SKYWATCH

esa

SENTINEL-1

Flood mapping using Snappy

Issued February 2019

Alex McVittie

Prerequisites

Before beginning, you must have the snap python libraries prepared, and the two libraries `pygeoif` and `pyshp` installed (available through pip or conda). For more information on getting snappy set up, please refer to this tutorial (https://senbox.atlassian.net/wiki/spaces/SNAP/pages/19300362/How+to+use+the+SNAP+API+ for set up. This is NOT the same package that is available through pip. Also please note that it is currently only supported on Python 2.7 and 3.4. Be sure that you are running the latest version of SNAP. This tutorial is written to use the processing framework provided in SNAP 6.0. Running an older version is not tested, and there are no guarantees that it will function properly on other versions of SNAP.

At the end of this tutorial, a full list of all SNAP processing tool function names are available for future reference, as well as a comprehensive script containing all the code snippets in one runnable section.

Introduction

Synthetic Aperture Radar (SAR) is a type of sensor that captures earth observation (EO) data at a wavelength shorter than the visible light spectrum, allowing it to penetrate through clouds, and observe during both day and night, unlike an optical sensor such as LANDSAT which cannot observe the Earth's surface at night or during cloudy conditions. This gives satellites collecting SAR data a unique advantage of continuous coverage of the earth's surface, even during inclement weather.

One of the rainiest locations on Earth, the Island of Kauai in Hawaii, USA, was recently affected by extreme weather, with 685 mm of rainfall over a 24 hour period in April 2018. By using SAR data and the tools available to us through SNAP, we can see what areas were affected by flooding through a simplified classification methodology.

Purpose of tutorial

This tutorial aims to familiarize the user with common SAR processing tools available in SNAP using snappy and python, in particular:

- basic product information queries
- terrain correction
- de-speckling
- image calibration
- orbit file application
- subsetting of data using vector file input
- image reclassification using band math calculations
- land cover downloading
- exporting of product objects to files.

This tutorial aims to apply these skills with some basic flood mapping analysis.

Reading products with snappy

Reading in products with Snappy is simple - you do not even need to have the data unzipped. You'll need to download two datasets for this tutorial - a boundary of Hawaii and the Sentinel 1 product. The Sentinel 1 product can be accessed through Copernicus Scihub, the full product name is S1A_IW_GRDH_1SDV_20180415T163146_20180415T163211_021480_025003_8E79. The island boundary file can be downloaded here (https://s3-us-west-2.amazonaws.com/skywatch-public/snap/shared/sentinel_flood_mapping_tutorial/island_boundary.zip)

You'll need to unzip the island_boundary.zip archive, but you do not need to unzip the S1A zip archive. For this tutorial, I put this data in a new folder named data, but you can put it anywhere you wish, just be sure to update the path information in the code.

```
%matplotlib inline

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import os

import snappy
from snappy import Product
from snappy import ProductIO
from snappy import ProductUtils
from snappy import WKTReader
from snappy import HashMap
from snappy import GPF

# For shapefiles
import shapefile
import pygeoif

path_to_sentinel_data =
"data/S1A_IW_GRDH_1SDV_20180415T163146_20180415T163211_021480_025003_8E79.zip"

product = ProductIO.readProduct(path_to_sentinel_data)
```

With our ZIP file now read into snappy as a SNAP product object, we can get some basic information printed about the product.

```
width = product.getSceneRasterWidth()
print("Width: {} px".format(width))
height = product.getSceneRasterHeight()
print("Height: {} px".format(height))
name = product.getName()
print("Name: {}".format(name))
band_names = product.getBandNames()
print("Band names: {}".format(", ".join(band_names)))
```

```
Width: 25220 px
Height: 16774 px
Name: S1A_IW_GRDH_1SDV_20180415T163146_20180415T163211_021480_025003_8E79
Band names: Amplitude_VH, Intensity_VH, Amplitude_VV, Intensity_VV
```

Lets create a method to show a product inline. The whole product is too big to be shown in line, but we will use it later.

```
def plotBand(product, band, vmin, vmax):

    band = product.getBand(band)
    w = band.getRasterWidth()
    h = band.getRasterHeight()
    print(w, h)

    band_data = np.zeros(w * h, np.float32)
    band.readPixels(0, 0, w, h, band_data)

    band_data.shape = h, w

    width = 12
    height = 12
    plt.figure(figsize=(width, height))
    imgplot = plt.imshow(band_data, cmap=plt.cm.binary, vmin=vmin, vmax=vmax)

    return imgplot
```

Image pre-processing

Orbit file application

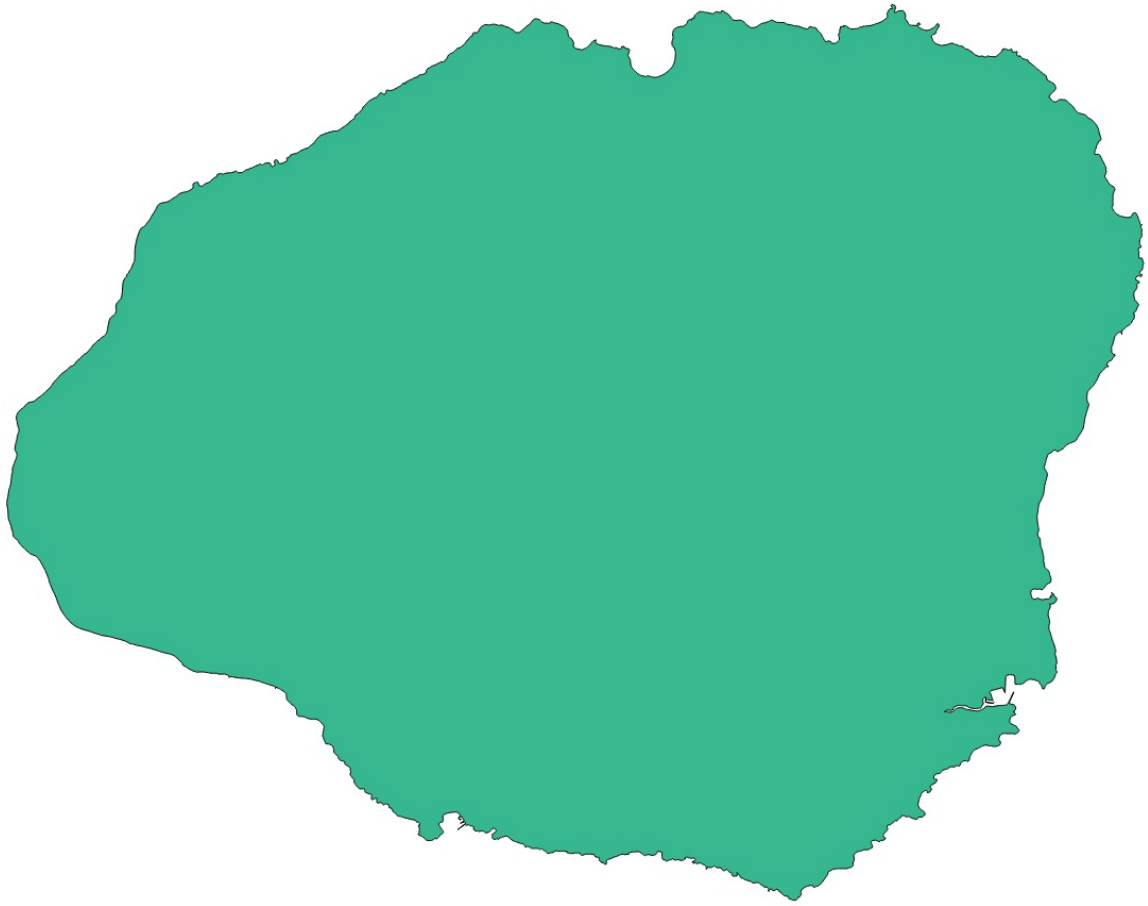
Before any SAR pre-processing steps occur, the product subset should be properly orthorectified to improve accuracy. To properly orthorectify the image, the orbit file is applied using the `Apply-Orbit-File` GPF module. SNAP is able to generate and apply a high accuracy satellite orbit file to a product.

```
parameters = HashMap()

GPF.getDefaultInstance().getOperatorSpiRegistry().loadOperatorSpis()

parameters.put('orbitType', 'Sentinel Precise (Auto Download)')
parameters.put('polyDegree', '3')
parameters.put('continueOnFail', 'false')

apply_orbit_file = GPF.createProduct('Apply-Orbit-File', parameters, product)
```

Clipping/subsetting images

The raw image is much larger than what we need to process, so to make the processing require less resources, it is important that we first extract a subset from the image. This step could be skipped, but processing the entire scene is unnecessary and time consuming.

To clip our image, the easiest way to do so is to convert a shape boundary file into a WKT (Well Known Text) string. This is easily obtained by using a few python modules. Our shapefile of the island looks like this:



The shapefile is provided with the tutorial, and was generated from the [State of Hawaii's planning department (http://planning.hawaii.gov/gis/download-gis-data/)](http://planning.hawaii.gov/gis/download-gis-data/) with a few modifications (selecting out the one island we are interested in, dissolving the individual electoral boundaries into one shape file, and reprojecting to EPSG:4326).

```
r = shapefile.Reader("data/island_boundary2.shp")

g=[]

for s in r.shapes():
    g.append(pygeoif.geometry.as_shape(s))

m = pygeoif.MultiPoint(g)

wkt = str(m.wkt).replace("MULTIPOINT", "POLYGON(") + ")"
```

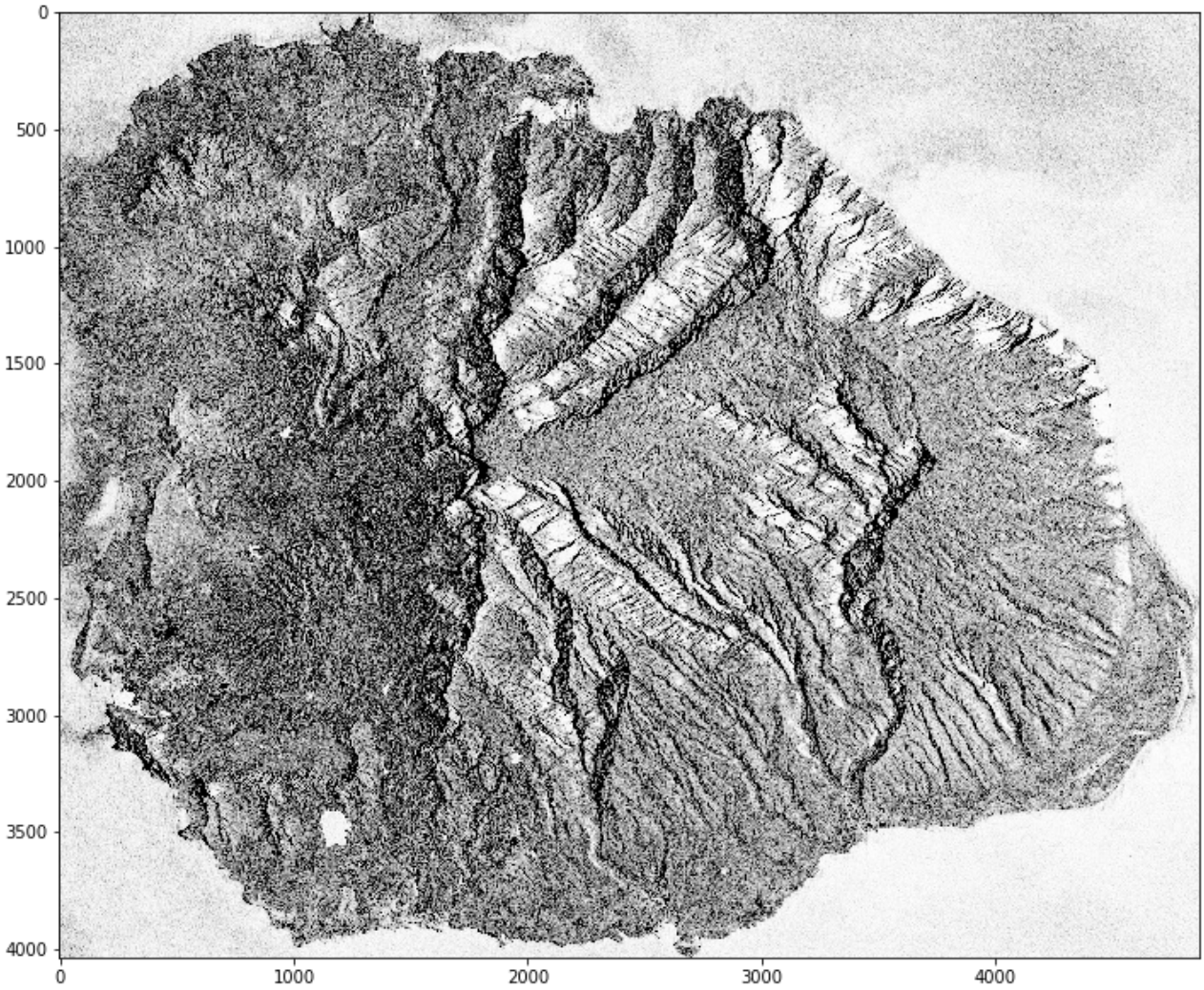
With our WKT string generated, it is time to use it to subset our data.

```
SubsetOp = snappy.jpy.get_type('org.esa.snap.core.gpf.common.SubsetOp')

bounding_wkt = wkt

geometry = WKTReader().read(bounding_wkt)

HashMap = snappy.jpy.get_type('java.util.HashMap')
GPF.getDefaultInstance().getOperatorSpiRegistry().loadOperatorSpis()
parameters = HashMap()
parameters.put('copyMetadata', True)
parameters.put('geoRegion', geometry)
product_subset = snappy.GPF.createProduct('Subset', parameters,
apply_orbit_file)
```

If we look at the width and height of the subset, we can see that it is now a much smaller product than our original, with all the original bands still available. We can also display the subset using the plotBand method we defined earlier.

```
width = product_subset.getSceneRasterWidth()
print("Width: {} px".format(width))
height = product_subset.getSceneRasterHeight()
print("Height: {} px".format(height))
band_names = product_subset.getBandNames()
print("Band names: {}".format(", ".join(band_names)))
band = product_subset.getBand(band_names[0])
print(band.getRasterSize())

plotBand(product_subset, "Intensity_VV", 0, 100000)
```

```
Width: 4871 px
Height: 4035 px
Band names: Amplitude_VH, Intensity_VH, Amplitude_VV, Intensity_VV
java.awt.Dimension[width=4871,height=4035]
(4871L, 4035L)




<matplotlib.image.AxesImage at 0x7fb7dbe67a90>
```



With our image now subsetted to the area we want to run our analysis on, we now need to apply some preprocessing and corrections to the data. Our end goal is to determine flooding with delineation techniques - literature shows that the VV polarization has advantages for flood delineation calculations over other polarizations, so we will apply processing on the VV intensity band. (https://onlinelibrary.wiley.com/doi/full/10.1111/jfr3.12303)

Image calibration

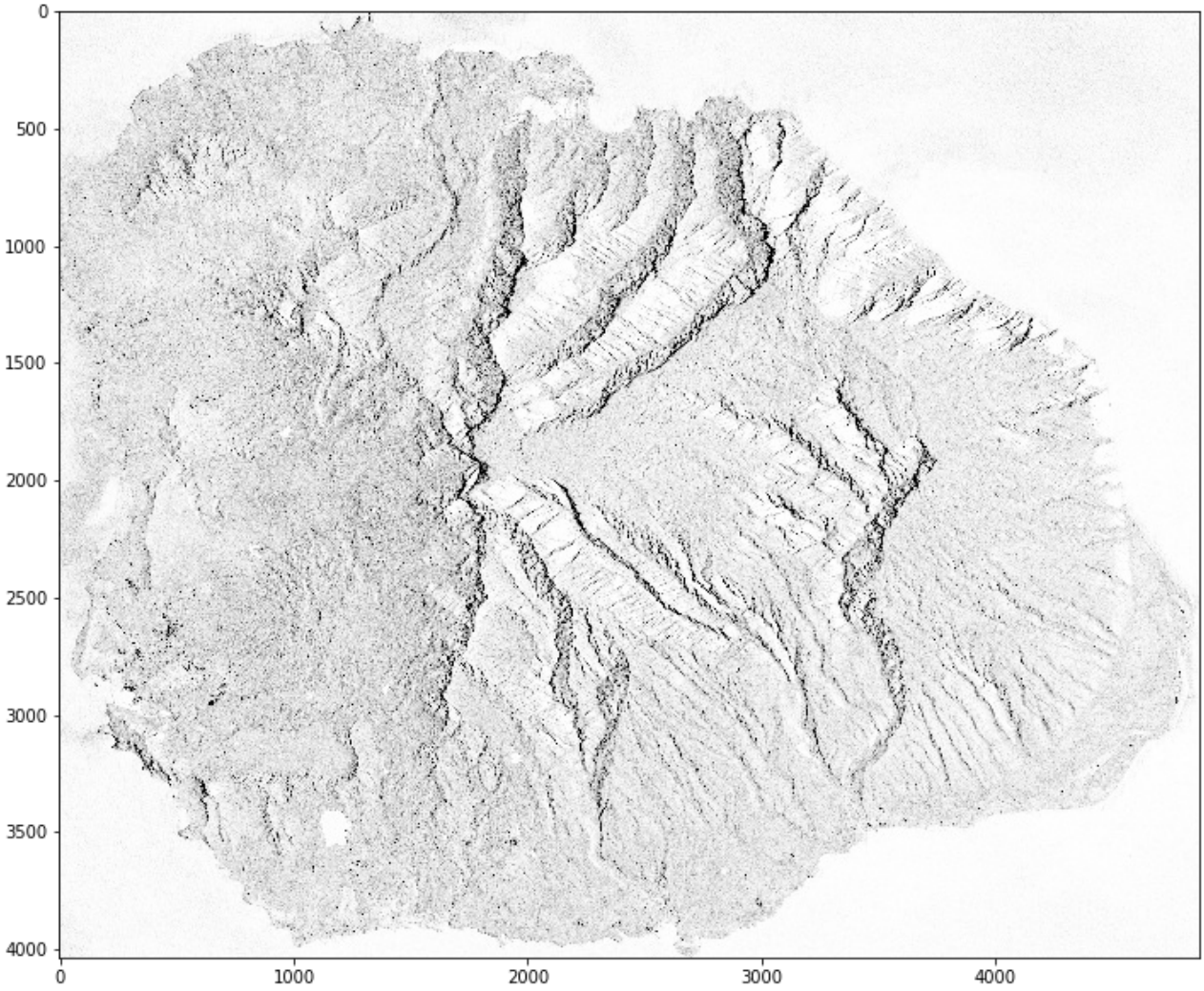
With our data now properly orthorectified with a high-quality generated orbit file, it is now time to calibrate the data to sigma-naught values.

```
parameters = HashMap()
parameters.put('outputSigmaBand', True)
parameters.put('sourceBands', 'Intensity_VV')
parameters.put('selectedPolarisations', "VV")
parameters.put('outputImageScaleInDb', False)

product_calibrated = GPF.createProduct("Calibration", parameters,
product_subset)
```

The data is now stored in a band called `Sigma0_VV` in the product object `product_calibrated` with a much smaller range of values.

```
plotBand(product_calibrated, "Sigma0_VV", 0, 1)
```

```
(4871L, 4035L)



<matplotlib.image.AxesImage at 0x7fb7e03ea9d0>
```

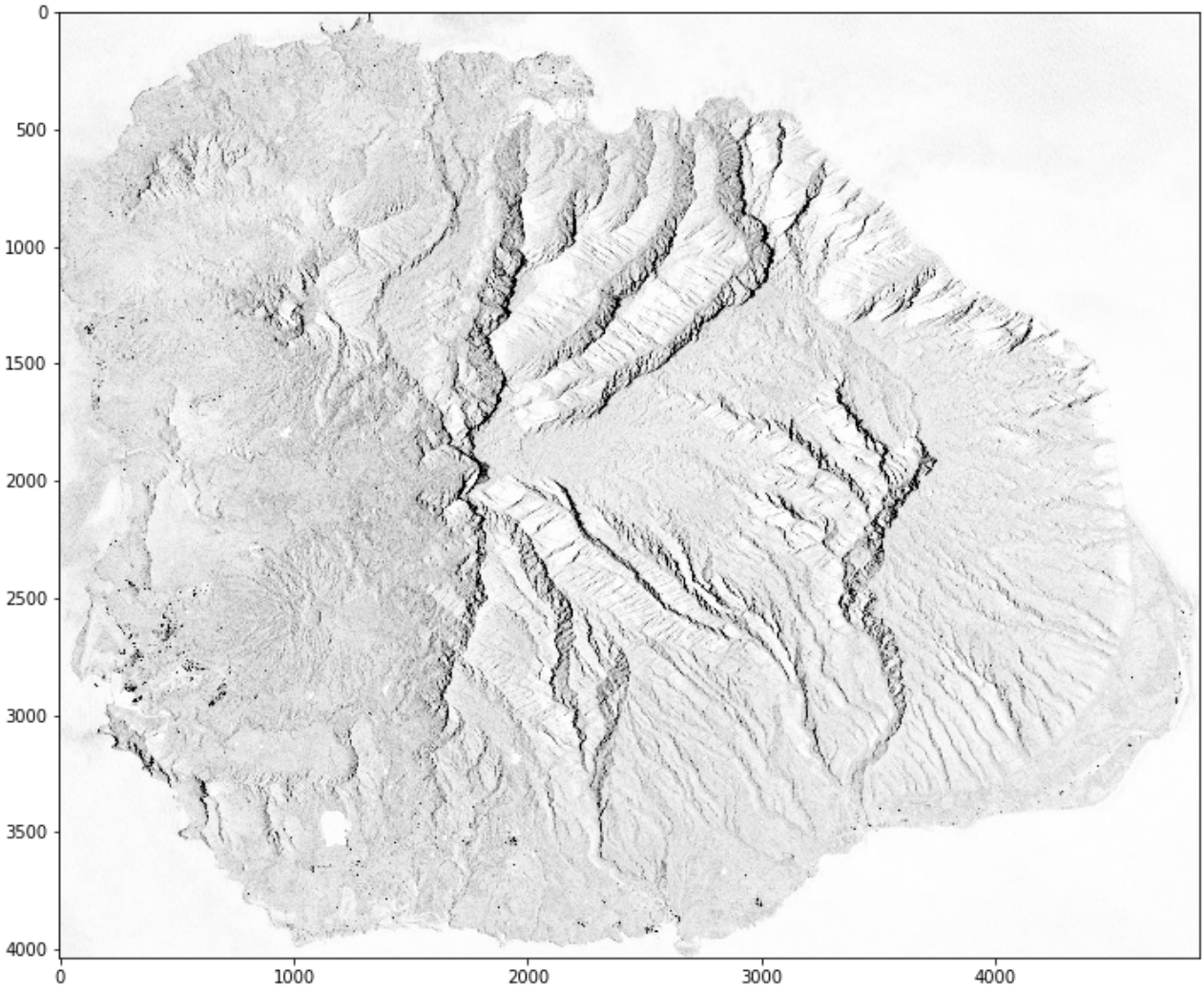


Speckle filter

In the next pre-processing section, we must apply a speckle filter to our area of interest. This removes [Speckle noise (https://en.wikipedia.org/wiki/Speckle_noise)](https://en.wikipedia.org/wiki/Speckle_noise) from the imagery.

```
filterSizeY = '5'
filterSizeX = '5'
parameters = HashMap()

parameters.put('sourceBands', 'Sigma0_VV')
parameters.put('filter', 'Lee')
parameters.put('filterSizeX', filterSizeX)
parameters.put('filterSizeY', filterSizeY)
parameters.put('dampingFactor', '2')
parameters.put('estimateENL', 'true')
parameters.put('enl', '1.0')
parameters.put('numLooksStr', '1')
parameters.put('targetWindowSizeStr', '3x3')
parameters.put('sigmaStr', '0.9')
parameters.put('anSize', '50')

speckle_filter = snappy.GPF.createProduct('Speckle-Filter', parameters,
product_calibrated)

plotBand(speckle_filter, 'Sigma0_VV', 0, 1)
```

```
(4871L, 4035L)



<matplotlib.image.AxesImage at 0x7fb7e0373b90>
```

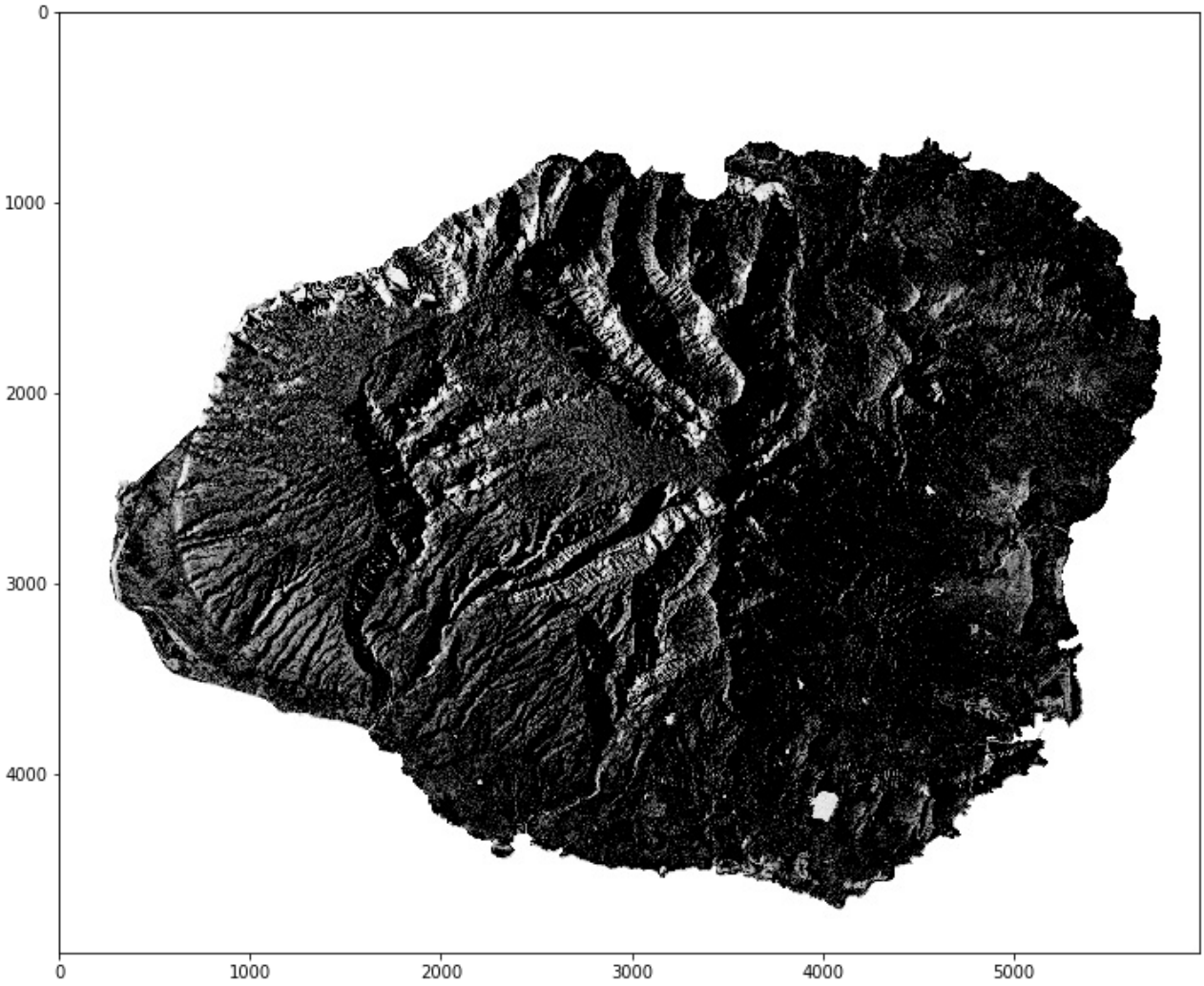


Applying terrain correction

Finally, we must apply terrain correction to our product.

```
parameters = HashMap()
parameters.put('demName', 'SRTM 3Sec')
parameters.put('pixelSpacingInMeter', 10.0)
parameters.put('sourceBands', 'Sigma0_VV')

speckle_filter_tc = GPF.createProduct("Terrain-Correction", parameters,
speckle_filter)
plotBand(speckle_filter_tc, 'Sigma0_VV', 0, 0.1)
```

```
(5976L, 4934L)



<matplotlib.image.AxesImage at 0x7fb7e02e6490>
```



Generating a binary flood mask

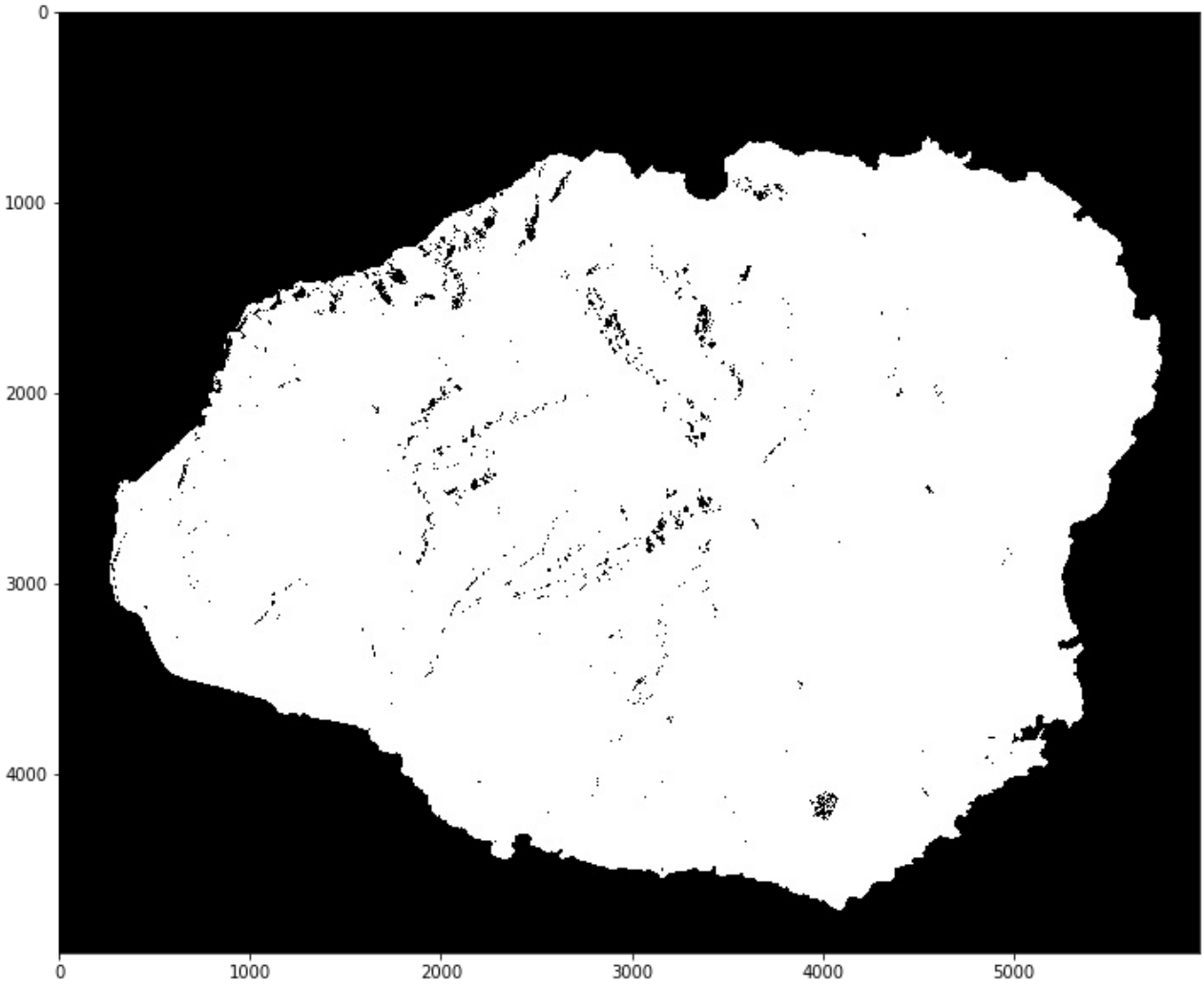
Since the current product we have processed contains backscatter coefficient values, pixels with a low backscatter will be flooded, while pixels with a high backscatter value will not be flooded.

For this particular area, a threshold of 0.013 was found to be sufficient to separate areas affected by the flooding from the rest of the image. We can apply this threshold to generate a binary flood mask using the `BandMaths` GPF module.

```
parameters = HashMap()

BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')

targetBand = BandDescriptor()
targetBand.name = 'Sigma0_VV_Flooded'
targetBand.type = 'uint8'
targetBand.expression = '(Sigma0_VV < 1.13E-2) ? 1 : 0'
targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
targetBands[0] = targetBand
parameters.put('targetBands', targetBands)

flood_mask = GPF.createProduct('BandMaths', parameters, speckle_filter_tc)
```

```
plotBand(flood_mask, 'Sigma0_VV_Flooded', 0, 1)
```

```
(5976L, 4934L)



<matplotlib.image.AxesImage at 0x7fb7da1c5a10>
```



Masking out known water areas

Currently, our backscatter coefficient classification has misidentified inland bodies of water as flooded areas. These can easily be masked out by importing a land cover layer, converting it to a binary mask, and removing any values that are already known as water, such as the lake in the lower left hand corner of the island.

To begin, we'll first use the `AddLandCover` GPF module to automatically download the [GlobCover (http://due.esrin.esa.int/page_globcover.php)](http://due.esrin.esa.int/page_globcover.php) land use dataset for our study area.

```
parameters = HashMap()
parameters.put("landCoverNames", "GlobCover")
mask_with_land_cover = GPF.createProduct('AddLandCover', parameters,
flood_mask)
```

Our GlobCover layer is added as a band with the name `land_cover_GlobCover`.

The classification scheme for GlobCover is as follows:

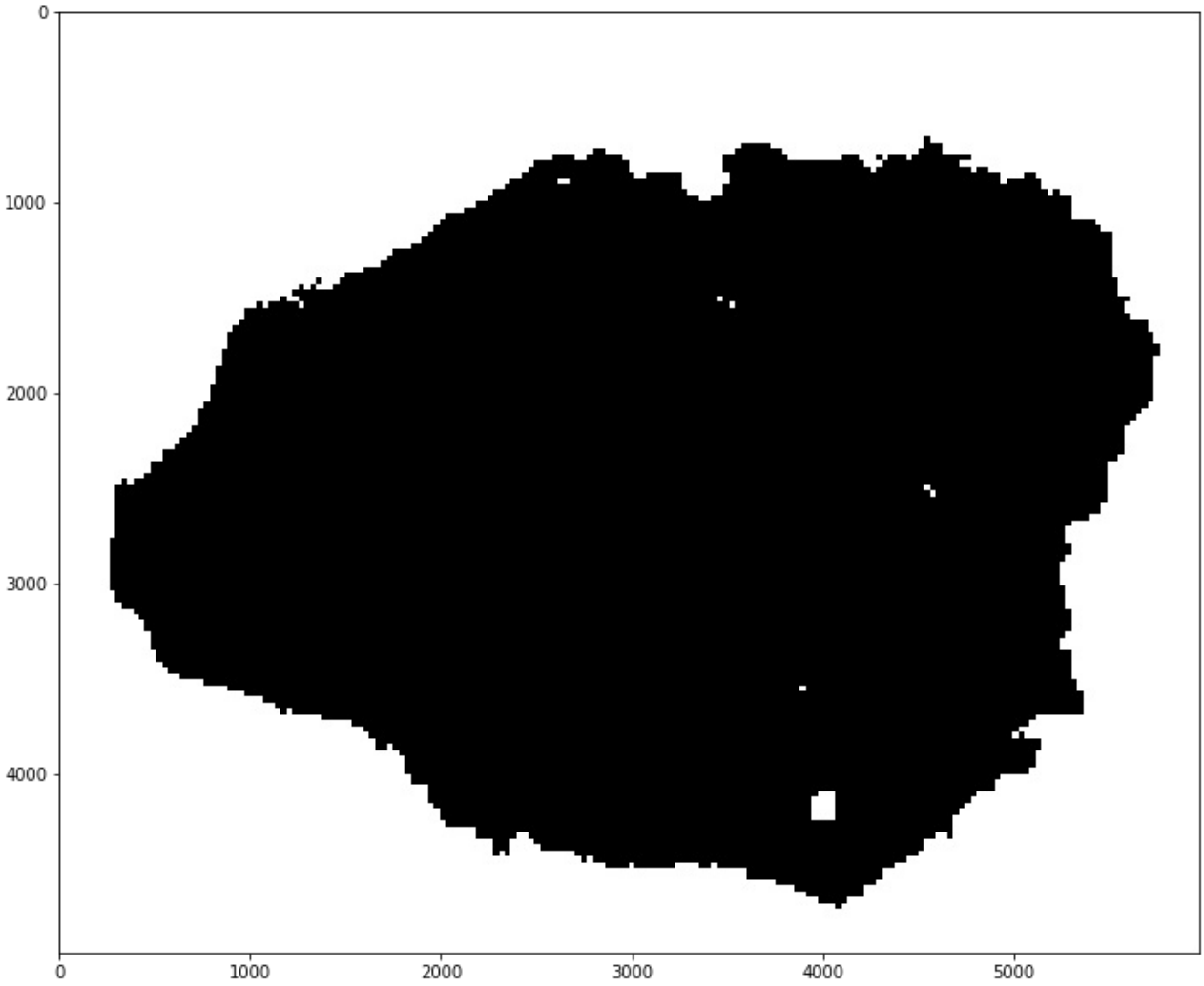
Label	..	Value
11 Post-flooding or irrigated croplands (or aquatic)		11
14 Rainfed croplands		14
20 Mosaic cropland (50-70%) / vegetation (grassland/shrubland/forest) (20-50%)		20
30 Mosaic vegetation (grassland/shrubland/forest) (50-70%) / cropland (20-50%)		30
40 Closed to open (>15%) broadleaved evergreen or semi-deciduous forest (>5m)		40
50 Closed (>40%) broadleaved deciduous forest (>5m)		50
60 Open (15-40%) broadleaved deciduous forest/woodland (>5m)		60
70 Closed (>40%) needleleaved evergreen forest (>5m)		70
90 Open (15-40%) needleleaved deciduous or evergreen forest (>5m)		90
100 Closed to open (>15%) mixed broadleaved and needleleaved forest (>5m)		100
110 Mosaic forest or shrubland (50-70%) / grassland (20-50%)		110
120 Mosaic grassland (50-70%) / forest or shrubland (20-50%)		120
130 Closed to open (>15%) (broadleaved or needleleaved, evergreen or deciduous) shrubland (<5m)		130
140 Closed to open (>15%) herbaceous vegetation (grassland, savannas or lichens/mosses)		140
150 Sparse (<15%) vegetation		150
160 Closed to open (>15%) broadleaved forest regularly flooded (semi-permanently or temporarily) - Fresh or brackish water		160
170 Closed (>40%) broadleaved forest or shrubland permanently flooded - Saline or brackish water		170
180 Closed to open (>15%) grassland or woody vegetation on regularly flooded or waterlogged soil - Fresh, brackish or saline water		180
190 Artificial surfaces and associated areas (Urban areas >50%)		190
200 Bare areas		200
210 Water bodies		210
220 Permanent snow and ice		220
230 No data		230

The value we want to include in our known water mask is 210. We can create our binary water mask by using code similar to what we used earlier in this tutorial to generate the flood mask.

```
parameters = HashMap()

BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')

targetBand = BandDescriptor()
targetBand.name = 'BinaryWater'
targetBand.type = 'uint8'
targetBand.expression = '(land_cover_GlobCover == 210) ? 0 : 1'
targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
targetBands[0] = targetBand
parameters.put('targetBands', targetBands)

water_mask = GPF.createProduct('BandMaths', parameters, mask_with_land_cover)
```

```
plotBand(water_mask, 'BinaryWater', 0, 1)

print(",".join(water_mask.getBandNames()))
print(",".join(flood_mask.getBandNames()))
```

```
(5976L, 4934L)
BinaryWater
Sigma0_VV_Flooded
```

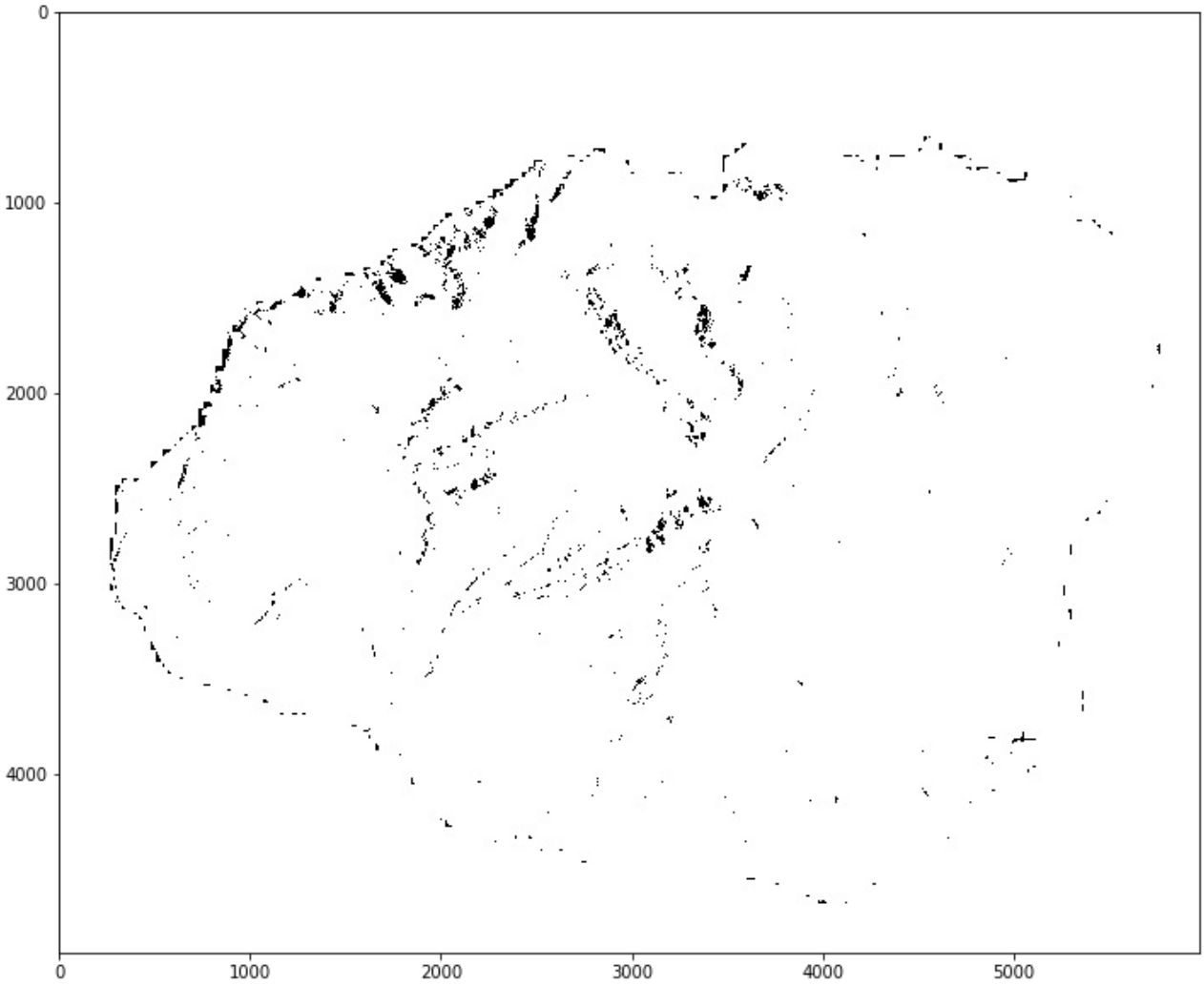


With our binary water mask created, it is time to mask out the lake and other small known water areas using it.

```
parameters = HashMap()

BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')
try:
    water_mask.addBand(flood_mask.getBand("Sigma0_VV_Flooded"))
except:
    pass

targetBand = BandDescriptor()
targetBand.name = 'Sigma0_VV_Flood_Masked'
targetBand.type = 'uint8'
targetBand.expression = '(BinaryWater == 1 && Sigma0_VV_Flooded == 1) ? 1 : 0'
targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
targetBands[0] = targetBand
parameters.put('targetBands', targetBands)

water_mask2 = GPF.createProduct('BandMaths', parameters, water_mask)
```

```
plotBand(water_mask2, 'Sigma0_VV_Flood_Masked', 0, 1)
```

```
(5976L, 4934L)



<matplotlib.image.AxesImage at 0x7fb7d9f444d0>
```



We can now see that the lake that was previously in our flood mask is now removed.

Saving the product

You can save the mask product (saves to `data/final_mask.tif` and then open it in SNAP or any other GIS/remote sensing platform with the following command:

```
ProductIO.writeProduct(water_mask2, "data/final_mask",  'GeoTIFF')
os.path.exists("data/final_mask.tif")
```

```
True
```

Now you have successfully done some basic SAR preprocessing and processing using snappy!

Additional information on using Snappy

To get information on what parameters any operator needs, you can use the following method

```
def listParams(operator_name):
    GPF.getDefaultInstance().getOperatorSpiRegistry().loadOperatorSpis()
    op_spi =
GPF.getDefaultInstance().getOperatorSpiRegistry().getOperatorSpi(operator_name)
    print('Op name:', op_spi.getOperatorDescriptor().getName())
    print('Op alias:', op_spi.getOperatorDescriptor().getAlias())
    param_Desc = op_spi.getOperatorDescriptor().getParameterDescriptors()
    for param in param_Desc:
        print(param.getName(), "or", param.getAlias())

listParams("AddLandCover")
```

```
('Op name:', 'org.esa.snap.landcover.gpf.AddLandCoverOp')
('Op alias:', 'AddLandCover')
('landCoverNames', 'or', None)
('externalFiles', 'or', None)
('resamplingMethod', 'or', None)
```

As of SNAP 6.0, the following are supported:

```
- Aatsr.SST                              Computes sea surface temperature (SST)
from (A)ATSR products.
- AATSR.Ungrid                          Ungrids (A)ATSR L1B products and extracts
geolocation and pixel field of view data.
-  AdaptiveThresholding                 Detect ships using Constant False Alarm
Rate detector.
-  AddElevation                         Creates a DEM band
-  AddLandCover                         Creates a land cover band
-  AerosolRetrieval.S2.Master           Aerosol retrieval from S2 MSI following
USwansea algorithm as used in GlobAlbedo project.
-  ALOS-Deskewing                       Deskewing ALOS product
-  Apply-Orbit-File                     Apply orbit file
-  Arc.SST                              Computes sea surface temperature (SST)
from (A)ATSR products.
-  ArviOp                               Atmospherically Resistant Vegetation
Index belongs to a family of indices with built-in atmospheric corrections.
-  AzimuthFilter                        Azimuth Filter
-  Back-Geocoding                       Bursts co-registration using orbit and
DEM
-  BandMaths                            Create a product with one or more bands
using mathematical expressions.
-  BandMerge                            Allows copying raster data from any
number of source products to a specified 'master' product.
-  BandSelect                           Creates a new product with only
selected bands
-  Bi2Op                                The Brightness index represents the
average of the brightness of a satellite image.
                                       This index is sensitive to the
brightness of soils which is highly correlated with the humidity and the
presence of salts in surface
-  Binning                              Performs spatial and temporal
```

```
aggregation of pixel values into cells ('bins') of a planetary grid
-  BiOp                                The Brightness index represents the
average of the brightness of a satellite image.
-  BiophysicalOp                       The 'Biophysical Processor' operator
retrieves LAI from atmospherically corrected Sentinel-2 products
-  c2rcc.landsat8                      Performs atmospheric correction and IOP
retrieval with uncertainties on Landsat-8 L1 data products.
-  c2rcc.meris                         Performs atmospheric correction and IOP
retrieval with uncertainties on MERIS L1b data products.
-  c2rcc.meris4                        Performs atmospheric correction and IOP
retrieval with uncertainties on MERIS L1b data products from the 4th
reprocessing.
-  c2rcc.modis                         Performs atmospheric correction and IOP
retrieval on MODIS L1C_LAC data products.
-  c2rcc.msi                           Performs atmospheric correction and IOP
retrieval with uncertainties on Sentinel-2 MSI L1C data products.
-  c2rcc.olci                          Performs atmospheric correction and IOP
retrieval with uncertainties on SENTINEL-3 OLCI L1B data products.
-  c2rcc.seawifs                       Performs atmospheric correction and IOP
retrieval on SeaWifs L1C data products.
-  c2rcc.viirs                         Performs atmospheric correction and IOP
retrieval on Viirs L1C data products.
-  Calibration                         Calibration of products
-  CCICloudShadow                      Algorithm detecting cloud shadow...
-  Change-Detection                    Change Detection.
-  CiOp                                Colour Index  was developed to
differentiate soils in the field.
                                      In most cases the CI gives complementary
information with the BI and the NDVI.
                                       Used for diachronic analyses, they help
for a better understanding of the evolution of soil surfaces.
-  CloudProb                           Applies a clear sky conservative cloud
detection algorithm.
-  Coherence                           Estimate coherence from stack of
coregistered images
-  Collocate                           Collocates two products based on their
geo-codings.
-  Convert-Datatype                    Convert product data type
-  CreateStack                         Collocates two or more products based
on their geo-codings.
-  Cross-Channel-SNR-Correction        Compute general polarimetric parameters
-  Cross-Correlation                   Automatic Selection of Ground Control
Points
-  CrossResampling                     Estimate Resampling Polynomial using
SAR Image Geometry, and Resample Input Images
-  DeburstWSS                          Debursts an ASAR WSS product
-  DEM-Assisted-Coregistration         Orbit and DEM based co-registration
-  Double-Difference-Interferogram     Compute double difference interferogram
-  DviOp                               Difference Vegetation Index retrieves
the Isovegetation lines parallel to soil line
-  EAP-Phase-Correction                EAP Phase Correction
-  Ellipsoid-Correction-GG             GG method for orthorectification
```

```
-  Ellipsoid-Correction-RD             Ellipsoid correction with RD method and
average scene height
-  EMClusterAnalysis                   Performs an expectation-maximization
(EM) cluster analysis.
-  Enhanced-Spectral-Diversity         Estimate constant range and azimuth
offsets for the whole image
-  Fill-DEM-Hole                       Fill holes in given DEM product file.
-  Find-Image-Pair                     DB query to find matching image pair
-  FlhMci                              Computes fluorescence line height (FLH)
or maximum chlorophyll index (MCI).
-  Flip                                flips a product horizontal/vertical
-  FUB.Water                           FUB/WeW WATER Processor to retrieve
case II water properties and atmospheric properties
-  FuClassification                    Colour classification based on the
discrete Forel-Ule scale.
-  GemiOp                              This retrieves the Global Environmental
Monitoring Index (GEMI).
-  GenericRegionMergingOp              The 'Generic Region Merging' operator
computes the distinct regions from a product
-  GLCM                                Extract Texture Features
-  GndviOp                             Green Normalized Difference Vegetation
Index
-  GoldsteinPhaseFiltering             Phase Filtering
-  GRD-Post                            Applies GRD post-processing
-  Idepix.Envisat.Meris                Pixel identification and classification
for MERIS.
-  Idepix.Landsat8.OLI                 Pixel identification and classification
for Landsat-8 OLI instrument.
-  Idepix.OrbView2.Seawifs             Pixel identification and classification
for SeaWiFS.
-  Idepix.Probav.Vegetation            Pixel identification and classification
for PROBA-V.
-  Idepix.Sentinel2                    Pixel identification and classification
for Sentinel-2.
-  Idepix.Sentinel3.Olci               Pixel identification and classification
for OLCI.
-  Idepix.SuomiNpp.Viirs               Pixel identification and classification
for VIIRS.
-  Idepix.TerraAqua.Modis              Pixel identification and classification
for MODIS.
-  Image-Filter                        Common Image Processing Filters
-  Import-Vector                       Imports a shape file into a product
-  IntegerInterferogram                Create integer interferogram
-  Interferogram                       Compute interferograms from stack of
coregistered S-1 images
-  IpviOp                              Infrared Percentage Vegetation Index
retrieves the Isovegetation lines converge at origin
-  IreciOp                             Inverted red-edge chlorophyll index
- KDTree-KNN-Classifier                KDTree KNN classifier
-  KMeansClusterAnalysis               Performs a K-Means cluster analysis.
-  KNN-Classifier                      K-Nearest Neighbour classifier
-  Land-Cover-Mask                     Perform decision tree classification
```

```
-  Land-Sea-Mask                        Creates a bitmask defining land vs
ocean.
-  LandWaterMask                        Operator creating a target product with
a single band containing a land/water-mask.
-  LinearToFromdB                       Converts bands to/from dB
-  Maximum-Likelihood-Classifier        Maximum Likelihood classifier
-  McariOp                              Modified Chlorophyll Absorption Ratio
Index, developed to be responsive to chlorophyll variation
-  Mci.s2                               Computes maximum chlorophyll index
(MCI) for Sentinel-2 MSI.
-  Merge                                Allows merging of several source
products by using specified 'master' as reference product.
-  Meris.CorrectRadiometry              Performs radiometric corrections on
MERIS L1b data products.
-  Meris.N1Patcher                      Copies an existing N1 file and replaces
the data for the radiance bands
-  Minimum-Distance-Classifier          Minimum Distance classifier
-  MndwiOp                              Modified Normalized Difference Water
Index, allowing for the measurement of surface water extent
-  Mosaic                               Creates a mosaic out of a set of source
products.
-  MphChl                               This operator computes maximum peak
height of chlorophyll (MPH/CHL).
-  Msavi2Op                             This retrieves the second Modified Soil
Adjusted Vegetation Index (MSAVI2).
-  MsaviOp                              This retrieves the Modified Soil
Adjusted Vegetation Index (MSAVI).
-  MtciOp                               The Meris Terrestrial Chlorophyll
Index,  aims at estimating the Red Edge Position (REP).
                                       This is the maximum slant point in the
red and near-infrared region of the vegetal spectral reflectance.
                                       It is useful for observing the
chlorophyll contents, vegetation senescence, and stress for water and
nutritional deficiencies, but it is less suitable for land classification
-  Multi-size Mosaic                    Creates a multi-size mosaic out of a
set of source products.
-  Multi-Temporal-Speckle-Filter        Speckle Reduction using Multitemporal
Filtering
-  Multilook                            Averages the power across a number of
lines in both the azimuth and range directions
-  Ndi45Op                              Normalized Difference Index using bands
4 and 5
-  NdpiOp                               The normalized differential pond index,
combines the short-wave infrared band-I and the green band
-  NdtiOp                               Normalized difference turbidity index,
allowing for the measurement of water turbidity
-  NdviOp                               The retrieves the Normalized Difference
Vegetation Index (NDVI).
-  Ndwi2Op                              The Normalized Difference Water Index,
allowing for the measurement of surface water extent
-  NdwiOp                               The Normalized Difference Water Index
was developed for the extraction of water features
```

```
-  Object-Discrimination                Remove false alarms from the detected
objects.
-  Offset-Tracking                      Create velocity vectors from offset
tracking
-  Oil-Spill-Clustering                 Remove small clusters from detected
area.
-  Oil-Spill-Detection                  Detect oil spill.
-  Orientation-Angle-Correction         Perform polarization orientation angle
correction for given coherency matrix
-  Oversample                           Oversample the datset
-  OWTClassification                    Performs an optical water type
classification based on atmospherically corrected reflectances.
-  PCA                                  Performs a Principal Component
Analysis.
-  PduStitching                         Stitches multiple SLSTR L1B product
dissemination units (PDUs) of the same orbit to a single product.
-  PhaseFilter                          Interferometric phase filtering
-  PhaseToDisplacement                  Phase To Displacement Conversion along
LOS
-  PhaseToElevation                     DEM Generation
-  PhaseToHeight                        Phase to Height conversion
-  PixEx                                Extracts pixels from given locations
and source products.
-  Polarimetric-Classification          Perform Polarimetric classification of
a given product
-  Polarimetric-Decomposition           Perform Polarimetric decomposition of a
given product
-  Polarimetric-Matrices                Generates covariance or coherency
matrix for given product
-  Polarimetric-Parameters              Compute general polarimetric parameters
-  Polarimetric-Speckle-Filter          Polarimetric Speckle Reduction
-  Principle-Components                 Principle Component Analysis
-  ProductSet-Reader                    Adds a list of sources
-  PssraOp                              Pigment Specific Simple Ratio,
chlorophyll index
-  PviOp                                Perpendicular Vegetation Index
retrieves the Isovegetation lines parallel to soil line. Soil line has an
arbitrary slope and passes through origin
-  Rad2Refl                             Provides conversion from radiances to
reflectances or backwards.
-  Random-Forest-Classifier             Random Forest based classifier
-  RangeFilter                          Range Filter
-  RayleighCorrection                   Performs radiometric corrections on
OLCI, MERIS L1b and S2 MSI data products.
-  Read                                 Reads a data product from a given file
location.
-  ReflectanceToRadianceOp              The 'Reflectance To Radiance Processor'
operator retrieves the radiance from reflectance using Sentinel-2 products
-  ReipOp                               The red edge inflection point index
-  Remove-GRD-Border-Noise              Mask no-value pixels for GRD product
-  RemoveAntennaPattern                 Remove Antenna Pattern
-  ReplaceMetadata                      Replace the metadata of the first
```

```
product with that of the second
-  Reproject                            Reprojection of a source product to a
target Coordinate Reference System.
-  Resample                             Resampling of a multi-size source
product to a single-size target product.
-  RiOp                                 The Redness Index was developed to
identify soil colour variations.
-  RviOp                                Ratio Vegetation Index retrieves the
Isovegetation lines converge at origin
-  S2.WaterVapour                       Water Vapour retrieval from S2 MSI
-  S2repOp                              Sentinel-2 red-edge position index
-  S2tbx-Reproject                      Reprojection of a source product to a
target Coordinate Reference System.
-  SAR-Mosaic                           Mosaics two or more products based on
their geo-codings.
-  SAR-Simulation                       Rigorous SAR Simulation
-  SARSim-Terrain-Correction            Orthorectification with SAR simulation
-  SaviOp                               This retrieves the Soil-Adjusted
Vegetation Index (SAVI).
-  SetNoDataValue                       Set NoDataValueUsed flag and
NoDataValue for all bands
-  SliceAssembly                        Merges Sentinel-1 slice products
-  SmacOp                               Applies the Simplified Method for
Atmospheric Corrections of Envisat MERIS/(A)ATSR measurements.
-  SmosNetcdfExport                     Exports SMOS Earth Explorer products to
NetCDF format.
-  SnaphuExport                         Export data and prepare conf file for
SNAPHU processing
-  SnaphuImport                         Ingest SNAPHU results into InSAR
product.
-  Speckle-Divergence                   Detect urban area.
-  Speckle-Filter                       Speckle Reduction
-  SRGR                                 Converts Slant Range to Ground Range
-  Stack-Averaging                      Averaging multi-temporal images
-  Stack-Split                          Writes all bands to files.
-  StampsExport                         Export data for StaMPS processing
-  StatisticsOp                         Computes statistics for an arbitrary
number of source products.
-  SubGraph                             Encapsulates a graph within a graph.
-  Subset                               Create a spatial and/or spectral subset
of a data product.
-  Supervised-Wishart-Classification    Perform supervised Wishart
classification
-  TemporalPercentile                   Computes percentiles over a given time
period.
-  Terrain-Correction                   RD method for orthorectification
-  Terrain-Flattening                   Terrain Flattening
-  Terrain-Mask                         Terrain Mask Generation
-  ThermalNoiseRemoval                  Removes thermal noise from products
-  Three-passDInSAR                     Differential Interferometry
-  TileWriter                           Writes a data product to a tiles.
-  TndviOp                              Transformed Normalized Difference
```

```
Vegetation Index retrieves the Isovegetation lines parallel to soil line
-  ToolAdapterOp                          Tool Adapter Operator
-  TopoPhaseRemoval                       Compute and subtract TOPO phase
-  TOPSAR-Deburst                         Debursts a Sentinel-1 TOPSAR product
-  TOPSAR-Merge                           Merge subswaths of a Sentinel-1 TOPSAR
product
-  TOPSAR-Split                           Creates a new product with only the
selected subswath
-  TsaviOp                                This retrieves the Transformed Soil
Adjusted Vegetation Index (TSAVI).
-  Undersample                            Undersample the datset
-  Unmix                                  Performs a linear spectral unmixing.
-  Update-Geo-Reference                   Update Geo Reference
-  Warp                                   Create Warp Function And Get Co-
registrated Images
-  WdviOp                                 Weighted Difference Vegetation Index
retrieves the Isovegetation lines parallel to soil line. Soil line has an
arbitrary slope and passes through origin
-  Wind-Field-Estimation                  Estimate wind speed and direction
-  Write                                  Writes a data product to a file
```

```
#!/usr/bin/env python2
## Full script.
import os
'''
Only necessary if you do not have snappy in your site-packages folder.
import sys
snappy_directory = "/home/alex/.snap/snap-python"
sys.path.append(snappy_directory)
'''
import snappy
from snappy import Product
from snappy import ProductIO
from snappy import ProductUtils
from snappy import WKTReader
from snappy import HashMap
from snappy import GPF
from snappy import jpy

# For shapefiles
import shapefile
import pygeoif

## Helper functions

def showProductInformation(product):
    width = product.getSceneRasterWidth()
    print("Width: {} px".format(width))
    height = product.getSceneRasterHeight()
    print("Height: {} px".format(height))
    name = product.getName()
    print("Name: {}".format(name))
```

```
    band_names = product.getBandNames()
    print("Band names: {}".format(", ".join(band_names)))

def shpToWKT(shp_path):
    r = shapefile.Reader(shp_path)
    g = []
    for s in r.shapes():
        g.append(pygeoif.geometry.as_shape(s))
    m = pygeoif.MultiPoint(g)
    return str(m.wkt).replace("MULTIPOINT", "POLYGON(") + ")"


## Preprocessing functions

def applyOrbit(product):
    parameters = HashMap()
    parameters.put('orbitType', 'Sentinel Precise (Auto Download)')
    parameters.put('polyDegree', '3')
    parameters.put('continueOnFail', 'false')
    return GPF.createProduct('Apply-Orbit-File', parameters, product)

def subset(product, shpPath):
    parameters = HashMap()
    wkt = shpToWKT(shpPath)
    SubsetOp = jpy.get_type('org.esa.snap.core.gpf.common.SubsetOp')
    geometry = WKTReader().read(wkt)
    parameters.put('copyMetadata', True)
    parameters.put('geoRegion', geometry)
    return GPF.createProduct('Subset', parameters, product)

def calibration(product):
    parameters = HashMap()
    parameters.put('outputSigmaBand', True)
    parameters.put('sourceBands', 'Intensity_VV')
    parameters.put('selectedPolarisations', "VV")
    parameters.put('outputImageScaleInDb', False)
    return GPF.createProduct("Calibration", parameters, product)

def speckleFilter(product):
    parameters = HashMap()

    filterSizeY = '5'
    filterSizeX = '5'

    parameters.put('sourceBands', 'Sigma0_VV')
    parameters.put('filter', 'Lee')
    parameters.put('filterSizeX', filterSizeX)
    parameters.put('filterSizeY', filterSizeY)
    parameters.put('dampingFactor', '2')
    parameters.put('estimateENL', 'true')
    parameters.put('enl', '1.0')
    parameters.put('numLooksStr', '1')
```

```
    parameters.put('targetWindowSizeStr', '3x3')
    parameters.put('sigmaStr', '0.9')
    parameters.put('anSize', '50')
    return GPF.createProduct('Speckle-Filter', parameters, product)

def terrainCorrection(product):
    parameters = HashMap()
    parameters.put('demName', 'SRTM 3Sec')
    parameters.put('pixelSpacingInMeter', 10.0)
    parameters.put('sourceBands', 'Sigma0_VV')

    return GPF.createProduct("Terrain-Correction", parameters, product)

# Flooding processing

def generateBinaryFlood(product):
    parameters = HashMap()

    BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')

    targetBand = BandDescriptor()
    targetBand.name = 'flooded'
    targetBand.type = 'uint8'
    targetBand.expression = '(Sigma0_VV < 1.13E-2) ? 1 : 0'
    targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
    targetBands[0] = targetBand
    parameters.put('targetBands', targetBands)

    return GPF.createProduct('BandMaths', parameters, product)

def maskKnownWater(product):
    # Add land cover band
    parameters = HashMap()
    parameters.put("landCoverNames", "GlobCover")
    mask_with_land_cover = GPF.createProduct('AddLandCover', parameters,
product)
    del parameters

    # Create binary water band
    BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')
    parameters = HashMap()
    targetBand = BandDescriptor()
    targetBand.name = 'BinaryWater'
    targetBand.type = 'uint8'
    targetBand.expression = '(land_cover_GlobCover == 210) ? 0 : 1'
    targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
    targetBands[0] = targetBand
    parameters.put('targetBands', targetBands)
```

```
    water_mask = GPF.createProduct('BandMaths', parameters,
mask_with_land_cover)
    del parameters

    parameters = HashMap()

    BandDescriptor =
snappy.jpy.get_type('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor')
    try:
        water_mask.addBand(product.getBand("flooded"))
    except:
        pass

    targetBand = BandDescriptor()
    targetBand.name = 'Sigma0_VV_Flood_Masked'
    targetBand.type = 'uint8'
    targetBand.expression = '(BinaryWater == 1 && flooded == 1) ? 1 : 0'
    targetBands =
snappy.jpy.array('org.esa.snap.core.gpf.common.BandMathsOp$BandDescriptor', 1)
    targetBands[0] = targetBand
    parameters.put('targetBands', targetBands)

    return GPF.createProduct('BandMaths', parameters, water_mask)




if __name__ == "__main__":
    ## GPF Initialization
    GPF.getDefaultInstance().getOperatorSpiRegistry().loadOperatorSpis()

    ## Product initialization
    path_to_sentinel_data =
"data/S1A_IW_GRDH_1SDV_20180415T163146_20180415T163211_021480_025003_8E79.zip"
    product = ProductIO.readProduct(path_to_sentinel_data)
    showProductInformation(product)


    product_orbitfile = applyOrbit(product)
    product_subset = subset(product_orbitfile, "data/island_boundary2.shp")
    showProductInformation(product_subset)

    # Apply remainder of processing steps in a nested function call
    product_preprocessed = terrainCorrection(
                            speckleFilter(
                                calibration(
                                    product_subset
                                    )
                                )
                            )
```

```
product_binaryflood = maskKnownWater(
                        generateBinaryFlood(
                            product_preprocessed
                            )
                        )
ProductIO.writeProduct(product_binaryflood, "data/final_mask",  'GeoTIFF')
```