ẹ \ ûç ,hÜb [¸ ibq^e [˙ [ga aẾ g` [¨g` f vr

2.1. Tạo mô đun khử các pixel bị ảnh hưởng bóng mây

1. Tạo mới script trong repository Sentinel 2 đặt tên là **shadow_module**

2. Dán đạn code dưới đây vào script **shadow_module** và lưu lại:

```
// zScoreThresh: Threshold for cloud shadow masking- lower number masks out
// less. Between -0.8 and -1.2 generally works well
var zScoreThresh = -1;

// shadowSumThresh: Sum of IR bands to include as shadows within TDOM and the
// shadow shift method (lower number masks out less)
var shadowSumThresh = 0.35;

// contractPixels: The radius of the number of pixels to contract (negative
// buffer) clouds and cloud shadows by. Intended to eliminate smaller cloud
// patches that are likely errors
// (1.5 results in a -1 pixel buffer)(0.5 results in a -0 pixel buffer)
// (1.5 or 2.5 generally is sufficient)
var contractPixels = 1.5;

// dilatePixels: The radius of the number of pixels to dilate (buffer) clouds
// and cloud shadows by. Intended to include edges of clouds/cloud shadows
// that are often missed
// (1.5 results in a 1 pixel buffer)(0.5 results in a 0 pixel buffer)
// (2.5 or 3.5 generally is sufficient)
var dilatePixels = 2.5;

////////////////////////////////////////////////////////////////////////////
/
// Function for finding dark outliers in time series.
// Original concept written by Carson Stam and adapted by Ian Housman.
// Adds a band that is a mask of pixels that are dark, and dark outliers.
exports.shadowMask = function(collection,studyArea) {

var inBands = ["B8",'B11'];
var shadowSumBands = ['nir','swir1'];

var allCollection =
ee.ImageCollection('COPERNICUS/S2').filterBounds(studyArea).select(inBands,shad
owSumBands);
// Get some pixel-wise stats for the time series
var irStdDev =
allCollection.select(shadowSumBands).reduce(ee.Reducer.stdDev());
var irMean = allCollection.select(shadowSumBands).mean();

var maskDarkOutliers = function(img){
var zScore = img.select(shadowSumBands).subtract(irMean).divide(irStdDev);
var irSum = img.select(shadowSumBands).reduce(ee.Reducer.sum());
var TDOMMask =
zScore.lt(zScoreThresh).reduce(ee.Reducer.sum()).eq(2).and(irSum.lt(shadowSumTh
resh));
```

```
TDOMMask =
TDOMMask.focal_min(contractPixels).focal_max(dilatePixels).rename('TDOMMask');
return img.updateMask(TDOMMask.not()).addBands(TDOMMask);
};

// Mask out dark dark outliers
collection = collection.map(maskDarkOutliers);

return collection;
};
```

2.2. Gọi mô đun hiệu chỉnh bóng mây vào chương trình chính

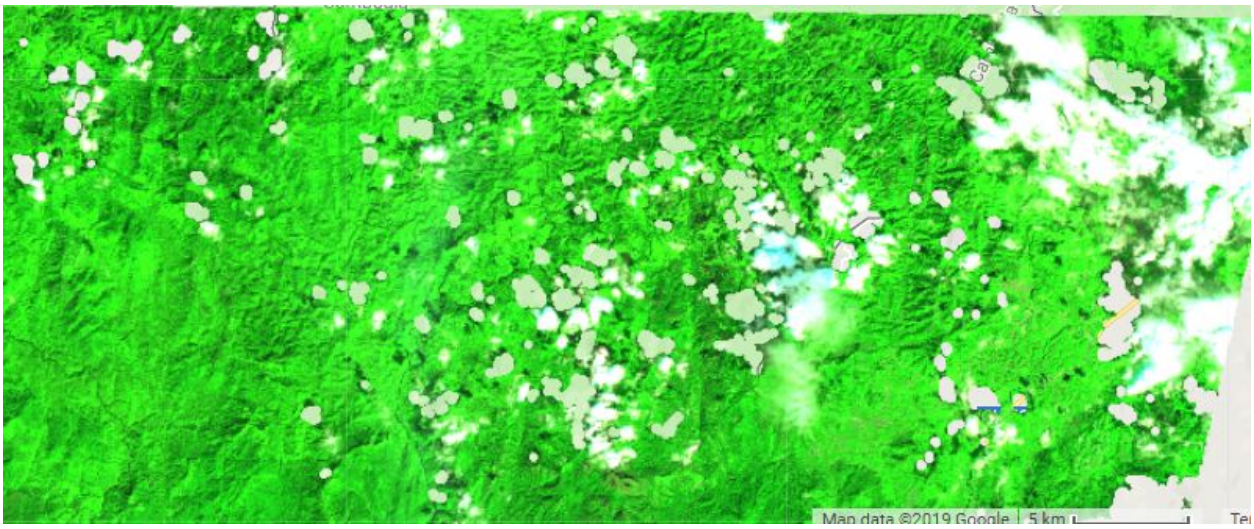
1. Thêm đoạn code dưới dây vào scrip **main** đã tạo ở bước 1:

```
var shadowS2 = require("users/username/Sentinel2:shadowMask_module");

print("applying cloud shadow mask");
s2 = shadowS2.shadowMask(s2,region);
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"Shadow
Mask");
print(ee.Image(s2.first()))
```

2. **Run/Chạy** code và **quan sát** kết quả

   i. Click nút Run và xem kết quả



ôô

"ẹ \ üÈ 4Ÿh f Hœ gŸ f vr mk~g mdi [ga 3^gmbg^e û

3.1. Tạo mô đun mặt nạ mây cho ảnh

1. Tạo mới một script, đặt tên là **cloud_module**

2. Dán đoạn code sau vào cloud_module:

```
var cloudScoreThresh = 20;
var cloudScorePctl = 0;
var contractPixels = 1.5;
```

```
var dilatePixels = 2.5;

////////////////////////////////////////////////////////////////////////////
/
// Compute a cloud score and adds a band that represents the cloud mask.
// Cloud masking algorithm for Sentinel2. Built on ideas from Landsat
cloudScore
// algorithm. Currently in beta and may need tweaking for individual study
areas
exports.sentinelCloudScore = function(s2s) {

function getCloudScore(img){
// Compute several indicators of cloudyness and take the minimum of them.
var score = ee.Image(1);
var blueCirrusScore = ee.Image(0);

// Clouds are reasonably bright in the blue or cirrus bands.
//Use .max as a pseudo OR conditional
blueCirrusScore = blueCirrusScore.max(rescale(img, 'img.blue', [0.1, 0.5]));
blueCirrusScore = blueCirrusScore.max(rescale(img, 'img.cb', [0.1, 0.5]));
blueCirrusScore = blueCirrusScore.max(rescale(img, 'img.cirrus', [0.1, 0.3]));
score = score.min(blueCirrusScore);

// Clouds are reasonably bright in all visible bands.
score = score.min(rescale(img, 'img.red + img.green + img.blue', [0.2, 0.8]));

// Clouds are reasonably bright in all infrared bands.
score = score.min(rescale(img, 'img.nir + img.swir1 + img.swir2', [0.3, 0.8]));

// However, clouds are not snow.
var ndsi = img.normalizedDifference(['green', 'swir1']);
score = score.min(rescale(ndsi, 'img', [0.8, 0.6]));

score = score.multiply(100).byte();
score = score.clamp(0,100);
return img.addBands(score.rename(['cloudScore']));
}
function maskScore(img){
//var cloudMask =
img.select(['cloudScore']).subtract(minCloudScore).lt(cloudScoreThresh)
// .focal_max(contractPixels).focal_min(dilatePixels).rename('cloudMask');
var cloudMask =
img.select(['cloudScore']).lt(cloudScoreThresh).focal_max(contractPixels).focal
_min(dilatePixels).rename('cloudMask');
return img.updateMask(cloudMask).addBands(cloudMask);
}

s2s = s2s.map(getCloudScore);

// Find low cloud score pctl for each pixel to avoid comission errors
var minCloudScore =
s2s.select(['cloudScore']).reduce(ee.Reducer.percentile([cloudScorePctl]));

s2s = s2s.map(maskScore);
```

```
return s2s;
};

////////////////////////////////////////////////////////////////////////////////
/
// Function to mask clouds using the Sentinel-2 QA band.
exports.QAMaskCloud= function(collection) {

function maskClouds(image){
var qa = image.select('QA60').int16();

// Bits 10 and 11 are clouds and cirrus, respectively.
var cloudBitMask = Math.pow(2, 10);
var cirrusBitMask = Math.pow(2, 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 the masked and scaled data.
return image.updateMask(mask);
}

collection = collection.map(maskClouds);

return collection;

};

////////////////////////////////////////////////////////////////////////////////
/
// Helper function to apply an expression and linearly rescale the output.
// Used in the sentinelCloudScore function below.
function rescale(img, exp, thresholds) {
return img.expression(exp, {img: img})
.subtract(thresholds[0]).divide(thresholds[1] - thresholds[0]);
}
```

3.2. Gọi mô đun mặt nạ mây vào chương trình chính

1. Thêm đoạn code dưới dây vào scrip **main** đã tạo ở bước 1:

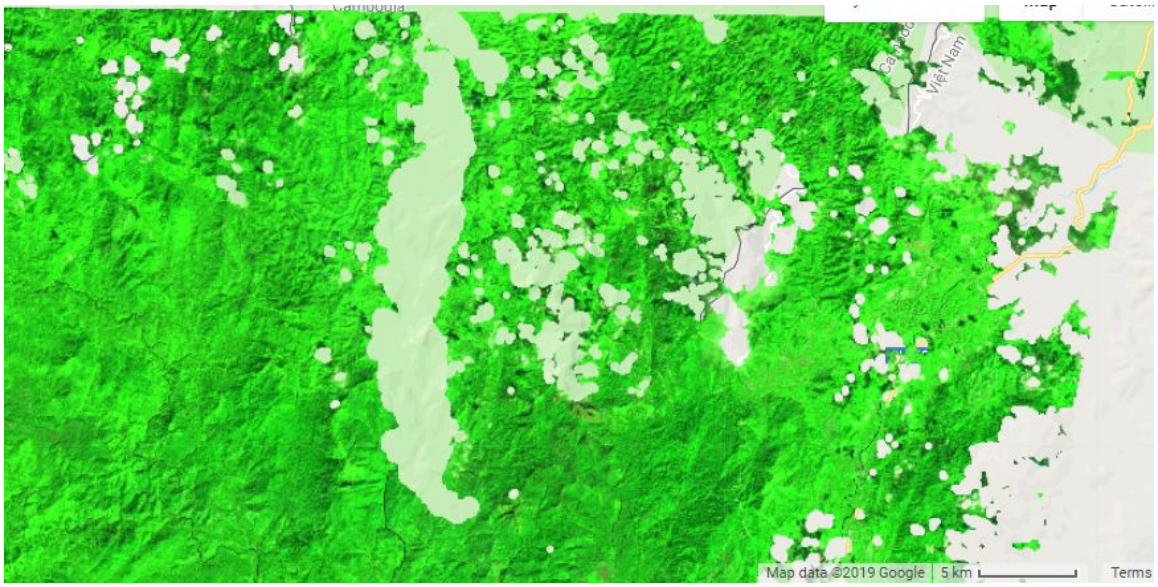
```
var clouds = require("users/username/Sentinel2:cloud_module");

print("applying QA cloud mask");
s2 = clouds.QAMaskCloud(s2);
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"after
QA cloudmask");
print(ee.Image(s2.first()));

print("applying sentinel cloud mask");
s2 = clouds.sentinelCloudScore(s2);
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"after
s2 cloudmask");
```

```
print(ee.Image(s2.first()));
```

2. Click vào nút Run xem kết quả



"Ẹ \ ýÈ (ḅ n \a̤ga `bu mḳ iaˇ iaʈg qЧmk~g at f iavg [´ û \abQn mug qЧmk~g đbWf ʈga

Hàm phân phối phản xạ hai chiều (BRDF) mô tả sự phụ thuộc theo hướng của năng lượng phản xạ của mục tiêu như là một chức năng chiếu sáng và xem hình học. BRDF phụ thuộc vào bước sóng. BRDF là thông số khá quan trọng trong viễn thám để hiệu chỉnh hiệu ứng góc nhìn và góc chiếu sáng.

ôô ỵÈúÈ Tạo mô đun hiệu chỉnh BRDF

1. Tạo mới một script đặt tên là **brdf_module**

2. Dán đoạn code dưới đây vào script **brdf_module** rồi lưu lại:

```
var PI = ee.Number(3.14159265359);
var MAX_SATELLITE_ZENITH = 7.5;
var MAX_DISTANCE = 1000000;
var UPPER_LEFT = 0;
var LOWER_LEFT = 1;
var LOWER_RIGHT = 2;
var UPPER_RIGHT = 3;

/* Creates a collection of mosaics with a given temporal interval.
 *
 * collection - the collection from which to make composites.
 * start - the date of the first composite (either a string or an ee.Date)
 * count - the number of composites to make
 * interval - The time between composites, in units of "units".
 * units - The units of step (day, week, month, year; see ee ee.Date.advance)
 */
exports.brdfS2 = function(collection) {
```

```
  collection = collection.map(applyBRDF);
  return collection;

  function applyBRDF(image){
    var date = image.date();
    var footprint =
ee.List(image.geometry().bounds().bounds().coordinates().get(0));
    var angles =  getsunAngles(date, footprint);
    var sunAz = angles[0];
    var sunZen = angles[1];

    var viewAz = azimuth(footprint);
    var viewZen = zenith(footprint);

    var kval = _kvol(sunAz, sunZen, viewAz, viewZen);
    var kvol = kval[0];
    var kvol0 = kval[1];
    var result = _apply(image, kvol.multiply(PI), kvol0.multiply(PI));

    return result;}

  /* Get sunAngles from the map given the data.
  *
  * date:  ee.date object
  * footprint: geometry of the image
  */
  function getsunAngles(date, footprint){
    var jdp = date.getFraction('year');
    var seconds_in_hour = 3600;
    var  hourGMT = ee.Number(date.getRelative('second',
'day')).divide(seconds_in_hour);

    var latRad =
ee.Image.pixelLonLat().select('latitude').multiply(PI.divide(180));
    var longDeg = ee.Image.pixelLonLat().select('longitude');

    // Julian day proportion in radians
    var jdpr = jdp.multiply(PI).multiply(2);

    var a = ee.List([0.000075, 0.001868, 0.032077, 0.014615, 0.040849]);
    var meanSolarTime = longDeg.divide(15.0).add(ee.Number(hourGMT));
    var localSolarDiff1 = value(a, 0)
            .add(value(a, 1).multiply(jdpr.cos()))
            .subtract(value(a, 2).multiply(jdpr.sin()))
            .subtract(value(a, 3).multiply(jdpr.multiply(2).cos()))
            .subtract(value(a, 4).multiply(jdpr.multiply(2).sin()));

    var localSolarDiff2 = localSolarDiff1.multiply(12 * 60);

    var localSolarDiff = localSolarDiff2.divide(PI);
    var trueSolarTime = meanSolarTime
            .add(localSolarDiff.divide(60))
            .subtract(12.0);
```

```
    // Hour as an angle;
    var ah = trueSolarTime.multiply(ee.Number(MAX_SATELLITE_ZENITH *
2).multiply(PI.divide(180))) ;
    var b = ee.List([0.006918, 0.399912, 0.070257, 0.006758, 0.000907,
0.002697, 0.001480]);
    var delta = value(b, 0)
          .subtract(value(b, 1).multiply(jdpr.cos()))
          .add(value(b, 2).multiply(jdpr.sin()))
          .subtract(value(b, 3).multiply(jdpr.multiply(2).cos()))
          .add(value(b, 4).multiply(jdpr.multiply(2).sin()))
          .subtract(value(b, 5).multiply(jdpr.multiply(3).cos()))
          .add(value(b, 6).multiply(jdpr.multiply(3).sin()));

    var cosSunZen = latRad.sin().multiply(delta.sin())
          .add(latRad.cos().multiply(ah.cos()).multiply(delta.cos()));
    var sunZen = cosSunZen.acos();

    // sun azimuth from south, turning west
    var sinSunAzSW = ah.sin().multiply(delta.cos()).divide(sunZen.sin());
    sinSunAzSW = sinSunAzSW.clamp(-1.0, 1.0);

    var cosSunAzSW = (latRad.cos().multiply(-1).multiply(delta.sin())

.add(latRad.sin().multiply(delta.cos()).multiply(ah.cos())))
                    .divide(sunZen.sin());
    var sunAzSW = sinSunAzSW.asin();

    sunAzSW = where(cosSunAzSW.lte(0), sunAzSW.multiply(-1).add(PI), sunAzSW);
    sunAzSW = where(cosSunAzSW.gt(0).and(sinSunAzSW.lte(0)),
sunAzSW.add(PI.multiply(2)), sunAzSW);

    var sunAz = sunAzSW.add(PI);
     // # Keep within [0, 2pi] range
    sunAz = where(sunAz.gt(PI.multiply(2)), sunAz.subtract(PI.multiply(2)),
sunAz);

    var footprint_polygon = ee.Geometry.Polygon(footprint);
    sunAz = sunAz.clip(footprint_polygon);
    sunAz = sunAz.rename(['sunAz']);
    sunZen = sunZen.clip(footprint_polygon).rename(['sunZen']);

    return [sunAz, sunZen];
  }

  /* Get azimuth.
  *
  *
  * footprint: geometry of the image
  */
  function azimuth(footprint){

    function x(point){return ee.Number(ee.List(point).get(0))}
    function  y(point){return ee.Number(ee.List(point).get(1))}
```

```
      var upperCenter = line_from_coords(footprint, UPPER_LEFT,
UPPER_RIGHT).centroid().coordinates();
      var lowerCenter = line_from_coords(footprint, LOWER_LEFT,
LOWER_RIGHT).centroid().coordinates();
      var slope =
((y(lowerCenter)).subtract(y(upperCenter))).divide((x(lowerCenter)).subtract(x(
upperCenter)));
      var slopePerp = ee.Number(-1).divide(slope);
      var azimuthLeft = ee.Image(PI.divide(2).subtract((slopePerp).atan()));
      return azimuthLeft.rename(['viewAz']);
  }

  /* Get zenith.
  *
  *
  * footprint: geometry of the image
  */
  function zenith(footprint){
      var leftLine = line_from_coords(footprint, UPPER_LEFT, LOWER_LEFT);
      var rightLine = line_from_coords(footprint, UPPER_RIGHT, LOWER_RIGHT);
      var leftDistance = ee.FeatureCollection(leftLine).distance(MAX_DISTANCE);
      var rightDistance =
ee.FeatureCollection(rightLine).distance(MAX_DISTANCE);
      var viewZenith = rightDistance.multiply(ee.Number(MAX_SATELLITE_ZENITH *
2))
          .divide(rightDistance.add(leftDistance))
          .subtract(ee.Number(MAX_SATELLITE_ZENITH))
          .clip(ee.Geometry.Polygon(footprint))
          .rename(['viewZen']);
    return viewZenith.multiply(PI.divide(180));
  }

  /* apply function to all bands
  *
  * http://www.mdpi.com/2072-4292/9/12/1325/htm#sec3dot2-remotesensing-09-01325
  * https://www.sciencedirect.com/science/article/pii/S0034425717302791
  *
  * image : the image to apply the function to
  * kvol:
  * kvol0
  *
  */
function _apply(image, kvol, kvol0){
      var f_iso = 0;
      var f_geo = 0;
      var f_vol = 0;
            var blue = _correct_band(image, 'blue', kvol, kvol0, f_iso=0.0774,
f_geo=0.0079, f_vol=0.0372);
            var green = _correct_band(image, 'green', kvol, kvol0,
f_iso=0.1306, f_geo=0.0178, f_vol=0.0580);
            var red = _correct_band(image, 'red', kvol, kvol0, f_iso=0.1690,
f_geo=0.0227, f_vol=0.0574);
```

```
            var re1 = _correct_band(image, 're1', kvol, kvol0, f_iso=0.2085,
f_geo=0.0256, f_vol=0.0845);
            var re2 = _correct_band(image, 're2', kvol, kvol0, f_iso=0.2316,
f_geo=0.0273, f_vol=0.1003);
            var re3 = _correct_band(image, 're3', kvol, kvol0, f_iso=0.2599,
f_geo=0.0294, f_vol=0.1197);
      var nir = _correct_band(image, 'nir', kvol, kvol0, f_iso=0.3093,
f_geo=0.0330, f_vol=0.1535);
      var re4 = _correct_band(image, 're4', kvol, kvol0, f_iso=0.2907,
f_geo=0.0410, f_vol=0.1611);
      var swir1 = _correct_band(image, 'swir1', kvol, kvol0, f_iso=0.3430,
f_geo=0.0453, f_vol=0.1154);
      var swir2 = _correct_band(image, 'swir2', kvol, kvol0, f_iso=0.2658,
f_geo=0.0387, f_vol=0.0639);
            return image.select([]).addBands([blue, green, red,
nir,re1,re2,re3,nir,re4,swir1, swir2]);
}

  /* correct band function
  *
  *
  * image : the image to apply the function to
  * band_name
  * kvol
  * kvol0
  * f_iso
  * f_geo
  * f_vol
  *
  */
  function _correct_band(image, band_name, kvol, kvol0, f_iso, f_geo, f_vol){
            //"""fiso + fvol * kvol + fgeo * kgeo"""
            var iso = ee.Image(f_iso);
            var geo = ee.Image(f_geo);
            var vol = ee.Image(f_vol);
            var pred =
vol.multiply(kvol).add(geo.multiply(kvol)).add(iso).rename(['pred']);
            var pred0 =
vol.multiply(kvol0).add(geo.multiply(kvol0)).add(iso).rename(['pred0']);
            var cfac = pred0.divide(pred).rename(['cfac']);
            var corr =
image.select(band_name).multiply(cfac).rename([band_name]);
            return corr;
  }

  /* calculate kvol and kvol0
  *
  * sunAZ
  * sunZen
  * viewAz
  * viewZen
  *
  */
  function _kvol(sunAz, sunZen, viewAz, viewZen){
```

```
            //"""Calculate kvol kernel.
            //From Lucht et al. 2000
            //Phase angle = cos(solar zenith) cos(view zenith) + sin(solar
zenith) sin(view zenith) cos(relative azimuth)"""

            var relative_azimuth = sunAz.subtract(viewAz).rename(['relAz']);
            var pa1 = viewZen.cos().multiply(sunZen.cos());
            var pa2 =
viewZen.sin().multiply(sunZen.sin()).multiply(relative_azimuth.cos());
            var phase_angle1 = pa1.add(pa2);
            var phase_angle = phase_angle1.acos();
            var p1 = ee.Image(PI.divide(2)).subtract(phase_angle);
            var p2 = p1.multiply(phase_angle1);
            var p3 = p2.add(phase_angle.sin());
            var p4 = sunZen.cos().add(viewZen.cos());
            var p5 = ee.Image(PI.divide(4));

            var kvol = p3.divide(p4).subtract(p5).rename(['kvol']);

            var viewZen0 = ee.Image(0);
            var pa10 = viewZen0.cos().multiply(sunZen.cos());
            var pa20 =
viewZen0.sin().multiply(sunZen.sin()).multiply(relative_azimuth.cos());
            var phase_angle10 = pa10.add(pa20);
            var phase_angle0 = phase_angle10.acos();
            var p10 = ee.Image(PI.divide(2)).subtract(phase_angle0);
            var p20 = p10.multiply(phase_angle10);
            var p30 = p20.add(phase_angle0.sin());
            var p40 = sunZen.cos().add(viewZen0.cos());
            var p50 = ee.Image(PI.divide(4));

            var kvol0 = p30.divide(p40).subtract(p50).rename(['kvol0']);

            return [kvol, kvol0]}

  /* helper function
  *
  *
  *
  */

  function line_from_coords(coordinates, fromIndex, toIndex){
      return ee.Geometry.LineString(ee.List([
        coordinates.get(fromIndex),
        coordinates.get(toIndex)]));
  }

  function where(condition, trueValue, falseValue){
        var trueMasked = trueValue.mask(condition);
        var falseMasked = falseValue.mask(invertMask(condition));
        return trueMasked.unmask(falseMasked);
    }

    function invertMask(mask){
```

```
      return mask.multiply(-1).add(1);
    }

    function value(list,index){
      return ee.Number(list.get(index));
    }

};

/* Creates a collection of mosaics with a given temporal interval.
 *
 * collection - the collection from which to make composites.
 * start - the date of the first composite (either a string or an ee.Date)
 * count - the number of composites to make
 * interval - The time between composites, in units of "units".
 * units - The units of step (day, week, month, year; see ee ee.Date.advance)
 */
exports.brdfL8 = function(image) {

  var date = image.date();
  var footprint =
ee.List(image.geometry().bounds().bounds().coordinates().get(0));
  var angles =  getsunAngles(date, footprint);
  var sunAz = angles[0];
  var sunZen = angles[1];

  var viewAz = azimuth(footprint);
  var viewZen = zenith(footprint);

  var kval = _kvol(sunAz, sunZen, viewAz, viewZen);
  var kvol = kval[0];
  var kvol0 = kval[1];
  var result = _apply(image, kvol.multiply(PI), kvol0.multiply(PI));

  return result;

  /* Get sunAngles from the map given the data.
  *
  * date:  ee.date object
  * footprint: geometry of the image
  */
  function getsunAngles(date, footprint){
    var jdp = date.getFraction('year');
    var seconds_in_hour = 3600;
    var  hourGMT = ee.Number(date.getRelative('second',
'day')).divide(seconds_in_hour);

    var latRad =
ee.Image.pixelLonLat().select('latitude').multiply(PI.divide(180));
    var longDeg = ee.Image.pixelLonLat().select('longitude');

    // Julian day proportion in radians
    var jdpr = jdp.multiply(PI).multiply(2);
```

```
    var a = ee.List([0.000075, 0.001868, 0.032077, 0.014615, 0.040849]);
    var meanSolarTime = longDeg.divide(15.0).add(ee.Number(hourGMT));
    var localSolarDiff1 = value(a, 0)
            .add(value(a, 1).multiply(jdpr.cos()))
            .subtract(value(a, 2).multiply(jdpr.sin()))
            .subtract(value(a, 3).multiply(jdpr.multiply(2).cos()))
            .subtract(value(a, 4).multiply(jdpr.multiply(2).sin()));

    var localSolarDiff2 = localSolarDiff1.multiply(12 * 60);

    var localSolarDiff = localSolarDiff2.divide(PI);
    var trueSolarTime = meanSolarTime
            .add(localSolarDiff.divide(60))
            .subtract(12.0);

    // Hour as an angle;
    var ah = trueSolarTime.multiply(ee.Number(MAX_SATELLITE_ZENITH *
2).multiply(PI.divide(180))) ;
    var b = ee.List([0.006918, 0.399912, 0.070257, 0.006758, 0.000907,
0.002697, 0.001480]);
    var delta = value(b, 0)
          .subtract(value(b, 1).multiply(jdpr.cos()))
          .add(value(b, 2).multiply(jdpr.sin()))
          .subtract(value(b, 3).multiply(jdpr.multiply(2).cos()))
          .add(value(b, 4).multiply(jdpr.multiply(2).sin()))
          .subtract(value(b, 5).multiply(jdpr.multiply(3).cos()))
          .add(value(b, 6).multiply(jdpr.multiply(3).sin()));

    var cosSunZen = latRad.sin().multiply(delta.sin())
          .add(latRad.cos().multiply(ah.cos()).multiply(delta.cos()));
    var sunZen = cosSunZen.acos();

    // sun azimuth from south, turning west
    var sinSunAzSW = ah.sin().multiply(delta.cos()).divide(sunZen.sin());
    sinSunAzSW = sinSunAzSW.clamp(-1.0, 1.0);

    var cosSunAzSW = (latRad.cos().multiply(-1).multiply(delta.sin())

.add(latRad.sin().multiply(delta.cos()).multiply(ah.cos())))
                    .divide(sunZen.sin());
    var sunAzSW = sinSunAzSW.asin();

    sunAzSW = where(cosSunAzSW.lte(0), sunAzSW.multiply(-1).add(PI), sunAzSW);
    sunAzSW = where(cosSunAzSW.gt(0).and(sinSunAzSW.lte(0)),
sunAzSW.add(PI.multiply(2)), sunAzSW);

    var sunAz = sunAzSW.add(PI);
     // # Keep within [0, 2pi] range
    sunAz = where(sunAz.gt(PI.multiply(2)), sunAz.subtract(PI.multiply(2)),
sunAz);

    var footprint_polygon = ee.Geometry.Polygon(footprint);
    sunAz = sunAz.clip(footprint_polygon);
    sunAz = sunAz.rename(['sunAz']);
```

```
    sunZen = sunZen.clip(footprint_polygon).rename(['sunZen']);

    return [sunAz, sunZen];
  }

  /* Get azimuth.
  *
  *
  * footprint: geometry of the image
  */
  function azimuth(footprint){

    function x(point){return ee.Number(ee.List(point).get(0))}
    function  y(point){return ee.Number(ee.List(point).get(1))}

      var upperCenter = line_from_coords(footprint, UPPER_LEFT,
UPPER_RIGHT).centroid().coordinates();
      var lowerCenter = line_from_coords(footprint, LOWER_LEFT,
LOWER_RIGHT).centroid().coordinates();
      var slope =
((y(lowerCenter)).subtract(y(upperCenter))).divide((x(lowerCenter)).subtract(x(
upperCenter)));
      var slopePerp = ee.Number(-1).divide(slope);
      var azimuthLeft = ee.Image(PI.divide(2).subtract((slopePerp).atan()));
      return azimuthLeft.rename(['viewAz']);
  }

  /* Get zenith.
  *
  *
  * footprint: geometry of the image
  */
  function zenith(footprint){
      var leftLine = line_from_coords(footprint, UPPER_LEFT, LOWER_LEFT);
      var rightLine = line_from_coords(footprint, UPPER_RIGHT, LOWER_RIGHT);
      var leftDistance = ee.FeatureCollection(leftLine).distance(MAX_DISTANCE);
      var rightDistance =
ee.FeatureCollection(rightLine).distance(MAX_DISTANCE);
      var viewZenith = rightDistance.multiply(ee.Number(MAX_SATELLITE_ZENITH *
2))
          .divide(rightDistance.add(leftDistance))
          .subtract(ee.Number(MAX_SATELLITE_ZENITH))
          .clip(ee.Geometry.Polygon(footprint))
          .rename(['viewZen']);
    return viewZenith.multiply(PI.divide(180));
  }

  /* apply function to all bands
  *
  * http://www.mdpi.com/2072-4292/9/12/1325/htm#sec3dot2-remotesensing-09-01325
  * https://www.sciencedirect.com/science/article/pii/S0034425717302791
  *
  * image : the image to apply the function to
  * kvol:
```

```
  * kvol0
  *
  */
function _apply(image, kvol, kvol0){
      var f_iso = 0;
      var f_geo = 0;
      var f_vol = 0;
            var blue = _correct_band(image, 'blue', kvol, kvol0, f_iso=0.0774,
f_geo=0.0079, f_vol=0.0372);
            var green = _correct_band(image, 'green', kvol, kvol0,
f_iso=0.1306, f_geo=0.0178, f_vol=0.0580);
            var red = _correct_band(image, 'red', kvol, kvol0, f_iso=0.1690,
f_geo=0.0227, f_vol=0.0574);
      var nir = _correct_band(image, 'nir', kvol, kvol0, f_iso=0.3093,
f_geo=0.0330, f_vol=0.1535);
      var swir1 = _correct_band(image, 'swir1', kvol, kvol0, f_iso=0.3430,
f_geo=0.0453, f_vol=0.1154);
      var swir2 = _correct_band(image, 'swir2', kvol, kvol0, f_iso=0.2658,
f_geo=0.0387, f_vol=0.0639);
            return image.select([]).addBands([blue, green, red, nir, swir1,
swir2]);
}

  /* correct band function
  *
  *
  * image : the image to apply the function to
  * band_name
  * kvol
  * kvol0
  * f_iso
  * f_geo
  * f_vol
  *
  */
  function _correct_band(image, band_name, kvol, kvol0, f_iso, f_geo, f_vol){
            //"""fiso + fvol * kvol + fgeo * kgeo"""
            var iso = ee.Image(f_iso);
            var geo = ee.Image(f_geo);
            var vol = ee.Image(f_vol);
            var pred =
vol.multiply(kvol).add(geo.multiply(kvol)).add(iso).rename(['pred']);
            var pred0 =
vol.multiply(kvol0).add(geo.multiply(kvol0)).add(iso).rename(['pred0']);
            var cfac = pred0.divide(pred).rename(['cfac']);
            var corr =
image.select(band_name).multiply(cfac).rename([band_name]);
            return corr;
  }

  /* calculate kvol and kvol0
  *
  * sunAZ
  * sunZen
```

```
  * viewAz
  * viewZen
  *
  */
  function _kvol(sunAz, sunZen, viewAz, viewZen){
            //"""Calculate kvol kernel.
            //From Lucht et al. 2000
            //Phase angle = cos(solar zenith) cos(view zenith) + sin(solar
zenith) sin(view zenith) cos(relative azimuth)"""

            var relative_azimuth = sunAz.subtract(viewAz).rename(['relAz']);
            var pa1 = viewZen.cos().multiply(sunZen.cos());
            var pa2 =
viewZen.sin().multiply(sunZen.sin()).multiply(relative_azimuth.cos());
            var phase_angle1 = pa1.add(pa2);
            var phase_angle = phase_angle1.acos();
            var p1 = ee.Image(PI.divide(2)).subtract(phase_angle);
            var p2 = p1.multiply(phase_angle1);
            var p3 = p2.add(phase_angle.sin());
            var p4 = sunZen.cos().add(viewZen.cos());
            var p5 = ee.Image(PI.divide(4));

            var kvol = p3.divide(p4).subtract(p5).rename(['kvol']);

            var viewZen0 = ee.Image(0);
            var pa10 = viewZen0.cos().multiply(sunZen.cos());
            var pa20 =
viewZen0.sin().multiply(sunZen.sin()).multiply(relative_azimuth.cos());
            var phase_angle10 = pa10.add(pa20);
            var phase_angle0 = phase_angle10.acos();
            var p10 = ee.Image(PI.divide(2)).subtract(phase_angle0);
            var p20 = p10.multiply(phase_angle10);
            var p30 = p20.add(phase_angle0.sin());
            var p40 = sunZen.cos().add(viewZen0.cos());
            var p50 = ee.Image(PI.divide(4));

            var kvol0 = p30.divide(p40).subtract(p50).rename(['kvol0']);

            return [kvol, kvol0]}

  /* helper function
    */

  function line_from_coords(coordinates, fromIndex, toIndex){
      return ee.Geometry.LineString(ee.List([
        coordinates.get(fromIndex),
        coordinates.get(toIndex)]));
  }

  function where(condition, trueValue, falseValue){
        var trueMasked = trueValue.mask(condition);
        var falseMasked = falseValue.mask(invertMask(condition));
        return trueMasked.unmask(falseMasked);
    }
```

```
    function invertMask(mask){
      return mask.multiply(-1).add(1);
    }

    function value(list,index){
      return ee.Number(list.get(index));
    }

};
```

4.2. Gọi mô đun hiệu chỉnh BRDF vào chương trình chính

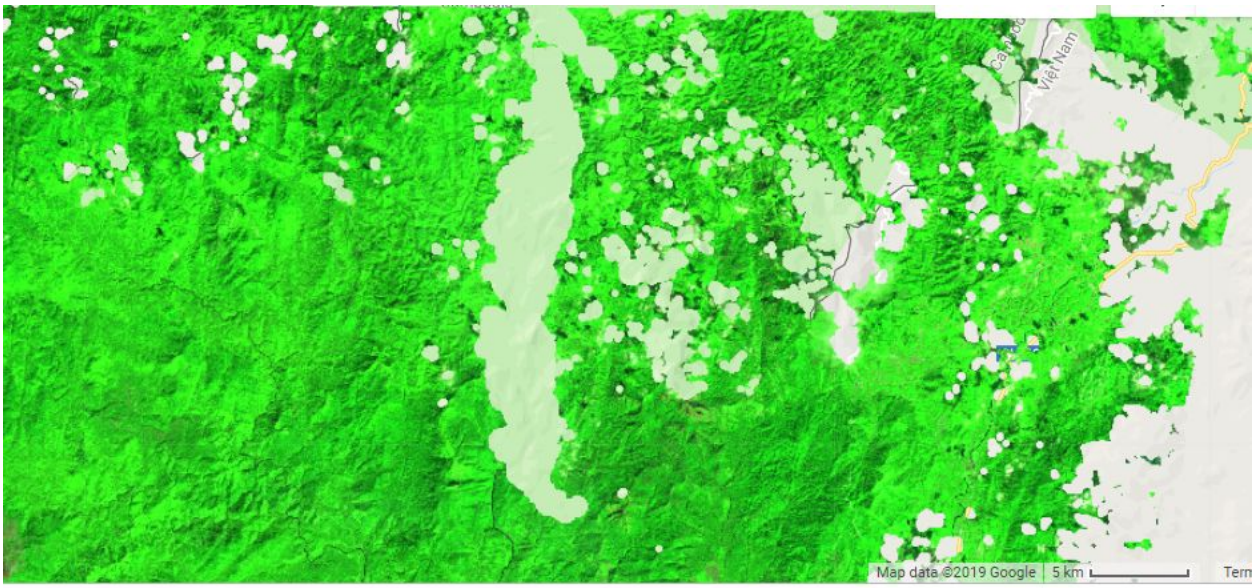
1. Bổ sung đoạn code dưới đây vào script **main**

```
var brdf = require("users/username/Sentinel2:brdf_module");

print("applying sentinel brdf");
s2 =  brdf.brdfS2(s2);
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"after
brdf");
```

```
print(ee.Image(s2.first()));
```

2. Chạy và kiểm tra kết quả



"Ẹ \ þÈ (ḅ n \a¨ga Ґga aЄ g` đ Z a¡ga

5.1. Tạo script mô đun hiệu chỉnh ảnh hưởng địa hình

úÈ 4ạh f ớb l\kb i m đặm m~g e t **topographic_module,**]ug đhạg \h]^ lZn o t eưn eạbÇ

```
var scale = 300;
var toaOrSR = 'SR';
```

```
// get terrain layers
var dem = ee.Image("USGS/SRTMGL1_003");
var degree2radian = 0.01745;

exports.topoCorrection = function(collection) {

  collection = collection.map(illuminationCondition);
  collection = collection.map(illuminationCorrection);

  //collection = correction.merge(notcorrection).sort("system:time_start");

  return(collection);

////////////////////////////////////////////////////////////////////////////////
  // Function to calculate illumination condition (IC). Function by Patrick
Burns
  // (pb463@nau.edu) and Matt Macander
  // (mmacander@abrinc.com)
  function illuminationCondition(img){

  // Extract image metadata about solar position
  var SZ_rad =
ee.Image.constant(ee.Number(img.get('MEAN_SOLAR_ZENITH_ANGLE'))).multiply(3.141
59265359).divide(180).clip(img.geometry().buffer(10000));
  var SA_rad =
ee.Image.constant(ee.Number(img.get('MEAN_SOLAR_AZIMUTH_ANGLE')).multiply(3.141
59265359).divide(180)).clip(img.geometry().buffer(10000));
  // Creat terrain layers
  var slp = ee.Terrain.slope(dem).clip(img.geometry().buffer(10000));
  var slp_rad =
ee.Terrain.slope(dem).multiply(3.14159265359).divide(180).clip(img.geometry().b
uffer(10000));
  var asp_rad =
ee.Terrain.aspect(dem).multiply(3.14159265359).divide(180).clip(img.geometry().
buffer(10000));

  // Calculate the Illumination Condition (IC)
  // slope part of the illumination condition
  var cosZ = SZ_rad.cos();
  var cosS = slp_rad.cos();
  var slope_illumination = cosS.expression("cosZ * cosS",
                                           {'cosZ': cosZ,
                                            'cosS': cosS.select('slope')});
  // aspect part of the illumination condition
  var sinZ = SZ_rad.sin();
  var sinS = slp_rad.sin();
  var cosAziDiff = (SA_rad.subtract(asp_rad)).cos();
  var aspect_illumination = sinZ.expression("sinZ * sinS * cosAziDiff",
                                            {'sinZ': sinZ,
                                             'sinS': sinS,
                                             'cosAziDiff': cosAziDiff});
  // full illumination condition (IC)
  var ic = slope_illumination.add(aspect_illumination);
```

```
  // Add IC to original image
  var img_plus_ic =
ee.Image(img.addBands(ic.rename('IC')).addBands(cosZ.rename('cosZ')).addBands(c
osS.rename('cosS')).addBands(slp.rename('slope')));
  return img_plus_ic;
  }

////////////////////////////////////////////////////////////////////////////
/
  // Function to apply the Sun-Canopy-Sensor + C (SCSc) correction method to
each
  // image. Function by Patrick Burns (pb463@nau.edu) and Matt Macander
  // (mmacander@abrinc.com)
  function illuminationCorrection(img){
    var props = img.toDictionary();
    var st = img.get('system:time_start');

    var img_plus_ic = img;
    var mask1 = img_plus_ic.select('nir').gt(-0.1);
    var mask2 = img_plus_ic.select('slope').gte(5)
                           .and(img_plus_ic.select('IC').gte(0))
                           .and(img_plus_ic.select('nir').gt(-0.1));
    var img_plus_ic_mask2 = ee.Image(img_plus_ic.updateMask(mask2));

    // Specify Bands to topographically correct
    var bandList = ['blue','green','red','nir','swir1','swir2'];
    var compositeBands = img.bandNames();
    var nonCorrectBands = img.select(compositeBands.removeAll(bandList));

    var geom = ee.Geometry(img.get('system:footprint')).bounds().buffer(10000);

    function apply_SCSccorr(band){
      var method = 'SCSc';
      var out =  ee.Image(1).addBands(img_plus_ic_mask2.select('IC', band))
                           .reduceRegion({reducer:
ee.Reducer.linearRegression(2,1),
                                          geometry:
ee.Geometry(img.geometry()),
                                          scale: scale,
                                          bestEffort :true,
                                          maxPixels:1e10});

        var fit = out.combine({"coefficients": ee.Array([[1],[1]])}, false);

                //Get the coefficients as a nested list,
                // ast it to an array, and get just the selected column
                var out_a = (ee.Array(fit.get('coefficients')).get([0,0]));
                var out_b = (ee.Array(fit.get('coefficients')).get([1,0]));
                var out_c = out_a.divide(out_b);

        // Apply the SCSc correction
        var SCSc_output = img_plus_ic_mask2.expression(
        "((image * (cosB * cosZ + cvalue)) / (ic + cvalue))", {
        'image': img_plus_ic_mask2.select(band),
```

```
        'ic': img_plus_ic_mask2.select('IC'),
        'cosB': img_plus_ic_mask2.select('cosS'),
        'cosZ': img_plus_ic_mask2.select('cosZ'),
        'cvalue': out_c
        });

      return SCSc_output;

    }

    var img_SCSccorr =
ee.Image(bandList.map(apply_SCSccorr)).addBands(img_plus_ic.select('IC'));
    var bandList_IC = ee.List([bandList, 'IC']).flatten();
    img_SCSccorr =
img_SCSccorr.unmask(img_plus_ic.select(bandList_IC)).select(bandList);

    return img_SCSccorr.addBands(nonCorrectBands)
      .setMulti(props)
      .set('system:time_start',st);
  }

};
```

5.2. Kết nối Mô dun hiệu chỉnh địa hình vào chương trình chính (main)

1. M˪ script **main** sau Ľ· b˚ sung th°m nh˙ng d˛ng l˘nh sau:

```
var topography = require("users/username/Sentinel2:topographic_module");

print("applying illumination correction");
s2 =  topography.topoCorrection(s2);
Map.addLayer(ee.Image(s2.first()),{min:0,max:0.3,bands:"swir2,nir,red"},"after
terrain correction");
print(ee.Image(s2.first()));
```

2. Chýy th˛ v¨ kiqm tra kεt quч