

APPENDIX C

Estimating Forest Stand Height Using L-band SAR – Chapter 4 Training Module

Developed by Helen Baldwin and Sarva Pulla with data and scripts from Paul Siqueira and Yang Lei

Input datasets:

- ALOS PALSAR or equivalent L-band dual-polarized imagery
- Forest height data (lidar or ground collection; GeoTIFF)
- A forest/non-forest mask (optional; GeoTIFF)

Software:

- A Unix/Linux environment required to run the Python scripts
- QGIS/ArcGIS/GoogleEarth (suggested for visualization)
- Anaconda for Python and packages

In this tutorial, we will estimate forest stand height (FSH) using L-band SAR data. The most accurate way to estimate FSH with repeat pass interferometry is by using a combination of SAR backscatter power and InSAR coherence. For this reason, the approach described here can be referred to as a combined SAR/InSAR estimation of FSH. Since the backscatter power relationship is most appropriate to calculate values of FSH below 10 m and values above this threshold are best determined by interferometric coherence, this algorithm computes FSH from interferometric coherence first, and the backscatter power empirical relationship is used if the FSH is below that threshold.

1 DATA ACQUISITION

One L-band SAR scene and one ancillary dataset are necessary for this tutorial. An additional ancillary dataset is recommended. To download an example dataset, please see section 2.5 of this module.

1.1 ALOS PALSAR

Since the structure of vegetation is on the order of 10's of centimeters, forest vegetation is often best ob-

served with P- or L-bands (~24 cm). At this bandwidth, the Japanese Aerospace Exploration Agency (JAXA)'s JERS-1 and ALOS-1 & -2 satellites are available, but geographically limited. This tutorial utilizes ALOS-1. Please refer to Marc Simard's Training Module in Appendix E for an explanation of how to acquire ALOS PALSAR data and select the Single-Look Complex (SLC) product. This tutorial could also potentially apply to NISAR data in the future.

Processing InSAR data to estimate FSH requires either raw satellite data that have been downlinked, but not processed, or SAR data that have been processed into SLC imagery appropriate for forming interferograms. If you have access to SLCs, it is recommended that you skip section 4 and proceed to section 5. If only raw data are available, then the additional processing explained in section 4 of this tutorial is necessary. One advantage of beginning with raw data are that the output formats of the interferograms and ancillary data make it easy to follow on the processing methods with additional steps implemented to estimate FSH.

1.2 Ancillary Datasets

FSH ground validation data are an important component of the FSH estimation methodology. There are two types of ancillary data utilized in the algorithm. Locations where forest height has been previously determined are required to train the empirical models. A forest/non-forest mask indicating where the estimates should be calculated is an optional dataset.

1.2.1 Forest Height Data

Independent measurements of forest height are necessary to determine values for the empirical models that relate radar backscatter power and interferometric coherence to FSH. Lidar data are preferred, since they acquire accurate measurements of vegetation height over an extended geographic region. Freely-available satellite resources of lidar data are currently or about to become accessible, including ICESAT-1 and -2, and the upcoming NASA GEDI mission. This dataset should be in a GeoTIFF format and resampled to the same resolution as the InSAR image. The margin/NoData values must be set to NaN or some number less than zero. Within the FSH scripts, this data set is referred to as "ref_file."

If lidar data are not available, then another form of independent forest height needs to be identified or created. A simple method is to perform a land cover classification of a region using optical data sets. Stands of different ages and species composition will have different heights, which can be estimated from the ground to the same accuracy as the FSH. This approach was used during the development and testing of the FSH algorithm with mixed results.

1.2.2 Map of Forest/Non-forest

The forest/non-forest map can be derived from a number of sources, or made independently by the user. Examples of sources that can be used to derive a forest/non-forest mask are i. JAXA's FNF mask, ii. the US National LandCover Dataset, and iii. The ESA's CCI Landcover (formerly GlobCover). These datasets are used to identify where forests are situated and, hence, where to estimate FSH. The forest/non-forest mask must be classified so that all regions where FSH should be estimated have a value of zero and all regions where FSH should not be estimated have a value of 1. This optional dataset should be a GeoTIFF and resampled to the same resolution as the InSAR image. This file must be in degrees; e.g., EPSG 4326. The margin/NoData values must be set to NaN or some number less than zero. Within the FSH scripts, this dataset is referred to as "mask_file".

2 LINUX ENVIRONMENT AND PYTHON SETUP

While Python scripts can be run in the Windows, OSX, and Unix environments, the methods in this module require a Unix or Linux environment. Please follow the instructions in section 2.1 to setup a Linux environment on your computer using Oracle Virtual-Box, section 2.2 to install Anaconda, and section 2.3 to install dependencies for the FSH scripts. If you already have a Linux environment, or have completed any of the other setup steps, please proceed to the next applicable section.

2.1 Download and Install VirtualBox

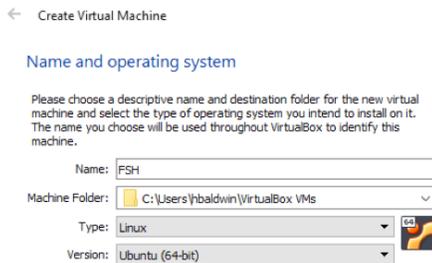
1. First, go to <https://www.virtualbox.org/> to download Oracle VM VirtualBox. Choose the host appropriate for your computer.
2. Next, go to <https://www.ubuntu.com/down->

load/desktop and download the latest version of Ubuntu. We will use this later on while setting up our virtual machine.

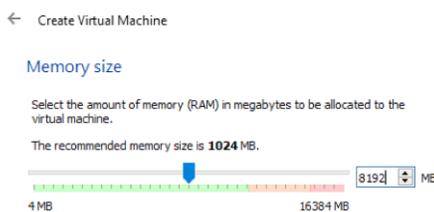
- Follow along with the Oracle VM VirtualBox installation wizard. Once installation is finished, open the Oracle VM VirtualBox.
- Click "New" in the menu located at the top of the Oracle VM VirtualBox Manager window to create the virtual machine you will use for this exercise. This menu bar is shown below.



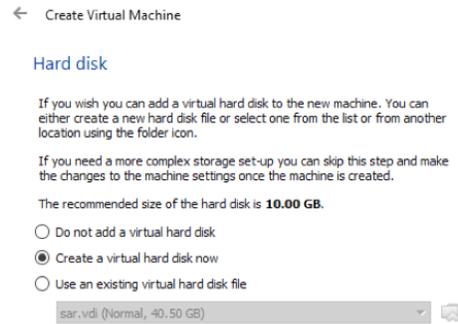
- The "Create Virtual Machine: Name and operating system" window shown below should pop up. Enter a name for your virtual machine. For this exercise, we chose "FSH". Browse to a folder where you would like to save your machine, select "Linux" from the dropdown menu as the type of machine, and select "Ubuntu (64-bit)" as your version.



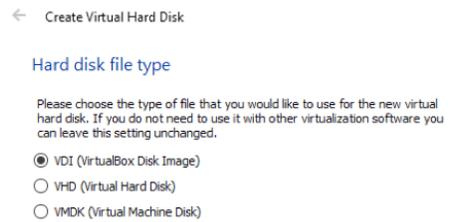
- Once the name and operating system for your new machine are set up as shown in the image above, click next. The "Create Virtual Machine: Memory size" window should pop up.
- Enter the amount of memory you would like to allocate to your machine. I chose 8192 MB, as shown below. Click next.



- Leave the Hard disk selection on "Create a virtual hard disk now" and click create, as shown below.



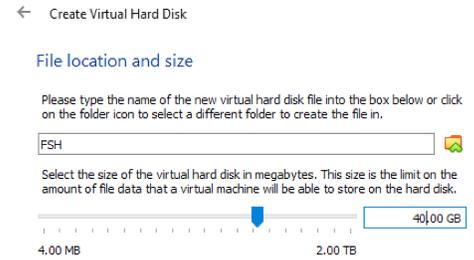
- Leave the Hard disk file type selection on "VDI (VirtualBox Disk Image)" and click next, as shown below.



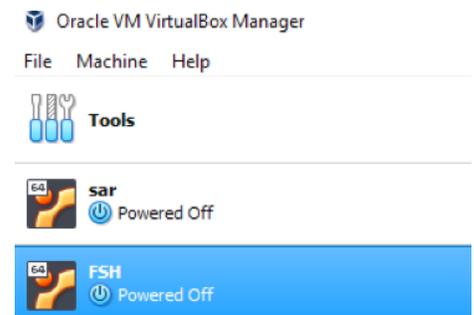
- Leave the Storage on physical hard disk selection on "Dynamically allocated" and select next, as shown below.



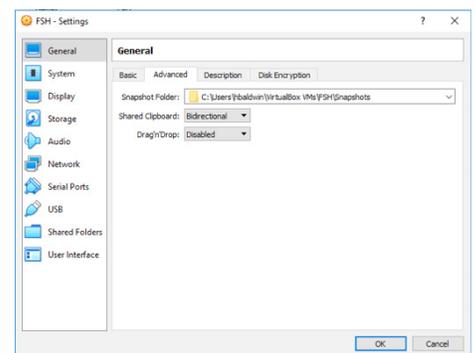
- Set up the file location and size as shown below. Your file name should automatically populate, but you can also navigate to a new folder to create the file if necessary. I selected 40GB for the virtual hard disk size. Select create, and the "Create Virtual Hard Disk" pop up window will close.



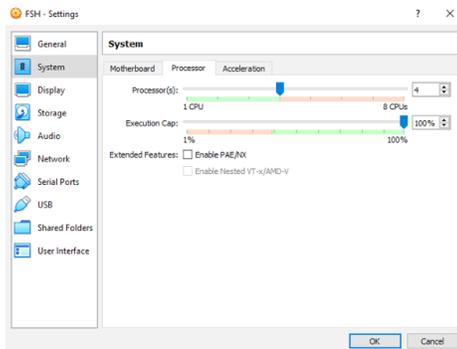
- Notice that your new virtual machine has been added to the list of virtual machines along the left side of your Oracle VM VirtualBox Manager. As shown below, I have a virtual machine named "sar" along with the virtual machine "FSH" that I just created.



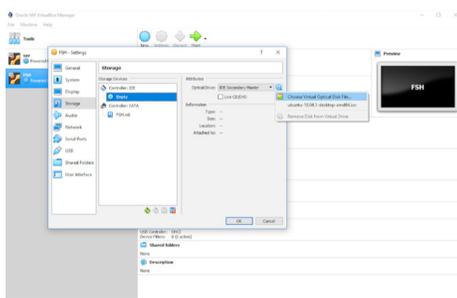
- Select your new virtual machine from the list. It should appear highlighted, as shown above.
- Click "Settings" in the menu located at the top of the Oracle VM VirtualBox Manager window to adjust the settings of your new virtual machine.
- Within the settings pop up window, navigate to the advanced tab.
- Under "Shared Clipboard," choose "Bidirectional" from the drop down menu. This will allow you to copy and paste between your host system and your virtual machine.



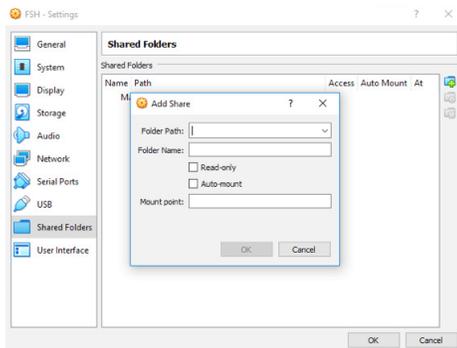
17. Navigate to "System" from the left hand menu. Choose the processor tab. Increase your number of CPUs; I chose 4, as that was the maximum within the suggested green range.



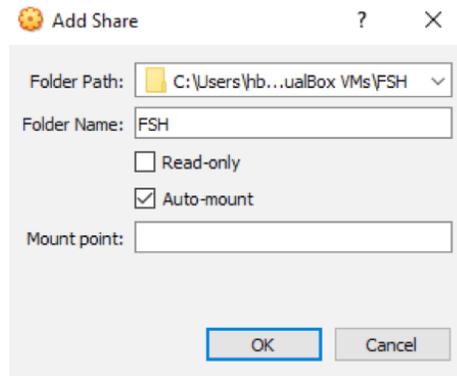
18. Navigate to "Storage" from the left hand menu.
 19. Select the "Empty" disk icon under the Controller IDE option. Under Attributes, click on the disk icon next to the optical drive selection "IDE Secondary Master." Navigate to the Ubuntu for desktop that you downloaded in step 2 using the "Choose Virtual Optical Disk File" option.



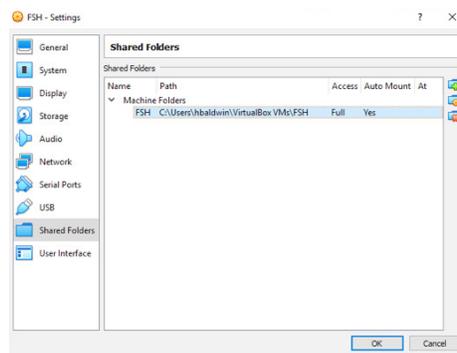
20. Navigate to "Shared Folders" from the left hand menu.
 21. Click the add folder icon  along the right of the shared folders window to get to the "Add share" pop up window as shown below.



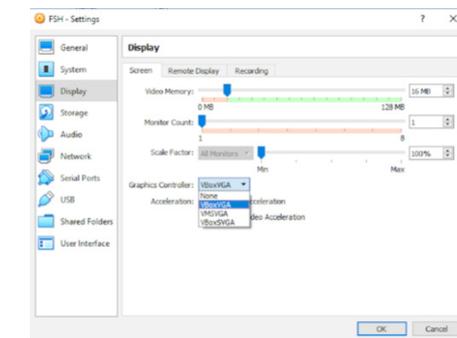
22. Navigate to the folder where your virtual machine is stored within the Folder Path option. The name of the folder will be automatically populated. Choose the "Auto-mount" option as shown below.



23. Click OK to return to the Shared Folders page. Your folder should now appear in the list of Machine Folders as shown below.

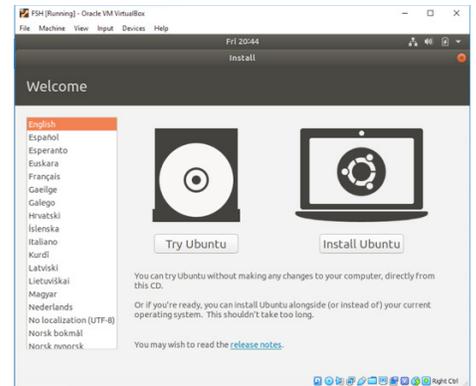


24. To avoid a blank screen after installing Guest Additions in a later step, navigate to "Display" from the left hand menu. Use the drop down menu for the Graphics Controller to select "VBoxVGA."



25. Click "OK" to apply these setting changes and return to the Oracle VM VirtualBox Manager. Click "Start" in the menu located at the top of the Oracle VM VirtualBox Manager window to run your new virtual machine.

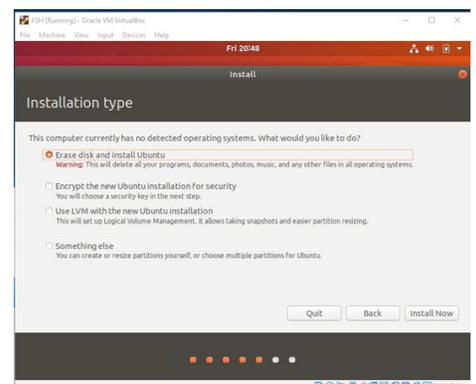
26. The welcome pop up shown below should appear. Choose your preferred language from the list and click "Install Ubuntu."



27. Click continue to utilize the default keyboard layout.

28. Click continue to utilize the default installation and update options.

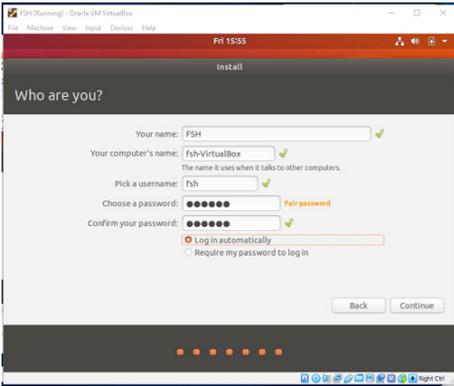
29. Click "Install Now" with the default selections as shown below.



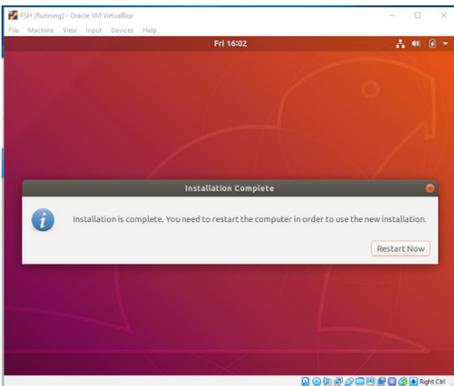
30. Click continue when the pop up window "Write the changes to disk?" appears.

31. Click continue after selecting your time zone.

32. Fill in your preferred name and password for your virtual box as shown below.



33. Once installation is complete, the window below should appear. Choose "Restart now" to use the new installation.



34. After a few minutes, the "What's new in Ubuntu" window shown below should appear. Click next.



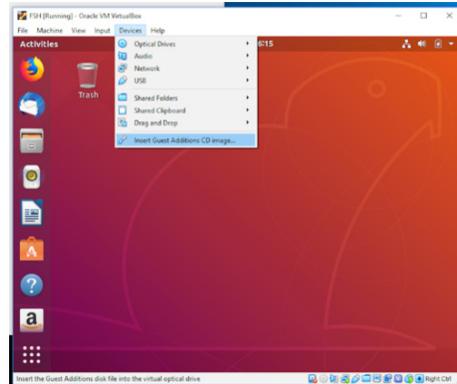
35. Click next to proceed past the Livepatch window.

36. Click next to proceed past the "Help improve

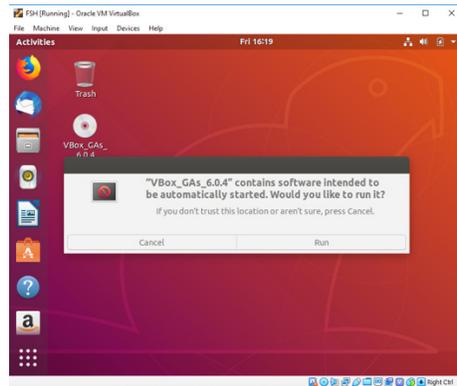
Ubuntu" window after choosing whether or not to report information to developers for improvement.

37. Click "Done" on the "Ready to go" window.

38. Click "Devices" in the menu on the top of your running machine and choose "Insert Guest Additions CD Image" from the drop down menu, as shown below.

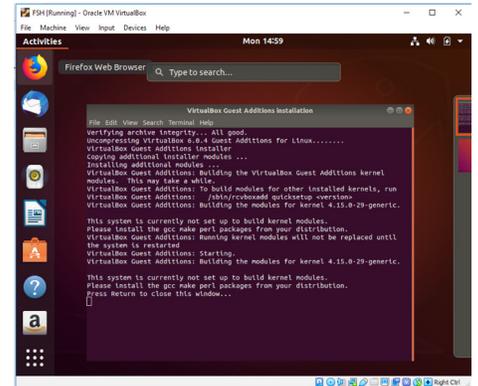


39. The VirtualBox Guest Additions CD (here: VBox_GAs_6.0.1) should appear on the desktop of your virtual machine and a warning window may appear as shown below. Click "run" to proceed. You may be prompted to enter in your password to run the Guest Additions disk.



40. Once the Guest Additions disk has finished running, the warning, "This system is currently not set up to build kernel modules" may appear at the end of the messages in the terminal, as shown below. If this is the case, press enter to close the window, and follow steps 40 through 48. If this warning does not appear, you may

move on to installing Anaconda in section 2.2.



41. Open the terminal using `ctr, alt, and t`. Then type in the command `sudo apt-get install linux-headers-`uname -r` dkms build-essential` or `sudo apt-get install linux-headers-$(uname -r) dkms build-essential`

42. You should be prompted to enter "y" to continue. The packages identified as missing in step 39 should now be installed. Press enter to close the window.

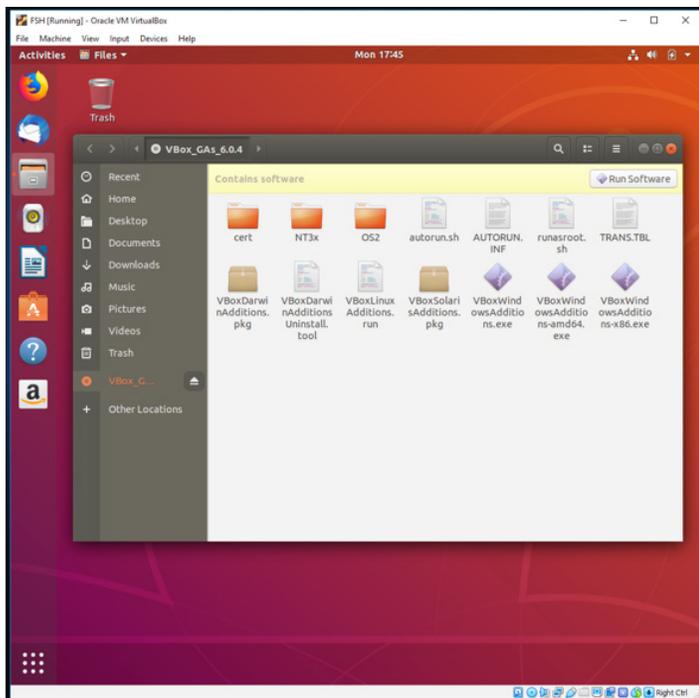
43. In order to use these packages, you will have to restart the virtual machine. Select the arrow along the top right menu (shown below).



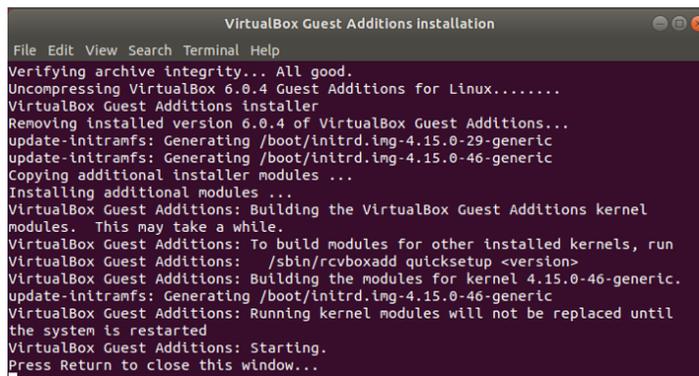
44. An additional menu, shown below, should open.



- Click on the power icon to open the Power Off window, and choose “Restart”. When the VM restarts, rerun the VirtualBox Guest Additions CD by clicking on the file icon in the menu on the left hand side. Click the Guest Additions disk in the left hand menu on the pop up window. Then select “Run Software”, as shown below.



- The VirtualBox Guest Additions Installation window should open as shown below. Press enter to close the window.

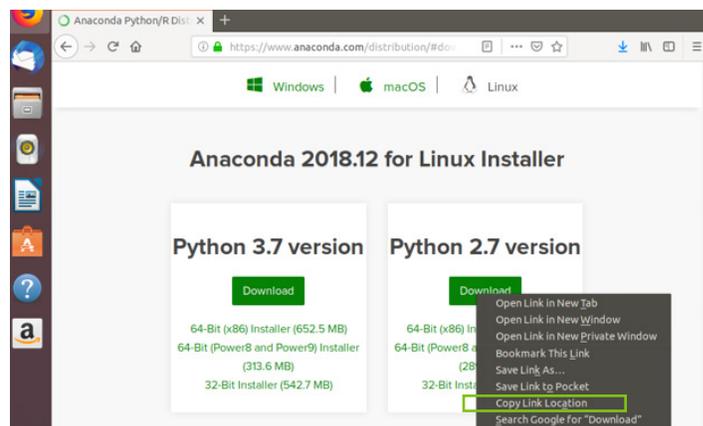


- Restart the machine as described in steps 42 through 45.

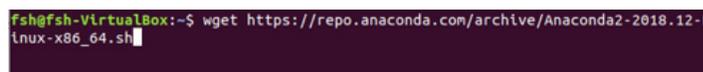
2.2 Download and Install Anaconda

- Open your web browser on your virtual machine, and navigate to <https://www.anaconda.com/distribution/#linux> or search for “install Anaconda.” Make sure to select the tab for the Linux operating system.
- Right click on the download button for the 2.7 version of Python as shown below, as the FSH scripts were developed and tested using this version. From the menu

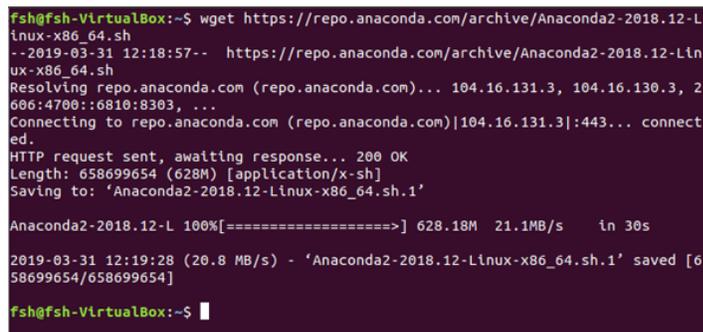
that appears, choose “Copy Link Location.”



- Open the terminal using ctr, alt, and t. Type in the the command “wget”, and then paste the location of the download for Python 2.7 version, as shown below.

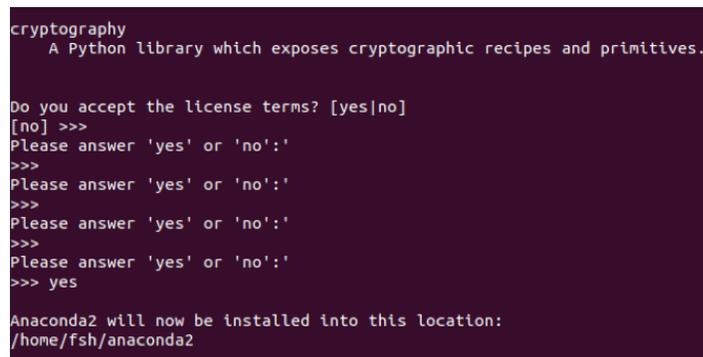


- The “Welcome to Anaconda” text should display in your terminal as shown below. Copy the highlighted “Anaconda2-2018.12-Linux-x86_64.sh.1” text.



- Enter the command “bash” and paste in the “Anaconda2-2018.12-Linux-x86_64.sh.1” text. Follow the prompts to review the Anaconda license information, and enter “yes” to confirm the installation of Anaconda when prompted.

- Enter the location where you would like Anaconda to be saved. I chose the



default as shown on the previous page.

54. After the installation is finished, you will be prompted to initialize Anaconda2 in your `.bashrc`, as shown below. Enter “yes.”

```
installing: blaze-0.11.3-py27_0 ...
installing: jupyter_console-5.2.0-py27_1 ...
installing: notebook-5.7.4-py27_0 ...
installing: qtconsole-4.4.3-py27_0 ...
installing: seaborn-0.9.0-py27_0 ...
installing: sphinx-1.8.2-py27_0 ...
installing: spyder-kernels-0.3.0-py27_0 ...
installing: anaconda-navigator-1.9.6-py27_0 ...
installing: anaconda-project-0.8.2-py27_0 ...
installing: jupyterlab_launcher-0.11.2-py27h28b3542_0 ...
installing: numpydoc-0.8.0-py27_0 ...
installing: widgetsnbextension-3.4.2-py27_0 ...
installing: ipywidgets-7.4.2-py27_0 ...
installing: jupyterlab-0.33.11-py27_0 ...
installing: spyder-3.3.2-py27_0 ...
installing: _ipyw_jlab_nb_ext_conf-0.1.0-py27_0 ...
installing: jupyter-1.0.0-py27_7 ...
installing: anaconda-2018.12-py27_0 ...
installing: conda-4.5.12-py27_0 ...
installing: conda-build-3.17.6-py27_0 ...
installation finished.
Do you wish the installer to initialize Anaconda2
in your /home/fsh/.bashrc ? [yes|no]
[no] >>> █
```

55. When prompted to proceed with the installation of Microsoft VSCode, as shown below, please enter “no.”

```
[no] >>> yes

Initializing Anaconda2 in /home/fsh/.bashrc
A backup will be made to: /home/fsh/.bashrc-anaconda2.bak

For this change to become active, you have to open a new terminal.

Thank you for installing Anaconda2!

=====

Anaconda is partnered with Microsoft! Microsoft VSCode is a streamlined
code editor with support for development operations like debugging, task
running and version control.

To install Visual Studio Code, you will need:
- Administrator Privileges
- Internet connectivity

Visual Studio Code License: https://code.visualstudio.com/license

Do you wish to proceed with the installation of Microsoft VSCode? [yes|no]
>>> █
```

56. Close your terminal and open a new terminal (ctr, alt, and t) for your installation of Anaconda to become active.

2.3 Download and Import Dependencies

1. To create a python environment named “sar” where we will store all the dependencies necessary to run the FSH scripts, enter the command “conda create -n sar python=2.7.” You can choose to name your environment something other than sar.
2. When prompted, enter “y” to proceed with the installation.
3. To activate this python environment in the future, use the command “conda activate sar” to enter the environment and “conda deactivate” to leave it. Notice as you use these commands that you will move from “base” to “sar” environments, as shown below.

```
(base) fsh@fsh-VirtualBox:~$ conda activate sar
(sar) fsh@fsh-VirtualBox:~$ conda deactivate
(base) fsh@fsh-VirtualBox:~$
```

4. Now, let’s set up our “sar” environment with the required python packages: NumPy, SciPy, SimPy, json, pillow, OsGeo/GDAL, simplekml, mpmath. Activate the “sar” environment by entering the command “conda activate sar” into the terminal. Install gdal, numpy, pillow, simplekml and scipy by entering the command “conda install -c conda-forge gdal numpy=1.15 pillow simplekml scipy” as shown below.

```
(base) fsh@fsh-VirtualBox:~$ conda activate sar
(sar) fsh@fsh-VirtualBox:~$ conda install -c conda-forge gdal numpy=1.15 pillow
simplekml scipy█
```

5. When prompted, enter “y” to proceed.
6. Enter the command “pip install simpy mpmath” to install additional prerequisites.
7. To confirm that you have installed all of the Python packages, you can enter the command “python.” Then enter “import gdal” or “import” followed by any of the other packages. If no errors pop up in your terminal and the arrows that indicate a new line appear, then the packages have been installed correctly.
8. Enter the command “exit (y)” to leave python.
9. To view the version and other information about the packages you have installed, in the “sar: environment of the terminal, enter the command “conda list pillow” or “conda list” plus any of the packages, as shown below.

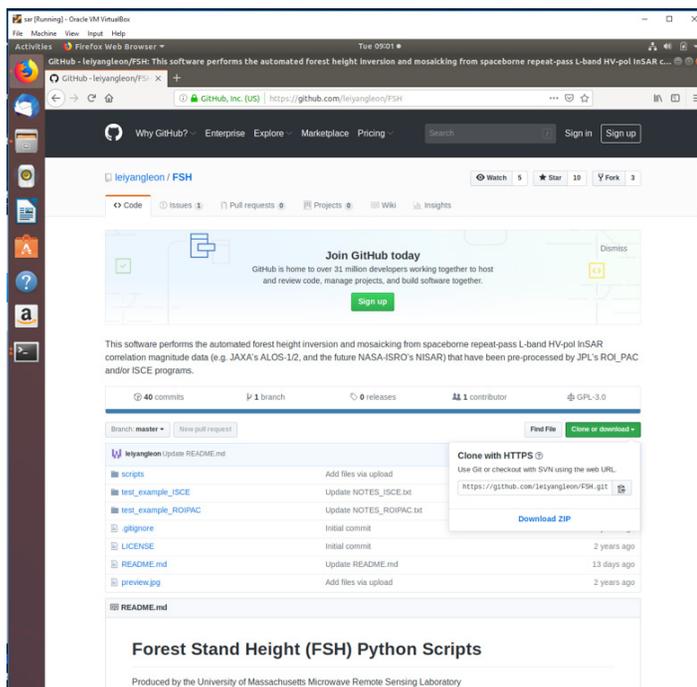
```
(sar) fsh@fsh-VirtualBox:~$ conda list pillow
# packages in environment at /home/fsh/anaconda2/envs/sar:
#
# Name                Version             Build Channel
pillow                 5.4.1               py27h00a061d_1000 conda-forge
(sar) fsh@fsh-VirtualBox:~$ █
```

3 DOWNLOAD MATERIALS FOR THE TUTORIAL

The Python scripts needed for this tutorial, written by Y. Lei the principal developer of the FSH technique, and an example dataset can be freely downloaded from GitHub or from the SERVIR Global website. The example data are preprocessed, and using these data allow you to skip sections 4 and 5 and proceed to section 6.

3.1 Obtaining the Scripts from GitHub

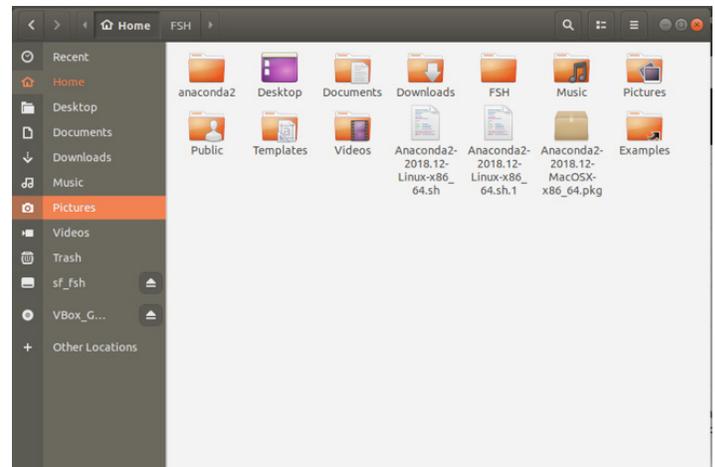
1. Navigate to the GitHub page <https://github.com/leiyangleon/FSH> using FireFox or another internet application on your virtual machine.
2. Click the green “Clone or download” button and copy the link shown under the “Clone with HTTPS” pop up window, as shown below.



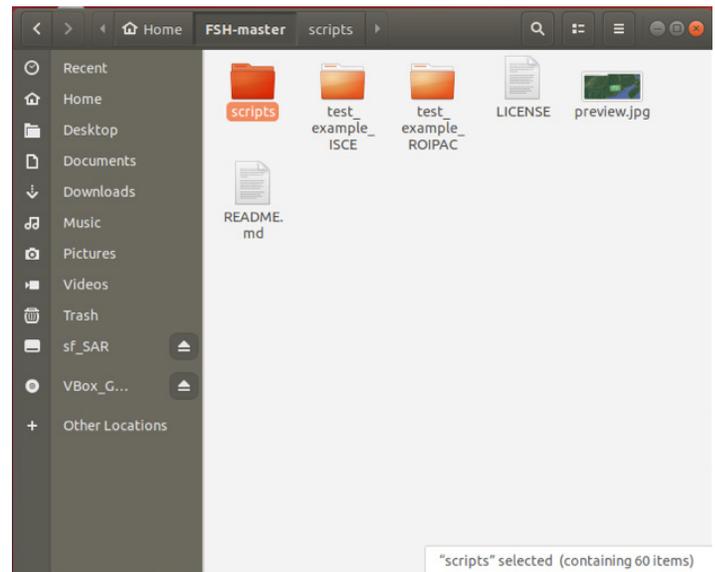
3. Open a terminal, and if you are not already in the “sar” environment created in section 2.3, navigate to the “sar” environment by entering the command “conda activate sar.”
4. Enter the command, “git clone” followed by pasting in the link you copied from the GitHub: <https://github.com/leiyangleon/FSH.git>, as shown below.

```
(sar) fsh@Fsh-VirtualBox:~$ git clone https://github.com/leiyangleon/FSH.git
```

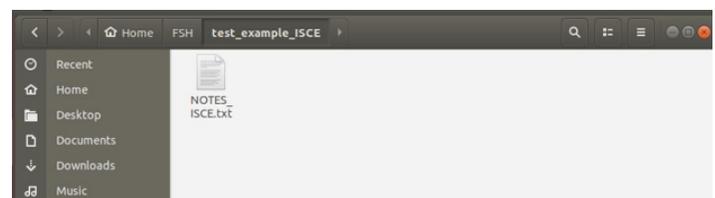
5. If git does not exist on your virtual machine, follow the prompts to install it using the command “sudo apt install git,” followed by your virtual machine’s password.
6. If you navigate to “Home” under the “Files” tab from the menu on the left hand side, you should be able to see the “FSH” folder that you downloaded with all of the scripts necessary for this tutorial.



7. Within your FSH folder, there should be three folders (scripts, test_example_ISCE, and test_example_ROIPAC) and three files (LICENSE, preview.jpg, README.md) inside, as shown below.



8. While there are two folders that seem like they should contain data (test_example_ISCE and test_example_ROIPAC), if you open these folders than you will find that they only include a text file, and no SAR data or other required files, as shown below.

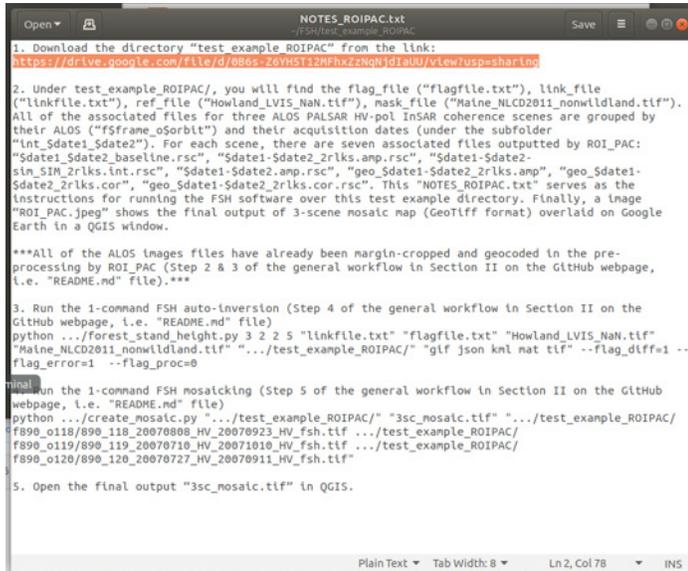


9. To download the example data, please proceed to the next section (3.2).

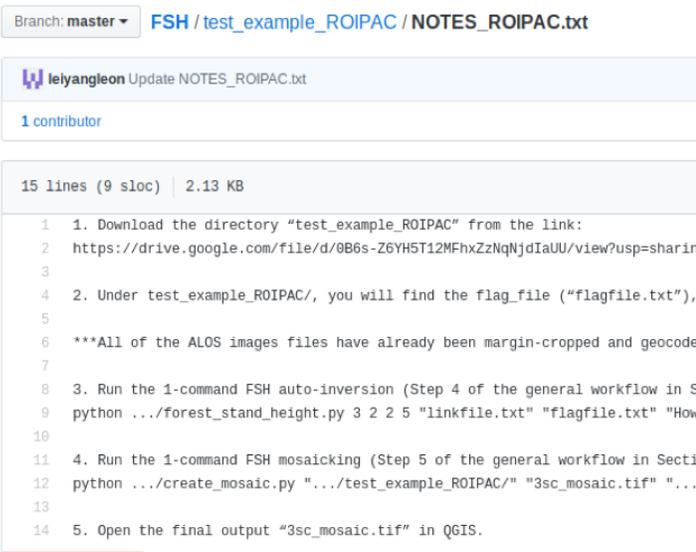
3.2 Downloading Example Data

The example data consists of three scenes, including a central scene with overlapping NASA LVIS LiDAR data and two adjacent scenes.

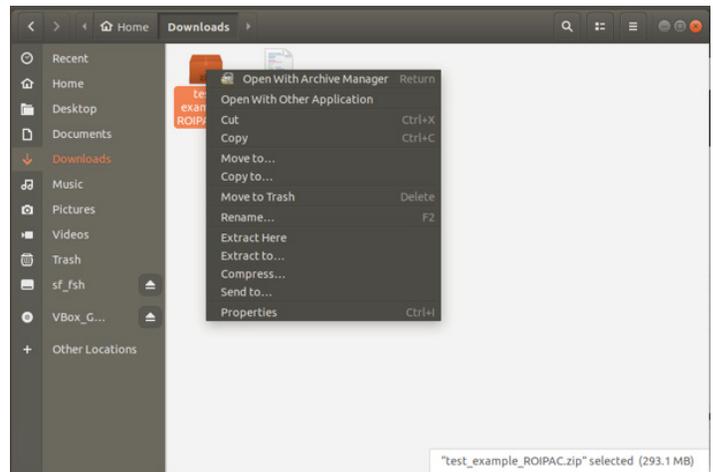
1. You can access the link to download the example data by opening the text document within the test_example_ROIPAC and test_example_ISCE folders respectively. See below for the location of the link within the text file for the ROIPAC data.



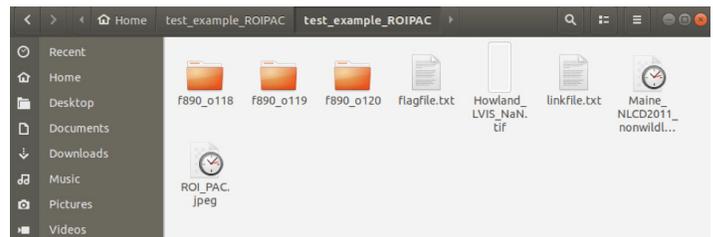
2. You can also navigate to this link through the GitHub, following the same folder tree, as shown below.



3. Either way you choose to find the link to the data, copy and paste this link into the web browser on your virtual machine. While both datasets are compatible with the FSH scripts, we will use the ROI_PAC as our example for this tutorial.
4. Choose the download icon to download the dataset from the Google Drive link. When prompted for confirmation due to the large size of the file, select "download anyway". When prompted to open the file, choose "save file" and press "OK."
5. Within "Files," navigate to "downloads." Right click on the example data zip, and from the pop up menu, choose "Extract to.."



6. Navigate to "Home" and press the green "Select" button to extract the example data there.
7. Within the "test_example_ROIPAC" folder, you will find "flagfile.txt" (referred to as the flag_file in the scripts), "lnkfile.txt" (link_file), "Howland_LVIS_NaN.tif" (ref_file), and "Maine_NLCD2011_nonwildland.tif" (mask_file). All of the associated files for the three ALOS PALSAR HV-pol InSAR coherence scenes are grouped by their ALOS ("f\$frame_o\$orbit") and their acquisition dates (under the subfolder "int_\$date1_\$date2"). For each scene, there are seven associated files outputted by the ROI_PAC software: "\$date1_\$date2_baseline.rsc", "\$date1-\$date2_2rlks.amp.rsc", "\$date1-\$date2-sim_SIM_2rlks.int.rsc", "\$date1-\$date2.amp.rsc", "geo_\$date1-\$date2_2rlks.amp", "geo_\$date1-\$date2_2rlks.cor", "geo_\$date1-\$date2_2rlks.cor.rsc". Finally, "ROI_PAC.jpeg" shows the final output of 3-scene mosaic map (GeoTiff format) overlaid on Google Earth in a QGIS window. Please see below for the file layout.



4 PROCESSING RAW SAR DATA

When processing SAR data, corrections are made for the motion of the satellite and image projection effects that arise from the atmosphere, viewing geometry and topography of the Earth. The steps of processing of ALOS SAR data from raw samples for the satellite include range compression, azimuth compression resulting in an SLC, and finally projection into map coordinates. Software for processing raw data into SLCs can be obtained both commercially and through open-source licensing agreements. Of the open source licensing processors, there are two that have been used for processing raw ALOS data into SLCs and then into estimates of FSH. These are the ROI_PAC (Repeat Orbit Interferometry PACKAGE) and ISCE (InSAR Scientific Computing Environment). In this tutorial, we focus on ROI_PAC as it has completed its development lifetime and is somewhat easier to obtain than ISCE. At the time of this writing, ISCE remains under development. With this in mind, the preprocessing scripts in section 3.2 and the scripts in section 4 that estimate FSH from SLCs have been designed to work with outputs from both ROI_PAC and ISCE.

4.1 Obtaining the Scripts from GitHub

1. Obtain the ROI_PAC processing software in tgz (gzipped tar) format from: http://www.openchannelfoundation.org/projects/ROI_PAC
2. Download and install a fortran compiler (e.g. gfortran) and the fftw library. See <http://roipac.org/cgi-bin/moin.cgi/installation> for additional details on the installation of ROI_PAC software.
3. Utilize the test data set that comes with the ROI_PAC software distribution to test the software installation. You can find the details of how to test the software in the ROI_PAC installation subdirectory: `fullpath/contrib/multtest.sh` where `fullpath` refers to the folder where you unzipped the ROI_PAC installation archive.

4.2 Processing ROI_PAC/ISCE outputs with Python scripts

1. To open the terminal within your virtual machine, press `ctrl, alt and t`.
2. Crop the ROI_PAC/ISCE output and eliminate the image margins by running the standalone python scripts `CROP_ROIPAC.py` and `CROP_ISCE.py` respectively. Please note that the amount cropped is hard coded based on the dimensions of the ALOS SAR image. The code would need to be adjusted for ALOS-2 and future NISAR images.

- For ROI_PAC processed results enter the command `python directory_of_scripts/CROP_ROIPAC.py dirname date1 date2`
- For ISCE-processed results, run the following command within the execution of `insarApp.py` `python directory_of_scripts/CROP_ISCE.py`

You will need to replace three parameters in these commands:

- Replace `directory_of_scripts` with the location of the ROI_PAC `amp/cor` files
- Replace `date1` with the date for the 1st SAR acquisition
- Replace `date2` with the date for the second SAR acquisition

For information on how to geocode the ROI_PAC/ISCE output, please see the Chapter 2 training module.

5 FILE CREATION & ORGANIZATION

5.1 File Structure

The data should be organized in a file structure such that the individual folders hold results from individual interferograms between two dates (the SLCs and ancillary data for individual scene (frame) and orbit (path) numbers). For any one frame and path number, there may exist multiple interferograms, related to multiple repeat-pass combinations of data from two different dates. These interferograms should be stored in sub-directories that have the naming convention: `int_date1_date2`. Scenes from differing frame and paths can be interferometrically processed in order to create an estimate of FSH over an extended geographic region.

The interferogram subdirectories will hold all of the data and information necessary for creating and documenting interferograms made for an observation on two specific dates (`date1` and `date2`). For ROI_PAC-processed data, the most important file looks like `geo_date1-date2_2rlks.cor` and `geo_date1-date2_2rlks.cor.rsc`. The resource “.rsc” file is a text file that has information the location and size of the geolocated correlation data held in `geo_date1-date2_2rlks.cor`. The format of the correlation file is known as sample-interleaved, or an `rmg-format` file.

Since radar data are organized in terms of orbits and scenes, in order to make a map of FSH over an extended geographic region it is necessary to mosaic the images. While the process of mosaicking can be done either before or after the estimation of FSH, it is best to do this beforehand to take advantage of the overlap region between images in adjacent paths. In these regions, while the value of the coherence magnitude may vary due to the fact that the observations (and image pairs) have occurred from different orbits (and hence, different dates), the overlap regions can be used to correct for these temporal differences and to adjust the coefficients for the empirical relationships of the SAR products to estimates of FSH.

For each ROI_PAC-processed scene, the following files should be located in a directory with the format “`f$frame_o$orbit/int_ $date1_ $date2`”:

```
$date1_ $date2_baseline.rsc
$date1- $date2.amp.rsc
$date1- $date2_2rlks.amp.rsc
$date1- $date2-sim_SIM_2rlks.int.rsc
geo_ $date1- $date2_2rlks.amp
geo_ $date1- $date2_2rlks.cor
geo_ $date1- $date2_2rlks.cor.rsc
```

Please note that the ROI_PAC's `process_2pass.pl` should be run with 2 range looks and 10 azimuth looks in both coherence estimation and multi-looking (equivalent to a 30m-by-30m area for JAXA's ALOS), with the following lines added to the process file:

```
Rlooks_int = 2
Rlooks_sim = 2
```

```
Rlooks_sml = 2
pixel_ratio = 5
```

A 5-point triangle window is hardcoded in ROI_PAC, which is equivalent to a 2-point rectangle window. For further details on running ROI_PAC, refer to the ROI_PAC manual. For each ISCE-processed scene, the following files should be located in a directory with the format “f\$frame_o\$orbit/int_ \$date1_ \$date2”:

```
isce.log
resampOnlyImage.amp.geo
resampOnlyImage.amp.geo.xml
topophase.cor.geo
topophase.cor.geo.xml
```

Please note that ISCE’s insarApp.py should be run with 2 range looks and 10 azimuth looks in both coherence estimation and multi-looking (equivalent to a 30m-by-30m area for JAXA’s ALOS), with the following lines added to the process file:

```
<property name="range looks">1</property>
<property name="azimuth looks">5</property>
```

A 5-point triangle window is hardcoded in ISCE, which is equivalent to a 2-point rectangle window. The .amp/.cor images then need to be multilooked by a factor of two. For further details on running ISCE see the ISCE manual.

The location of the output files depends on whether they are related to the overall processing of the entire dataset, or are directly associated with a single scene. Examples of each would be the SC iteration files as a general output, and a single forest stand height image as a scene-specific output. The general outputs will be stored in a directory named “output” located within the main file directory (file_directory). The scene specific outputs will be stored with the other scene data as described earlier.

5.2 Create Flag File

Once the data have been organized into directories of scenes described by their individual row and path numbers, and the interferograms have been examined to determine which SLC pairs yield the data with the highest coherence (i.e. least amount of temporal decorrelation), there remains the task of creating what is known as a “flag file” and a “link file.” In this context, the flag file is a listing of all the interferograms that will be used in creating the region-wide mosaic of FSH. In the example dataset, there are three such row/path combinations that will create a three-scene mosaic of FSH located in central Maine. The middle of the three scenes overlaps with the forest height data (ref_file) discussed in Section 1.2, and all scenes are within the region where identifications of forest/non-forest (mask_file) is used for determining geographic locations where the FSH algorithm will be applied. An example of the contents of a flag file in text format is:

```
001  890_120_20070727_MV_20070911_MV  070727 070911  890  120  MV
002  890_119_20070710_MV_20071010_MV  070710 071010  890  119  MV
003  890_118_20070808_MV_20070923_MV  070808 070923  890  118  MV
```

In this example, the first column of numbers indicates the interferogram number, the second is the root file name of the data that forms the interferogram, the third and fourth are the dates that the data were collected for the interferometric pairs, the fifth and sixth are the satellite path and orbit respectively, and the last indicates the polarization of the data.

5.3 Create Link File

The link file provides information on which files are expected to have some degree of geographic overlap, and will be used in propagating the coefficients of FSH. While many files may have such a geographic overlap, and that indeed this overlap can be automatically calculated, a separate link file is desired so that links can be added or broken as necessary in order to account for the varying quality of data in the overlap region used to estimate the coefficients (e.g. a scene with a particularly high degree of temporal decorrelation can be removed from the link list). A simple example of the test-formatted link file is:

```
2  1
2  3
```

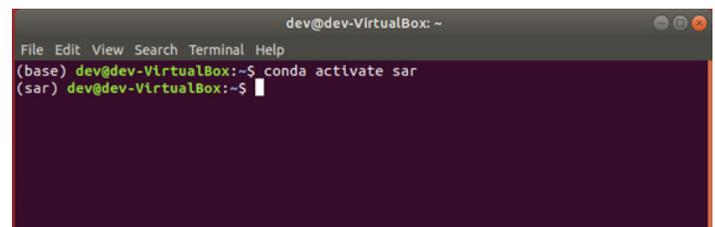
This indicates that image 2 is connected to image 1, and that image 2 is also connected to image 3 (and also that images 1 and 3 are also not connected).

6 ESTIMATE FOREST STAND HEIGHT

Once the SLC, forest/non-forest mask, vegetation height, link file, and flag files are created and put into place, you can run the FSH scripts by calling them in the terminal and passing the input file names and ancillary information as arguments. You can run each script one at a time, or call the main script. For this tutorial, we will run the FSH scripts from Anaconda in the virtual machine we set up. All five possible final output data types are produced. Please note that runtime does not increase linearly with each additional scene. Runtime for most of the steps are linear in the number of scenes; however, the core part of the inversion and mosaicking depends on the number of edges, which increases a bit faster as the number of scenes increases.

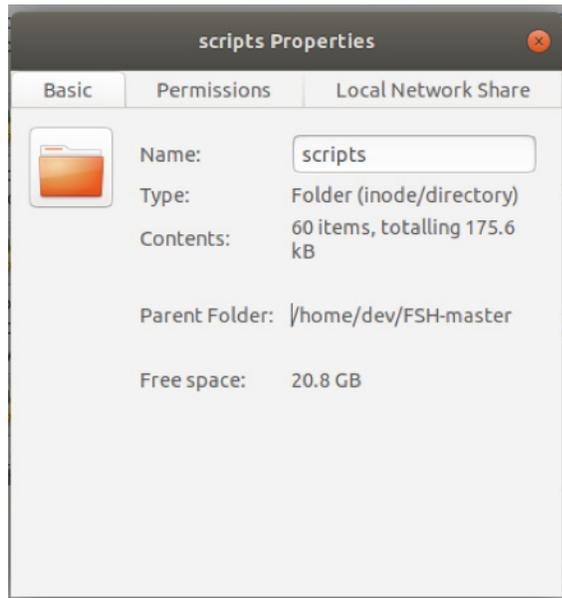
6.1 Access the Anaconda Environment

1. Press the green arrow to run your virtual machine.
2. To open the terminal press ctr, alt, and t.
3. Type in the command “conda activate sar” to access the Anaconda environment and dependencies that you installed in section 2.2. Notice that your terminal should change from “base” to “sar” environment as shown below.



6.2 Find the Directory of Scripts and Example Data

1. In order to run the scripts from your terminal, you will need the directory to your scripts and the directory to your example data. To get to the directory of your files, right click within the folder that they are stored.
2. From the popup menu that appears, choose "Properties."
3. You can then copy and paste the Parent Folder plus the folder name from the properties window into your script. Below is the properties window for the folder that holds my scripts.



6.3 Run Main FSH Script

1. Now let's call the first script! For the ROI_PAC processed example files, enter the command into the terminal "python ../forest_stand_height.py 3 2 2 5 "linkfile.txt" "flagfile.txt" "Howland_LVIS_NaN.tif" "Maine_NLCD2011_non-wildland.tif" ../test_example_ROIPAC/ "gif json kml mat tif" --flag_proc=0" into the terminal, where "..." is the path to your forest_stand_height script and your example ROI_PAC data respectively, as shown below. For the ISCE data this would look like "python /home/dev/FSH-master/scripts/forest_stand_height.py 3 2 2 5 "linkfile.txt" "flagfile.txt" "Howland_LVIS_NaN.tif" "Maine_NLCD2011_nonwildland.tif" /home/dev/Downloads/test_example_ISCE/test_example_ISCE/ "gif json kml mat tif" --flag_proc=1"

```
(sar) fsh@fsh-VirtualBox:~$ python /home/fsh/FSH/scripts/forest_stand_height.py 3 2 2 5 "linkfile.txt" "flagfile.txt" "Howland_LVIS_NaN.tif" "Maine_NLCD2011_nonwildland.tif" /home/fsh/test_example_ROIPAC/test_example_ROIPAC/ "gif json kml mat tif" --flag_proc=0
```

2. Let's review what each of these inputs mean:
 - First, we call "python" in order to run the python scripts within the terminal. The following parameters for the FSH scripts listed in brackets are optional, while the other parameters require input.
 - file_directory/forest_stand_height.py calls the main FSH script that in turn calls the rest of the scripts necessary to calculate FSH. You must provide

the appropriate file directory to this script. For this example, the file directory is "/home/fsh/FSH/scripts/forest_stand_height.py."

- Scenes - enter the number of scenes in the dataset. This must be an integer. If using a single radar scene, enter 1. In this example, we have 3 scenes.
- Edges - enter the number of scene to scene borders. If using a single radar scene, enter 0. In this example, we have 2 scene to scene borders.
- start_scene (int) - flag value of the central scene that overlaps the forest stand height ground truth (e.g. LiDAR, field) data. In this example, the central scene is 2.
- iterations (int) - number of iterations to run the nonlinear least squares part of the model. In this example, we want to run the nonlinear least squares part of the model 5 times.
- link_file - a text file that lists all the edge scene pairs. Each line consists of the two numbers that correspond to the flag numbers for those two scenes. (e.g. "2 1" would be the line for the edge of the above scenes 001 and 002). If using a single ALOS scene, this file is unneeded, and input "-" instead of the file name for the terminal arguments. For this example, the file name is "linkfile.txt."
- flag_file - a text file that lists all the flags and corresponding full file names and associated file information (dates, scene location (frame#, orbit#), polarization). In this example, the file name is "flagfile.txt." Examples of what this text file would contain are:
001 890_120_20070727_HV_20070911_HV 070727 070911 890 120 HV
002 890_119_20070710_HV_20071010_HV 070710 071010 890 119 HV
003 890_118_20070708_HV_20070923_HV 070708 070923 890 118 HV
- ref_file - reference tree height data (lidar or field inventory) in raster format. Currently the code is set up to use a GeoTIFF file, but other reference data in raster format could potentially be used with some code adjustments. In this example, the reference tree height data is "Howland_LVIS_NaN.tif."
- mask_file - land cover mask that excludes all water areas and areas of human disturbance (urban, agriculture). This is currently set up to be a GeoTIFF file. Other reference data in raster format could potentially be used with some code adjustments. File must be in degrees (i.e., EPSG 4326). This file is recommended, but optional. If unused, put "-" in place of the file name for the terminal arguments. For this example, the file name is "Maine_NLCD2011_nonwildland.tif."
- file_directory - the root directory to folders containing the individual SAR scenes. Each scene should have a directory named "f\$frame_o\$orbit" (e.g. "f890_o120" for the above scene 001). This directory contains either the input ROI_PAC processed or ISCE processed files and is also the output location for all files that are associated with that scene. For this example, the directory is: /home/fsh/test_example_ROIPAC/test-ex-

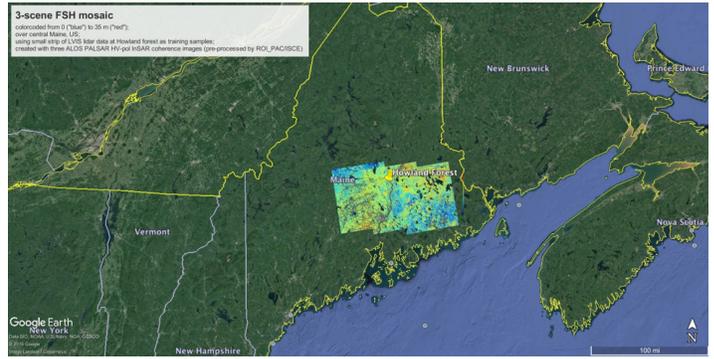
ample_ROIPAC/. Please note that no quotes are used in the terminal for this parameter.

- Output file types - the list of output formats should be in quotes, and can contain one or all of the following: "tif kml gif mat json". In other words, output formats can be created for any of these options. For this example, all options are listed.
 - The command option --flag_proc 0 indicates that the input data has been processed into SLCs by the ROI_PAC algorithm. If the data was processed by ISCE, please use 1 instead. For this example, we use a 0 to indicate that the data was processed by ROI_PAC.
3. The scripts are also able to be run with a single radar scene. To do this use "--" instead of a link_file name, and in the input have 0 edges.
 - For example: python ../forest_stand_height.py 1 0 1 5 - "flagfile.txt" "Howland_LVIS_NaN.tif" "Maine_NLCD2011_nonwildland.tif" /directory_of_files/ "gif json kml mat tif" --flag_proc=1
 4. In the case that you are running the FSH scripts on your own data, or would like to call each FSH script individually in the command line, please find the inputs, outputs, and terminal command lines in section 6.5. Please note that there are additional, unrequired parameters for the forest_stand_height.py that are explained in section 6.5 that are not included in the example. Otherwise, proceed to section 6.4 to generate a mosaic of your forest stand height estimation.

6.4 Generate Mosaic

1. To create a mosaic of the generated forest height maps for all the scenes in GeoTiff format, run the following command "python directory_of_scripts/create_mosaic.py directory mosaic_file list_of_files" in the terminal. You will need to replace three parameters.
 - Replace directory_of_scripts with the location of the scripts.
 - Replace mosaic_file with the name you would like to give your final mosaic of forest stand heights.
 - Replace list_of_files with paths to each map that you would like to be combined within the mosaic in the format "file1 file2 file3."
2. For example:


```
/home/dev/test_example_ROIPAC/test_example_ROIPAC//
create_mosaic.py /home/dev/test_example_ROIPAC/test_ex-
ample_ROIPAC/ "3sc_mosaic.tif" ".../test_example_ROIPAC/
f890_o118/890_118_20070808_HV_20070923_HV_fsh.tif
.../test_example_ROIPAC/f890_o119/890_119_20070710_
HV_20071010_HV_fsh.tif .../test_example_ROIPAC/
f890_o120/890_120_20070727_HV_20070911_HV_fsh.tif"
```
3. Following is a snapshot of the expected mosaicked forest stand height results using the example dataset.



6.5 Overview of Scripts

Let's review the scripts in the general order that they are called, including their main purpose, inputs, outputs, and terminal commands.

1. forest_stand_height.py is the main script, which in turn calls nine other scripts with a total runtime of around 23 minutes 22 secs for the example data. Some of the other scripts call additional scripts.

The command line call is:

- python file_directory/forest_stand_height.py scenes edges start_scene iterations link_file flag_file ref_file mask_file file_directory "output_file_types" [--Nd_pairwise] [--Nd_self] [--N_pairwise] [--N_self] [--bin_size] [--flag_sparse] [--flag_diff] [--flag_error] [--numLooks] [--noiselevel] [--flag_proc] [--flag_grad]."

The inputs for this script in the order entered into the terminal are:

- scenes (int) - number of scenes in the data set
- edges (int) - number of edges (aka scene-scene borders)
- start_scene (int) - flag value of the central scene that overlaps the ground truth (e.g. LiDAR, field) data
- iterations (int) - number of iterations to run the nonlinear least squares part of the model
- Link_file (string) - file name of the file that lists all the edge scene pairs or '-' if processing a single scene
- flag_file (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- ref_file (string) - filename of reference data raster file (ground truth data, e.g. LiDAR, field)
- maskfile (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable; if no mask is a available input '-' as the filename)
- file_directory (string) - directory path of where the input and output files are located

- a. filetypes (string) - list of the desired output file types formatted as a single string with quotation marks (e.g. "kml json tif")
- b. [--Nd_pairwise] (int) - optional pixel-averaging parameter for edge fitting (default=20)
- c. [--Nd_self] (int) - optional pixel-averaging parameter for central scene fitting (default=10)
- d. [--N_pairwise] (int) - optional pixel-averaging parameter for edge error metrics (default=20)
- e. [--N_self] (int) - optional pixel-averaging parameter for central scene error metrics (default=10)
- f. [---bin_size] (int) - optional bin size for density calculation in sparse data cloud fitting (default=100)
- g. [--flag_sparse] (int) - optional flag for sparse data cloud filtering (choose 0 or 1, default=0)
- h. [--flag_diff] (int) - optional flag for exporting differential height maps (choose 0 or 1, default=0)
- i. [--flag_error] (int) - optional flag for exporting .json error metric files (choose 0 or 1, default=0)
- j. [--numLooks] (int) - number of looks in the correlation estimation (default=20)
- k. [--noiselevel] (float) - sensor thermal noise level (ALOS's value hardcoded as default if no value provided)
- l. [--flag_proc] (int) - flag for InSAR processor selection (choose 0 for ROI_PAC or 1 for ISCE, default=0)
- m. [--flag_grad] (int) - flag for correction of large-scale temporal change gradient (choose 0 or 1, default=0)

There are no direct outputs from this script, as all the file outputs are created within the scripts that are called by this main script.

The scripts called by forest_stand_height.py are: auto_tree_height.py, read_linkfile.py, intermediate.py, intermediate_self.py, auto_mosaicking_new.py, write_deltaSC.py, write_mapfile_new.py, write_diff_height_map.py, and cal_error_metric.py

2. auto_tree_height_many.py is called by the forest_stand_height. This script extracts data from ROI_PAC/ISCE output files and formats them for use in the rest of the scripts. For each scene, this script runs auto_tree_height_single.py, and then saves the output correlation magnitudes, kz, and coordinates in a .mat file, and geo data (lines, samples, corner latitude and longitude, and latitude and longitude step size) in a text file.

The command line call for this script is python directory_of_scripts/auto_tree_height_many.py scenes flagfile directory

The inputs for auto_tree_height_many.py are:

- scenes (int) - number of scenes in the data set
- flagfile (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- directory (string) - directory path of where the input and output files are located
- [--numLooks] (int) - number of looks in the correlation estimation (default=20)
- [--flag_proc] (int) - flag for InSAR processor selection (input 0 for ROI_PAC or 1 for ISCE, default=0)
- [--flag_grad] (int) - flag for correction of large-scale temporal change gradient (input 0 or 1, default=0)

The outputs for this script are:

- scenename_orig.mat - .mat file that stores correlation map, kz value, and corner coordinates
- scenename_geo.txt - text file that stores the geodata (width, lines, corner lat and lon, and lat and lon step values)

Auto_tree_height_many.py calls auto_tree_height_single_ROIPAC and auto_tree_height_single_ISCE.

3. auto_tree_height_single_ROIPAC.py calls the script read_rsc_data.py in order to read the value of the given parameter from the rsc file produced by ROI_PAC processing of SAR data. This script also calls remove_corr_bias.py to remove correlation bias associated with ROI_PAC. This script is called by auto_tree_height_many.py and cannot be run in the terminal on its own as it needs to be iterated for each scene in the analysis.

The inputs for this script are:

- directory (string) - directory path of where the input and output files are located
- date1 (string) - date of the first image of the interferogram (format however they are listed in the scene data text file, such as 070911 for September 11, 2007)
- date2 (string) - date of the second image of the interferogram (same format as date1)
- numLooks (int) - number of looks in the correlation estimation
- noiselevel (float) - sensor thermal noise level (ALOS's value hardcoded as default)
- flag_grad (int) - flag for correction of large-scale temporal change gradient (input 0 or 1)

The outputs for this script are:

- `corr_vs` (numpy array) - array of the correlation magnitudes
- `kz` (float) - kz parameter
- `coords` (numpy array) - array of max lat and lon values in the format [north, south, west, east]
- `geo_width` (int) - number of columns of image data
- `geo_nlines` (int) - number of rows of image data
- `corner_lat` (float) - max latitude value (north)
- `corner_lon` (float) - min latitude value (west)
- `step_lat` (float) - latitude pixel size in decimal degrees
- `step_lon` (float) - longitude pixel size in decimal degrees

4. `read_rsc_data.py` reads a parameter from the `ROI_PAC.rsc` text output file. This script is called by `auto_tree_height_single_ROIPAC.py` and is not meant to be run in the terminal.

Inputs for this script are:

- `filename` (string) - file name of the `ROI_PAC` text output file containing the desired parameter (may include subdirectories containing the `ROI_PAC` output files - everything lower than the main file directory)
- `directory` (string) - directory path of where the input and output files are located
- `param` (string) - name of the desired parameter

Outputs for this script are the parameter values as floats (result)

5. `auto_tree_height_single_ISCE.py` calls `remove_corr_bias.py` to remove correlation bias associated with ISCE. This script is called by `auto_tree_height_many.py` and cannot be run in the terminal on its own since it needs to be iterated for each scene in the analysis.

The inputs for this script are:

- `directory` (string) - directory path of where the input and output files are located
- `date1` (string) - date of the first image of the interferogram (format however they are listed in the scene data text file, such as 070911 for September 11, 2007)
- `date2` (string) - date of the second image of the interferogram (same format as `date1`)
- `numLooks` (int) - number of looks in the correlation estimation
- `noiselevel` (float) - sensor thermal noise level (ALOS's value hardcoded as

default if no value provided)

- `flag_grad` (int) - flag for correction of large-scale temporal change gradient (input 0 or 1)

The outputs for this script are:

- `corr_vs` (numpy array) - array of the correlation magnitudes
- `kz` (float) - kz parameter
- `coords` (numpy array) - array of max lat and lon values in the format [north, south, west, east]
- `geo_width` (int) - number of columns of image data
- `geo_nlines` (int) - number of rows of image data
- `corner_lat` (float) - max latitude value (north)
- `corner_lon` (float) - min latitude value (west)
- `step_lat` (float) - latitude pixel size in decimal degrees
- `step_lon` (float) - longitude pixel size in decimal degrees

6. `intermediate.py` calculates the overlap between each pair of images. This script is called by `forest_stand_height.py`.

To run in the terminal, enter the command: `python directory_of_scripts/intermediate.py edges start_scene linkfile maskfile flagfile ref_file directory`

The inputs for this script are:

- `edges` (int) - number of edges (aka scene-scene borders)
- `start_scene` (int) - flag value of the central scene that overlaps the ground truth forest height data
- `linkarray` (numpy array) - array of the scene pairs that correspond to each edge in the format `array([[scene1, scene2], [scene1, scene3], etc])`
- `maskfile` (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable)
- `flagfile` (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- `ref_file` (string) - filename of the reference data raster file
- `directory` (string) - directory path of where the input and output files are located

There's no direct output for this script since all file outputs are created in subprocesses.

`Intermediate.py` calls `intermediate_self.py` and `intermediate_pairwise.py`.

7. `remove_corr_bias.py` removes the correlation bias associated with processing by ROI_PAC or ISCE.

The inputs for this script are:

- `C` (numpy array) - correlation magnitude array
- `numLooks` (int) - number of looks in the correlation estimation

The output for this script is `YC` (numpy array) - correlation magnitude array (with bias removed)

8. `intermediate_pairwise.py` calculates the overlap between each pair of scenes, reading the data directly from `auto_tree_height_single` rather than from an intermediary file. This script in turn calls `flag_scene_file.py` and `remove_nonforest.py`. This script is called by `auto_tree_height_single.py` and is not meant to be run from the terminal.

The inputs for this script are:

- `flag1` (int) - flag value of one scene in the pair
- `flag2` (int) - flag value of the other scene in the pair
- `flagfile` (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- `maskfile` (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable)
- `directory` (string) - directory path of where the input and output files are located
- `filename1_orig.mat`: correlation map and associated parameters for the first scene (generated in previous steps)
- `filename2_orig.mat`: correlation map and associated parameters for the second scene (generated in previous steps)

The outputs for this script are link files: one for each overlapping edge region, with the filename format `flag1_flag2.mat`

9. `intermediate_self.py` calculates the overlap between the forest height validation data and central scene. This script in turn calls `flag_scene_file.py` and `remove_nonforest.py`. This script is called by `intermediate.py` and is not meant to be run from the terminal.

The inputs for this script are:

- `start_scene` (int) - flag value of the central scene that overlaps the ground truth data
- `flagfile` (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)

- `directory` (string) - directory path of where the input and output files are located
- `filename_orig.mat`: correlation map and associated parameters for the central scene (generated in previous steps)
- reference data raster file (already exists; main input)

The output for this script is `self.mat`, a link file for the central scene-ground truth overlap region

10. `flag_scene_file.py` associates flag numbers with the name, dates, ALOS location (frame and orbit), and polarization of each scene. This script is called by `intermediate_pairwise.py`, `write_deltaSC.py`, and `write_mapfile_new.py` and is not meant to be run from the terminal.

The inputs for this script are:

- `flagfilename` (string) - file name of the file that lists all the flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- `flag` (int) - flag of the desired scene
- `directory` (string) - directory path of where the input and output files are located

The output for this script is a `data_array` (list) - list of the data associated with the given flag number.

11. `remove_nonforest.py` removes all non-forest areas from the image based on the non-forest mask file. This script is called by `intermediate_pairwise.py` and `write_mapfile_new.py` and is not meant to be run in the terminal.

The inputs for this script are:

- `I` (numpy array) - the image data
- `func_coords` (numpy array) - array of corner coordinates
- `maskfile` (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable)
- `directory` (string) - directory path of where the input and output files are located

The output for this script is `O` (numpy array) - image without the non-forest sections.

12. `auto_mosaicking_new.py` calculates the S and C parameters automatically by iterating through all the scenes in preparation for forest height estimation. This script is called by `forest_stand_height.py`. `auto_mosaicking_new.py` calls `ls_deltaSC.py` and `read_linkfile.py`

To run in the terminal, enter the command: `python directory_of_scripts/auto_mosaicking_new.py scenes edges start_scene N linkfile directory`

The inputs for this script are:

- scenes (int) - number of scenes in the data set
- edges (int) - number of edges (aka scene-scene borders)
- start_scene (int) - flag value of the central scene that overlaps the ground truth data
- N (int) - number of iterations to run the nonlinear least squares part of the model
- Linkfile - the filename of the file that lists all the edge scene pairs.
- linkarray (numpy array) - array of the scene pairs that correspond to each edge in the format array([[scene1, scene2], [scene1, scene3], etc])
- directory (string) - directory path of where the input and output files are located
- [--Nd_pairwise] (int) - pixel-averaging number for image fitting between two overlapped radar scenes (default=20)
- [--Nd_self] (int) - pixel-averaging number for image fitting between single radar scene and the overlapped ground truth data (default=10)
- [--bin_size] (int) - bin size for density calculation in scatter plot fitting when ground truth data are sparse (default=100)
- [--flag_sparse] (int) - flag for sparse data cloud fitting (input 0 or 1, default=0)

The outputs produced by this script are iteration files (.json format; e.g. "SC_#_iter.json" for "#th iteration) that store the increment steps of S and C parameters and the residual; no values are returned by the function.

13. ls_deltaSC.py runs least squares on the change in S and C parameters. This script in turn calls cal_KB.py. This script is called by auto_mosaicking_new.py and is not meant to be run from the terminal.

The inputs for this script are:

- dp (numpy array) - array of increment steps of S and C parameter values
- edges (int) - number of edges (aka scene-scene borders)
- scenes (int) - number of scenes in the data set
- start_scene (int) - flag value of the central scene that overlaps the ground truth data
- linkarray (numpy array) - array of the scene pairs that correspond to each edge in the format array([[scene1, scene2], [scene1, scene3], etc])
- directory (string) - directory path of where the input and output files are located
- Nd_pairwise (int) - pixel-averaging number for image fitting between two overlapped radar scenes
- Nd_self (int) - pixel-averaging number for image fitting between single

radar scene and the overlapped ground truth data

- bin_size (int) - bin size for density calculation in scatter plot fitting
- flag_sparse (int) - flag for sparse data cloud filtering (input 0 or 1)

The outputs for this script are:

- changeSC (numpy array) - updated S and C parameters as referenced to the average S (=0.6) and C (=13)
- res (float) - residual k and b error compared to k = 1 and b = 0

14. cal_KB.py calculates the K and B parameters. This script in turn calls cal_KB_pairwise_new.py and cal_KB_self_new.py. This script is called by ls_deltaSC.py and is not meant to be run from the terminal.

The inputs for this script are:

- R (float) - R parameter for this edge
- RSME (float) - RSME parameter for this edge
- R_RSME_files: one for each edge with the filename format scene1_scene2_l1andl2.json

The output for this script is YY (numpy array), an array of k and b values.

15. cal_KB_pairwise_new.py calculates K and B between image pairs. In turn, this script calls arc_sinc.py, mean_wo_nan.py, extract_scatterplot_density.py, and remove_outlier.py. This script is called by cal_KB and is not meant to be run in the terminal.

The inputs for this script are:

- scene1 (int) - flag value of one scene in the pair
- scene2 (int) - flag value of the other scene in the pair
- deltaS1 (float) - change in S value for one scene in the pair
- deltaC1 (float) - change in C value for one scene in the pair
- deltaS2 (float) - change in S value for the other scene in the pair
- deltaC2 (float) - change in C value for the other scene in the pair
- directory (string) - directory path of where the input and output files are located
- Nd_pairwise (int) - pixel-averaging number for image fitting between two overlapped radar scenes
- bin_size (int) - bin size for density calculation in scatter plot fitting
- link files: one for each overlapping edge region, with the filename format scene1_scene2.mat (generated in previous steps)

The outputs for this script are:

- k (float) - k parameter for this edge
- b (float) - b parameter for this edge

16. `cal_KB_self_new.py` calculates K and B between the central image and the forest height validation data. In turn, this script calls `arc_sinc.py`, `mean_wo_nan.py`, `extract_scatterplot_density.py`, and `remove_outlier.py`. This script is called by `cal_KB` and is not meant to be run in the terminal.

The inputs for this script are:

- `deltaS2` (float) - change in S value for the central scene
- `deltaC2` (float) - change in C value for the central scene
- `directory` (string) - directory path of where the input and output files are located
- `Nd_self` (int) - pixel-averaging number for image fitting between single radar scene and the overlapped ground truth data
- `bin_size` (int) - bin size for density calculation in scatter plot fitting
- `flag_sparse` (int) - flag for sparse data cloud filtering (input 0 or 1)
- `self.mat`: link file for the central scene-ground truth overlap region (generated in previous steps)

The outputs for this script are:

- `k` (float) - k parameter for this edge
- `b` (float) - b parameter for this edge

17. `arc_sinc.py` calculates the inverse sinc function as part of calculating K and B values. This script is called by `cal_KB_pairwise` and `write_mapfile_new.py` and is not meant to be run in the terminal.

The inputs for this script are:

- `X` - A numpy array of x values for the inverse sinc function
- `c_parama` - C parameter (float) from the Forest Stand Height model

The outputs for this script are:

- `y` - a numpy array of y values of inverse sinc function satisfying $x = \text{sinc}(y/C)$

18. `mean_wo_nan.py` calculates and returns the mean of all number values in an array as part of calculating K and B values. This script is called by `cal_KB_pairwise_new.py` and is not meant to be run in the terminal.

Inputs for this script are:

- `A` (numpy array) - input array of values

Outputs for this script are:

- mean of B (A excluding NaN values) (float)

19. `extract_scatterplot_density.py` calculates the 2D histogram of the scatterplot between pairs of forest height and returns the forest height pairs with relatively large density. This script is intended to replace `remove_outlier.py` in order to distinguish between forest disturbance and forest height estimation. This script is called by `cal_KB_pairwise` and is not intended to be run in the terminal.

The inputs for this script are:

- `x` (numpy array) - array of x values of points
- `y` (numpy array) - array of y values of points
- `bin_size` (int) - bin size for density calculation in scatter plot fitting (default = 100)
- `threshold` (float) - density threshold (default = 0.5)

The outputs for this script are:

- `Hm_den` (numpy array) - array of x values of the points with densities above the inputted threshold
- `Pm_den` (numpy array) - array of y values of the points with densities above the inputted threshold

20. `remove_outlier.py` this script is called by `cal_KB_self_new.py`, `cal_KB_pairwise.py`, `cal_KB_pairwise_new.py`, `cal_error_metric_pairwise.py`, and `cal_error_metric_self.py` to remove outliers, and is supplemented by the function of `extract_scatterplot_density.py`.

The inputs for this script are:

- `x` (numpy array) - array of x values of points
- `y` (numpy array) - array of y values of points
- `win_size` (float) - window size to search for neighboring points (defaults to 0.5)
- `threshold` (int) - number of neighboring points needed within the window to not count as an outlier (defaults to 5)

The outputs for this script are:

- `XX` (numpy array) - array of x values of the points excluding those counted as outliers
- `YY` (numpy array) - array of y values of the points excluding those counted as outliers

21. `read_linkfile.py` reads in a text file containing a list of all the scene pairs and returns a 2D array of the pairs. This script is called by `auto_mosaicking_new.py`

To run this script in the terminal, use the following command: `python directory_of_scripts/read_linkfile.py edges filename directory`

The inputs for this script are:

- edges (int) - number of edges (aka scene-scene borders)
- filename (string) - file name of the file that lists all the edge scene pairs
- directory (string) - directory path of where the input and output files are located

The outputs for this script is linkarray (numpy array) - array of the scene pairs that correspond to each edge in the format array([[scene1, scene2], [scene1, scene3], etc])

22. write_deltaSC.py calculates the temporal change parameters (S and C) as referenced to the average values: S=0.6, C=13 based on the final iteration. This script is called by forest_stand_height.py. write_deltaSC.py in turn calls flag_scene_file.py.

To run in the terminal, enter the command: python directory_of_scripts/write_deltaSC.py scenes N flagfile directory

The inputs for this script are:

- scenes (int) - number of scenes in the data set
- N (int) - number of iterations to run the nonlinear least squares part of the model
- flagfile (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- directory (string) - directory path of where the input and output files are located
- SC_#_iter.json: final iteration file (generated in previous steps)

The output for this script is one file per scene that contains delta S and C. The file name format is "scenename_tempD.json"

23. write_mapfile_new.py calculates and writes the tree height map to a file. This script is called by forest_stand_height.py. This script calls flag_scene_file.py, arc_sinc.py, remove_nonforest.py and write_file_type.py.

The inputs for this script are:

- scenes (int) - number of scenes in the data set
- flagfile (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- maskfile (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable) (optional - if no mask available use '-' as an input to forest_stand_height.py)

- directory (string) - directory path of where the input and output files are located
- output_files (string) - list of the desired output file types formatted as a single string (e.g. "kml json tif")
- scenename_orig.mat: correlation map and associated parameters for the central scene (generated in previous steps)
- scenename_tempD.json: delta S and C files produced (generated in previous steps)

There's no direct output (all file output created in write_file_type.py).

24. write_file_type.py writes the input array from the tree height map or the diff_height map to a file, with the file type depending on input parameters: gif, json, kml, mat, or tif. In turn this script calls read_geo_data.py. This script is called by write_mapfile_new.py and is not meant to be run in the terminal.

The inputs for this script are:

- data (numpy array) - array to be written to the file
- outtype (string) - string to signify which input (tree height "stand_height" or differential height "diff_height") is being output
- filename (string) - scene file name
- directory (string) - directory path of where the input and output files are located
- filetype (string) - file extension for the desired output file type (.gif, .json, .kml, .mat, and .tif accepted -> input without the "." (e.g. "kml" instead of ".kml")
- coords (numpy array) - array of max lat and lon values in the format [north, south, west, east]
- reffile (string) - reference filename containing ground truth data (optional; only needed for differential height map)

The outputs for this script are output file(s) of the array image saved in the file type specified in the input.

25. read_geo_data.py reads in latitude, longitude, pixel size, and image size from a GeoTIFF or text file based on ROI_PAC output. This script is called by write_file_type.py and is not meant to be run in the terminal.

Inputs for this script are:

- coord_file (string) - file name of the input data file with the location information (lat/long, step size, image size)
- directory (string) - directory path of where the input and output files are located

Outputs for this script are:

- width (int) - width/number of columns of the image
- nlines (int) - lines/number of rows of the image
- corner_lat (float) - latitude of the upper left corner
- corner_long (float) - longitude of the upper left corner
- post_lat (float) - latitude step size
- post_long (float) - longitude step size

26. write_diff_height_map.py writes the forest differential height map between SAR and overlapping forest height ground truth images. This script is called from forest_stand_height if the parameter --flag_diff is entered.

Inputs for this script are:

- start_scene (int) - flag value of the central scene that overlaps the ground truth data
- reffile (string) - reference filename containing ground truth data
- flagfile (string) - file name of the file that lists all the scene flags and corresponding full file names and associated file date (dates, scene location (frame#,orbit#), polarization)
- maskfile (string) - filename of the mask file that excludes all non-forest areas (mask excluding water and human disturbed areas such as urban and agriculture is also acceptable) (optional; if no masks are available, use '-' as an input to forest_stand_height.py)
- directory (string) - directory path of where the input and output files are located
- output_files (string) - list of the desired output file types formatted as a single string (e.g. "kml json tif")

There is no direct output for this script, as all file output is created in write_file_type.py.

27. cal_error_metric.py calculates the R and RMSE error metrics for the model. This script is called from forest_stand_height.py if the parameter --flag_error is entered. This script calls cal_error_metric_pairwise.py and cal_error_metric_self.py.

The inputs for this script are:

- dp (numpy array) - array of increment steps of S and C parameter values
- edges (int) - number of edges (aka scene-scene borders)
- start_scene (int) - flag value of the central scene that overlaps the ground truth data
- link (numpy array) - array of the scene pairs that correspond to each edge in the format array([[scene1, scene2], [scene1, scene3], etc])
- directory (string) - directory path of where the input and output files are

located

- N_pairwise (int) - pixel-averaging number for scatter plot
- N_self (int) - pixel-averaging number for scatter plot

The output for this script is YY, a numpy array of R and RMSE values.

28. cal_error_metric_pairwise.py calculates the R and RMSE error metrics. This script calls arc_sinc.py, mean_wo_nan.py and remove_outlier.py. It is called by cal_error_metric.py and is not meant to be run in the terminal.

The inputs for this script are:

- scene1 (int) - flag value of one scene in the pair
- scene2 (int) - flag value of the other scene in the pair
- deltaS1 (float) - change in S value for one scene in the pair
- deltaC1 (float) - change in C value for one scene in the pair
- deltaS2 (float) - change in S value for the other scene in the pair
- deltaC2 (float) - change in C value for the other scene in the pair
- directory (string) - directory path to where the input and output files are located
- N_pairwise (int) - pixel-averaging number for the scatter plot
- link files: one for each overlapping edge region, with the filename format scene1_scene2.mat (generated in previous steps)

The outputs for this script are:

- R (float) - R parameter for this edge
- RSME (float) - RSME parameter for this edge
- R_RSME_files: one for each edge, with the filename format scene1_scene2_I1andI2.json

29. cal_error_metric_self.py calculates R and RMSE between the central image and the forest height ground validation data. This script calls arc_sinc.py, mean_wo_nan.py, and remove_outlier.py. This script is called by cal_error_metric.py and is not meant to be run in the terminal.

The inputs for this script are:

- deltaS2 (float) - change in S value for the central scene
- deltaC2 (float) - change in C value for the central scene
- directory (string) - directory path of where the input and output files are located
- N_self (int) - pixel-averaging number for scatter plot
- self.mat: link file for the central scene-ground truth overlap region (generated in previous steps)

The output for this script is YY (numpy array) - array of R and RMSE values.