SNAP-PYRATE

Command line InSAR tutorial

Issued January 2022

Alex McVittie

Introduction

PyRate is an open source toolkit to estimate velocity of surface movement in unwrapped phase interferograms using SBAS processing. It is developed by Geoscience Australia. The source code and setup instructions can be found here: https://github.com/GeoscienceAustralia/PyRate. As of the SNAP 8.0 public release, an additional GAMMA export of SNAP has been added to allow for SNAP to export interferograms in a format that can be read in by PyRate. This tutorial will demonstrate how to generate an interferogram of Mexico City, which is consistently sinking, using Sentinel-1 radar imagery and export to PyRate for further surface velocity analysis.

This tutorial assumes that you are running SNAP 9.0, have PyRate installed and set up and have snaphu configured. Snaphu can be installed on Debian-based Linux systems by running *sudo apt install snaphu*, or by downloading the source code from https://web.stanford.edu/group/radar/softwareandlinks/sw/snaphu/ and compiling manually.

PyRate uses a large amount of memory. It is highly recommended to have 64gb+ of memory available to avoid out of memory errors. While your machine may not have this amount of RAM installed, you can get around this by providing swap space on Linux or MacOS, or increase the page file size in Windows.

All commands in `this font` preceded by a $ to be interpreted as command-line instructions run on the BASH shell. A basic understanding of using BASH is recommended for this tutorial.

Study area

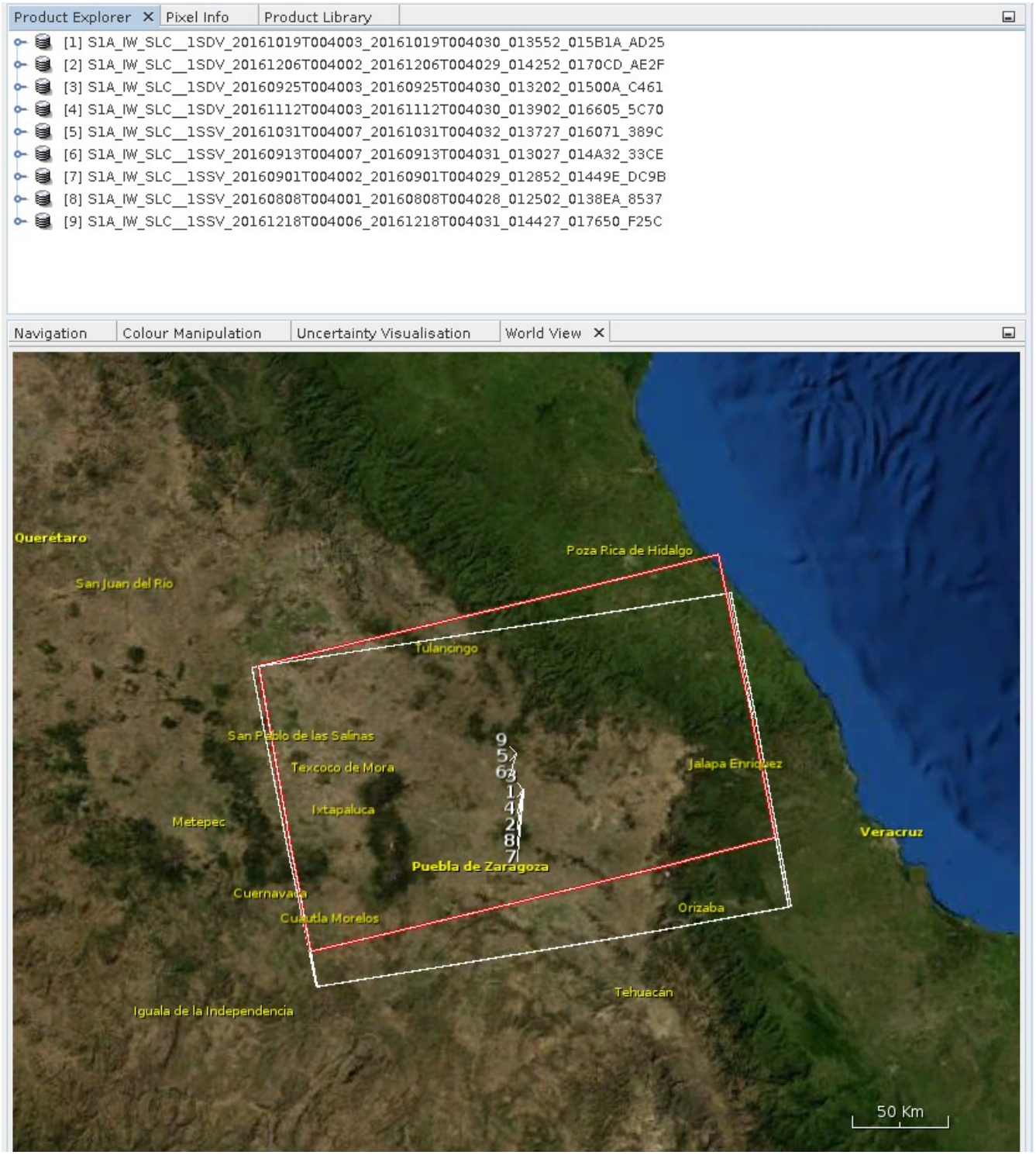
This tutorial focuses on analyzing the ground movement of Mexico City, which is sinking due to groundwater aquifer depletion. You can read more about this issue in this article published by The Guardian.

Data sets

Nine S1 SLC swath datasets are selected for this tutorial from 2016 to generate a set of eight coregistered interferograms. These can be downloaded from sentinel hub. While PyRate can run on a minimum of two interferograms, this is not a good representation of SBAS processing.

S1A_IW_SLC__1SSV_20160808T004001_20160808T004028_012502_0138EA_8537
S1A_IW_SLC__1SSV_20160901T004002_20160901T004029_012852_01449E_DC9B
S1A_IW_SLC__1SSV_20160913T004007_20160913T004031_013027_014A32_33CE
S1A_IW_SLC__1SDV_20160925T004003_20160925T004030_013202_01500A_C461
S1A_IW_SLC__1SDV_20161019T004003_20161019T004030_013552_015B1A_AD25
S1A_IW_SLC__1SSV_20161031T004007_20161031T004032_013727_016071_389C
S1A_IW_SLC__1SDV_20161112T004003_20161112T004030_013902_016605_5C70
S1A_IW_SLC__1SDV_20161206T004002_20161206T004029_014252_0170CD_AE2F
S1A_IW_SLC__1SSV_20161218T004006_20161218T004031_014427_017650_F25C

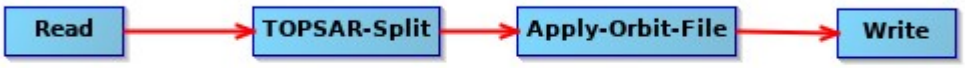
For this tutorial, it is recommended to download these files using the ASF Vertex portal. You can bulk-download a set of selected products using their auto-generated Python download script.



Visualization of all 9 products opened in SNAP, showing the overlay.

All the processing in SNAP will be done via the **gpt** command-line interface. For a full tutorial on this, please follow the SNAP Command line tutorial available online.

Let's begin our processing chain by getting the swath that we need, and correcting the orbit of all 9 products. Mexico City is located within the IW1 swath of all the images. The easiest way to generate an XML processing graph is to use SNAP desktop to create it in the graph builder. Your graph will look like this:



Save this and open it in your preferred text editor. We're going to want to replace a few things so that within a command-line interface, we can run this in batch on all of our products.

The first <operator> tag should be the Read operator. We're going to want to go to the <parameters > section and modify the <file> path provided to be a variable.
We're going to set this to be ${inputFile}.

The second modification we're going to want to do is to modify our Write operator <file> parameter to be a variable as well. We're going to replace that file path to be ${outputFile}.

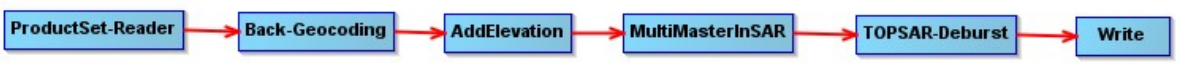
In Linux, looping through files is fairly straightforward using the BASH shell. Navigate to the folder with your products, and we will construct the following bash loop:

```
$ mkdir iw1; for f in *.zip; do gpt myGraph.xml -PinputFile=$f -PoutputFile="iw1/$f"; done
```

This will loop through and apply the graph processing to all the products in our data folder, and output the orbit corrected, IW1 swath images into our folder titled iw1.

The next thing we'll need to do is to construct a second graph XML file that is going to create our co-registered stack, and use a new operator in SNAP 8 to create a stack containing multiple master/slave pairs.

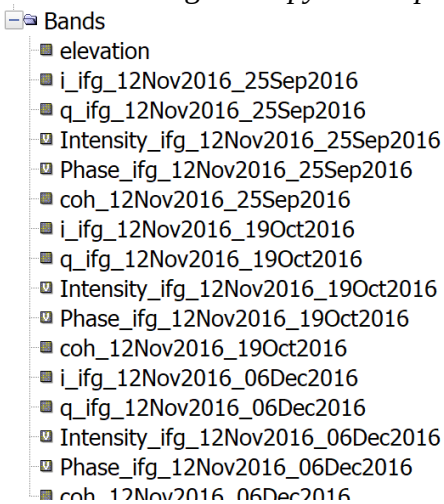
We'll create the following graph, populating the ProductSet-Reader operator with all the products just generated within the **iw1** folder, and modifying the sliders within the MultiMasterInSAR (Referred to in the GUI as Multi Reference InSAR) operator to increase the number of pairs in our interferogram stack. After the MultiMasterInSAR operator, we must then run a TOPSAR-Deburst to remove burst lines from our Sentinel-1 interferograms.



Let's save this XML graph as *pyratePrepMMIFG.xml*. We can now run it on the command line as *pyratePrepMMIFG.xml* and then execute it as such:

```
$ gpt pyratePrepMMIFG.xml
```

The resulting product will be a debursted multi-master InSAR interferogram stack. At this point, it is recommended that you subset the stack. This can be done either with a new graph, or adding a Subset operator after TOPSAR Deburst before writing in the *pyratePrepMMIFG.xml* graph.



Some of the bands that will be in your unwrapped band stack.

With our now subsetted and debursted InSAR stack, we have wrapped phase bands. PyRate requires unwrapped phase, so we must use snaphu to unwrap. This is a freely available tool on Linux, available either via compiling from source, or in some package managers. In Ubuntu, it is available through the command

```
$ sudo apt install snaphu
```

Let's create a new graph as follows:

The manual for snaphu goes over the parameters for phase band unwrapping in more detail, but the defaults that SNAP presents you with will adequately unwrap your bands. The only things you may wish to alter to tailor to your system is the number of processors – if you are running on a machine wit an 8 core CPU, you may wish to bump that number up from the default of 4, to 6 or 7, leaving one core free if you are planning on using the machine while it processes your data. If you are on a less powerful machine with not many cores, consider dropping this number down to 2 or 3.

Add a target folder for where you wish to save your snaphu configuration files and inputs to the snaphu tool, and save the graph as *snaphuExport.xml*.

You can then run the graph using

```
$ gpt snaphuExport.xml
```

and the folder you targeted will populate with a number of files. A recent update to SNAP allows the snaphu export tool to handle stacks with multiple phase bands, creating multiple snaphu configuration input files. These files end in *snaphu.conf* preceded by the phase band name as identification.

If you open one of these snaphu.conf files, you will see the following information:

```
1  # CONFIG FOR SNAPHU
2  # ----------------------------------------------------------------
3  # Created by SNAP software on: 16:54:52 28/11/2021
4  #
5  # Command to call snaphu:
6  #
7  #       snaphu -f Phase_ifg_12Nov2016_06Dec2016snaphu.conf Phase_ifg_12Nov2016_06Dec2016.snaphu.img 1775
8
9  ########################
10 # Unwrapping parameters #
11 ########################
```

The line we want to be able to extract from this is line 7. Since this is a Linux utility, we can simply use some bash and regex to extract this line and run from the command line.

First, let's use awk to get the line, including that pound sign and padding:

```
$ awk '{if(NR==7) print $0}' Phase_ifg_12Nov2016_06Dec2016snaphu.conf
```

This will return the following:

```
#       snaphu -f Phase_ifg_12Nov2016_06Dec2016snaphu.conf
Phase_ifg_12Nov2016_06Dec2016.snaphu.img 1775
```

To remove the first 8 characters, we can pipe that text into the UNIX cut utility as such:

```
$ awk '{if(NR==7) print $0}' Phase_ifg_12Nov2016_06Dec2016snaphu.conf | cut -c9-100000000
```

This will now return the following:

```
snaphu -f Phase_ifg_12Nov2016_06Dec2016snaphu.conf Phase_ifg_12Nov2016_06Dec2016.snaphu.img
1775
```

Instead of just printing this out to the console, we should be writing it to a bash file that can be executed after. This can be done easily as follows:

```
$ awk '{if(NR==7) print $0}' Phase_ifg_12Nov2016_06Dec2016snaphu.conf | cut -c9-100000000 >> run.sh
```

We can now use a bash loop to loop over this and get all the commands as such into one runnable file:

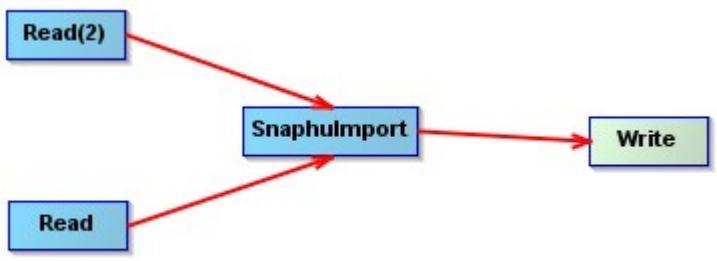
```
$ for a in *snaphu.conf; do awk '{if(NR==7) print $0}' $a | cut -c9-10000000 >> run.sh; done
```

Simply make this script executable with chmod and then you can unwrap all the header bands in one go.

```
$ chmod 700 run.sh; ./run.sh
```

With all the phase bands unwrapped now, it is time to import them back into your product.

Create a new graph as follows:



Set the Read operator to be the wrapped interferogram stack (created from *pyratePrepMMIFG.xml* or from a subsequent subset graph). Be sure that the “Do not save wrapped interferogram” checkbox remains unchecked in SnaphuImport.

Save this graph as *snaphuImport.xml* and open this up in a text editor. Replace the file path in the file tag for the Read(2) operator to be ${unwImg}. Replace the file path in the file tag for the Read operator to be ${stack}. Replace the file path in the file tag for the Write operator to be ${outFile}. Save this file, and now from the command line, we can start programatically adding our unwrapped phase bands to our product like so:

```
$ gpt snaphuImport.xml -PunwImg=data/snaphu/UnwPhase_ifg_12Nov2016_06Dec2016.snaphu.img
-Pstack=data/stack/subset_0_of_S1A_IW_SLC__1SDV_20161112T004003_20161112T004030_013902_016605_5C70.dim
-PoutFile=data/stack/s
data/stack/subset_0_of_S1A_IW_SLC__1SDV_20161112T004003_20161112T004030_013902_016605_5C70.dim
```

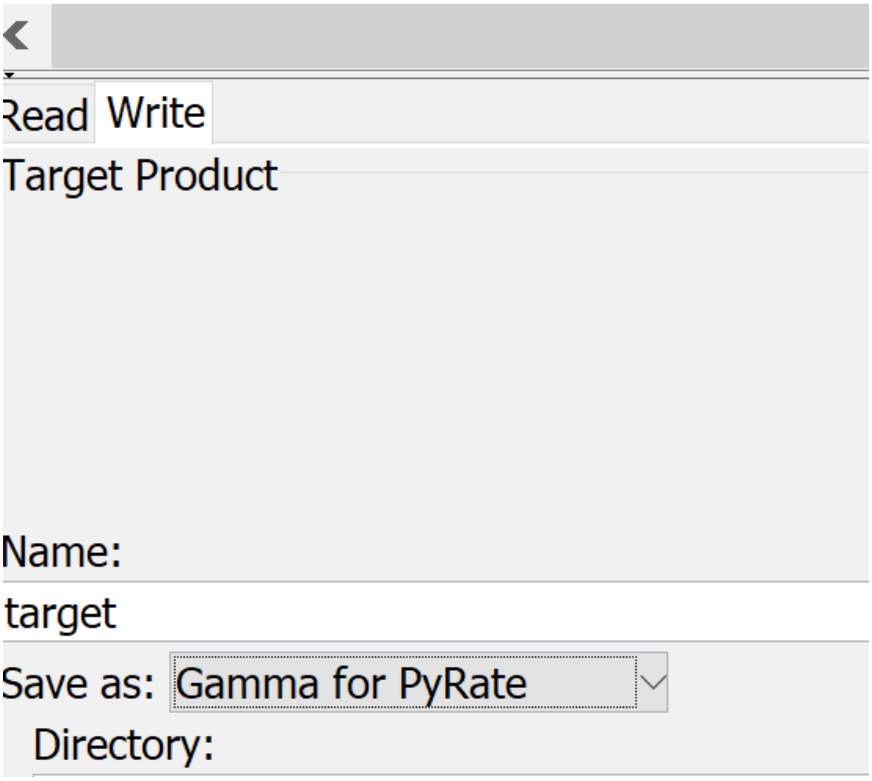
We can then run this as an iterative loop, as we did earlier in this tutorial.

```
$ export stackProduct=’
data/stack/subset_0_of_S1A_IW_SLC__1SDV_20161112T004003_20161112T004030_013902_016605_5C70.d
im’
```

```
$ for a in UnwPhase_ifg*; do gpt snaphuImport.xmp -PunwImg=$a -Pstack=$stackProduct -
PoutFile=$stackProduct; done
```

After running this script, you will have all unwrapped phase bands added to your stack product.

For the final step, we must export this stack into the Gamma for PyRate format. Simply create a read-write graph and specify Gamma for PyRate as an output format, and choose a folder for the data to go.



Configuring PyRate

Now that we have our data prepared for InSAR processing with PyRate, we must first set up some of the input parameters that PyRate needs in order to process our interferograms. PyRate provides sample configuration files in its github repository https://raw.githubusercontent.com/GeoscienceAustralia/PyRate/master/input_parameters.conf .

We will take the one provided and modify it to suit our needs.

First, PyRate requires a file containing all paths of .rslc raw image files. It should look similar to this small portion:

/home/alex/WORK/data/InSAR/ifgs_pyrate/gamma/
Unw_Phase_ifg_IW1_VV_13Sep2016_25Sep2016_20160913-20160925.rslc

/home/alex/WORK/data/InSAR/ifgs_pyrate/gamma/
Unw_Phase_ifg_IW1_VV_25Sep2016_19Oct2016_20160925-20161019.rslc

/home/alex/WORK/data/InSAR/ifgs_pyrate/gamma/
Unw_Phase_ifg_IW1_VV_19Oct2016_31Oct2016_20161019-20161031.rslc

Save this as a plaintext file and provide it as path for the ifgfilelist argument. Be sure to have the files in chronological order in the text document.

Additionally, create a plaintext file that contains a list of your coherence files (coh_ files) in chronological order, and provide that as a path for the cohfilelist parameter.

Set obsdir, and slcFileDir to be the same location (eg. /home/alex/WORK/data/InSAR/ifgs_pyrate/gamma)

demHeaderFile and demfile to be pointing to the location of the .rslc.par and .rslc files respectively of your elevation file you generated, and outdir to be where you want PyRate to store the output and intermediate files.

Be sure to set the nsig argument near the bottom of the file to be less than the number of interferograms you have. If you have 10 interferograms, set it to be 7 or 8.

You will want to set parallel to be 1 and processes as many cores as you can give to PyRate to decrease processing time when possible (e.g. with an 8 core cpu you would want to give 6 cores to PyRate).

Set ifgxfirst, ifgxlast, ifyfirst, ifylast to be the bounds of the image. For this area, a successful run has

been:

ifgxfirst: -99.0927229

ifgxlast: -98.9016766

ifgyfirst: 19.5561865

ifgylast: 19.3623714

Set maxsig to be slightly higher at 5 to allow for not rejecting slight standard deviations.

With this all set up, we can now run PyRate on our input datasets. Pyrate, as of version 0.6.0 (latest version at the time of writing this tutorial) is run with the four commands:

pyrate conv2tif -f input_parameters.conf

pyrate prepifg -f input_parameters.conf

pyrate correct -f input_parameters.conf

pyrate merge -f input_parameters.conf

More information about the PyRate workflow can be found on

https://geoscienceaustralia.github.io/PyRate/