1. Exploring the `insarApp.py` processing option space

`insarApp.py` represents the simplest kind of InSAR processing workflow, that of taking two images acquired from nearly the same vantage point in orbit but at different times and creating an interferogram in geocoded coordinates that represents any motion on the ground that may have occurred between these times. In subsequent labs, we will see more sophisticated processing of a time series of data allowing us to track these changes over time. First, however, we will illustrate some of the flexibility built into the insarApp.py workflow. These flexibilities are built into the framework, and relate to configurable parameters of individual processing "components," so these would be applicable to other workflows built from these components.

There are multiple ways to control the workflow. One way is to alter one of the configurable parameters in the input xml file that controls `insarApp.py`. The list of configurable parameters can be seen by using the `--help` command line option, which also prints a usage statement. The other way is to manipulate the steps of the work using the `--steps` command line option. `--steps` allows the user to start the processing from a particular point in the workflow and end it at another location. Clearly the processing cannot be started beyond a point where a previous processing run completed (for a fresh data set, you must start at the beginning!), but `--steps` allows the user to run individual workflow components one at a time or in sequence allowing the alteration of input parameters for each workflow component.

In this lab, we will exercise the `--steps` option to prepare the raw data for a data set, then alter a processing parameter to reduce the total size of the data to be processed. Once we then process all the way through, we will alter another processing parameter to allow phase unwrapping of the result. With these simple examples, we will convey the main ideas of flow control, and you will then be prepared to experiment on your own with other parameters and steps options.

2. Preparing the raw data

Let's first look at what is possible with `--steps`.

```
> insarApp.py --steps --help
2013-09-18 00:52:07,992 - isce.insar - INFO - ISCE VERSION = 1.0.0,
RELEASE_SVN_REVISION = 739,RELEASE_DATE = 20120814, CURRENT_SVN_REVISION = 1154M
ISCE VERSION = 1.0.0, RELEASE_SVN_REVISION = 739,RELEASE_DATE = 20120814,
CURRENT_SVN_REVISION = 1154M

       Insar Application:
       Implements InSAR processing flow for a pair of scenes from
       sensor raw data to geocoded, flattened interferograms.

A description of the individual steps can be found in the README file
and also in the ISCE.pdf document

Use command line options '--start=<step>', '--end=<step>', --dostep=<step> to choose
the step names from the following list:

self.step_list =  ['startup', 'preprocess', 'verifyDEM', 'pulsetiming',
'estimateHeights', 'mocompath', 'orbit2sch', 'updatepreprocinfo', 'formslc',
'offsetprf', 'outliers1', 'prepareresamps', 'resamp', 'resamp_image',
'mocompbaseline', 'settopoint1', 'topo', 'shadecpx2rg', 'rgoffset', 'rg_outliers2',
'resamp_only', 'settopoint2', 'correct', 'coherence', 'filter', 'unwrap', 'geocode',
'endup']

If --start is missing, then processing starts at the first step.
If --end is missing, then processing ends at the last step.
If --dostep is used, then only the named step is processed.
```

Note that each of the names in the list called `self.step_list` are workflow component names, each carrying out a specific function briefly described in the table below (see ISCE.pdf for a description of each component).

Step name	**Short functional description**
startup	Initialization of python objects for interferogram processing.
preprocess	Extract raw radar echoes from original sensor files and store them in an ISCE compatible format. Populate metadata fields for use in processing.
verifyDEM	Check if the user has provided a DEM. If not download a DEM from the SRTM archive.
pulsetiming	Determine antenna position for every raw echo line by interpolating the

	state vectors.
estimateHeights	Estimates the average heights for each of the SAR acquisitions from the interpolated state vectors.
mocomppath	Determines the reference mocomp orbit for focusing the SAR acquisitions.
orbit2sch	Transforms the state vector information for both SAR acquisitions into the SCH coordinate system, while accounting for the reference mocomp orbit.
updatepreprocinfo	Updates the parameters with common values for SAR focusing.
formslc	Focuses raw radar echoes into a single-look complex image.
offsetprf	Estimates offsets between the master and slave image while accounting for slight differences in the PRFs.
outliers1	Culls the offset field by removing noisy offset estimates.
prepareresamps	Setup the resampling routine for interferogram generation.
resamp	Resample the slave SLC and cross multiply with the master SLC, to create an interferogram.
resamp_image	Dump the offset field that was used for resampling as images.
mocompbaseline	Estimate the baseline to be used for topography removal, on a line by line basis.
settopoint1	Sets the file names for the output of the topo module. To be read as <set_topo_int1>.
topo	Estimate the DEM in radar coordinates using the master orbit information.
shadecpx2rg	Simulate an amplitude image from the estimated DEM in radar coordinates.
rgoffset	Determine offset field between interferogram and simulated amplitude.
rg_outliers2	Cull the offsetfield to remove noise offset estimates.
resamp_only	Resample the interferogram to match the DEM. In this paradigm, we trust the orbits and the geometry more than the focusing modules.
settopoint2	Sets the file names for the correct module. <To be read as set_topo_int2>.

correct	Remove the topography component of phase using the outputs of topo module and mocompbaseline.
coherence	Estimate the coherence from the topo-corrected interferogram.
filter	Filter the corrected interferogram using an adaptive filter. Also estimate the coherence for filtered interferogram using the phase standard deviation.
unwrap	Unwrap the interferogram using method of choice.
geocode	Geocode the requested set of outputs.
endup	Clean up and close files as needed.

For our purposes, we simply want to prepare the data for image formation. `formSLC` is the image formation component, so we want to run the following workflow:

```
> cd /home/ubuntu/data/lab3/alos/20070612_20090802
> insarApp.py --steps --end="updatepreprocinfo" insar_input.xml
```

This will process from raw data all the way to where the inputs are established for running form SLC. `--steps` creates a `PICKLE` directory which stores all the information needed to restart the process. This directory is referenced when the next run with `--steps` is executed. If you examine the screen output at the end of the above command, you should see:

```
2013-09-18 00:42:14,352 - isce.insar.runFdMocomp - INFO - Updated Doppler
Centroid: 0.0419756826644
Dumping the application's pickle object _insar to file
PICKLE/updatepreprocinfo
The remaining steps are (in order):  ['formslc', 'offsetprf', 'outliers1',
'prepareresamps', 'resamp', 'resamp_image', 'mocompbaseline', 'settopoint1',
'topo', 'shadecpx2rg', 'rgoffset', 'rg_outliers2', 'resamp_only',
'settopoint2', 'correct', 'coherence', 'filter', 'unwrap', 'geocode', 'endup']
```

with the appropriate time stamp for your run. This shows you that you processed successfully up to the step before `formSLC`. Now we will see how to control the processing to process only a portion of the available data.

3. Altering the processing parameters

The best way to see the possible options in a nutshell is with the help function of `insarApp.py`, as follows:

```
> insarApp.py --help
2013-09-18 00:49:20,766 - isce.insar - INFO - ISCE VERSION = 1.0.0,
RELEASE_SVN_REVISION = 739,RELEASE_DATE = 20120814, CURRENT_SVN_REVISION =
1154M
ISCE VERSION = 1.0.0, RELEASE_SVN_REVISION = 739,RELEASE_DATE = 20120814,
CURRENT_SVN_REVISION = 1154M

      Insar Application:
      Implements InSAR processing flow for a pair of scenes from
      sensor raw data to geocoded, flattened interferograms.

The currently supported sensors are:  ['ALOS', 'GENERIC', 'RADARSAT2',
'ENVISAT', 'COSMO_SKYMED_SLC', 'COSMO_SKYMED', 'RADARSAT1', 'ERS', 'TERRASARX',
'JERS']
Usages:
insarApp.py <input-file.xml>
insarApp.py --steps
insarApp.py --help
insarApp.py --help --steps

See the table of configurable parameters listed in the table
below for alist of parameters that may be specified in the
input file.  See example input xml files in the isce 'examples'
directory.  Read about the input file in the ISCE.pdf document.

The user configurable inputs are given in the following table.
Those inputs that are of type 'component' are also listed in
table of facilities below.
To configure these parameters, enter the desired value in the
input file using a property tag with public_name = to the name
given the table
```

name	type	mandatory	doc
sensor name	str	mandatory	Sensor name
slc offset method	str	optional	SLC offset estimation method name. Use value=ampcor to run ampcor

Master	component	mandatory	Master raw data component
slc offsetter	component	optional	SLC offset estimator.
peg longitude (deg)	float	optional	Peg Longitude in degrees
demFilename	str	optional	Filename of the DEM init file
posting	int	optional	posting for interferogram
Slave	component	mandatory	Slave raw data component
Form SLC	component	optional	SLC formation module
use_dop	float	optional	Choose whether to use master, slave, or average Doppler for processing.
range looks	float	optional	Number of range looks to use in resamp
doppler method	str	optional	Doppler calculation method.Choices: 'useDOPIQ', 'useCalcDop', 'useDoppler'.
Run Unwrapper	component	optional	Unwrapping module
useHighResolutionDemOnly	int	optional	If True and a dem is not specified in input, it will only download the SRTM highest resolution dem if it is available and fill the missing portion with null values (typically -32767).
geoPosting	float	optional	Output posting for geocoded images in degrees (latitude = longitude)
peg latitude (deg)	float	optional	Peg Latitude in degrees
offset search window size	int	optional	Search window size used in offsetprf and rgoffset.
unwrap	bool	optional	True if unwrapping is desired. To be unsed in combination with UNWRAPPER_NAME.
geocode list	tuple	optional	List of products to geocode.
unwrapper name	str	optional	Unwrapping method to use. To be used in combination with UNWRAP.
azimuth looks	float	optional	Number of azimuth looks to use in resamp
correlation_method	str	optional	Select coherence estimation method: cchz=cchz_wave phase_gradient=phase gradient
Slave Doppler	component	optional	Master Doppler calculation method
Dem	component	optional	Dem Image configurable component. Do not include this in the input file and an SRTM

			Dem will be downloaded for you.
peg radius (m)	float	optional	Peg Radius of Curvature in meters
peg heading (deg)	float	optional	Peg Heading in degrees
gross range offset	int	optional	Override the value of the gross range offset for offsetestimation prior to interferogram formation
azimuth patch size	int	optional	Size of overlap/save patch size for formslc
pickle dump directory	str	optional	If steps is used, the directory in which to store pickle objects.
gross azimuth offset	int	optional	Override the value of the gross azimuth offset for offset estimation prior to interferogram formation
Culling Sequence	tuple	optional	TBD
patch valid pulses	int	optional	Size of overlap/save save region for formslc
number of patches	int	optional	How many patches to process of all available patches
Master Doppler	component	optional	Master Doppler calculation method
pickle load directory	str	optional	If steps is used, the directory from which to retrieve pickle objects

The help list above illustrates a number of parameters that control the workflow. Rather than describing each one in detail, we will focus on just a few to illustrate how you would go about changing them, and showing the effect of changing them on the processing. Let’s start with a simple one: “number of patches”. This input parameter would be specified in the .xml input file that is read by insarApp.py. The processing of radar imagery is done in chunks, where several thousand lines of raw data are read, and processed to form a sub-image, then the next chunk is read in with some overlap to create the next subimage, and so forth, until the entire image is processed. These sub-images are then put together to form the complete image. Each chunk is traditionally called a “patch”, and the user is allowed to either take the default, which is to process the entire image, or specify the number of patches they wish to process.

First examine our familiar `insarApp.xml` (we used this in Lab 3.1):

```
> more insarApp.xml
<insarApp>
```

```
<component name="insarApp">
    <property name="sensor name">
      <value>ALOS</value>
    </property>
    <component name="Master">
      <catalog>Master.xml</catalog>
    </component>
    <component name="Slave">
      <catalog>Slave.xml</catalog>
    </component>
</component>
</insarApp>
```

Note that most parameters are not specified. Defaults are taken based on the sensor data. Now let's set the number of patches parameter here. Adding the lines

```
    <property name="number of patches">
      <value>1</value>
    </property>
```

will do the trick. We have done this in a new file `insarApp_1patch.xml`, which you can list to verify:

```
> more insarApp_1patch.xml
<insarApp>
<component name="insarApp">
    <property name="sensor name">
      <value>ALOS</value>
    </property>
    <property name="number of patches">
      <value>1</value>
    </property>
    <component name="Master">
      <catalog>Master.xml</catalog>
    </component>
    <component name="Slave">
      <catalog>Slave.xml</catalog>
    </component>
</component>
</insarApp>
```

Now by issuing the following command, we can pick up from where we left off with, specifying that we only want to process one patch of data.

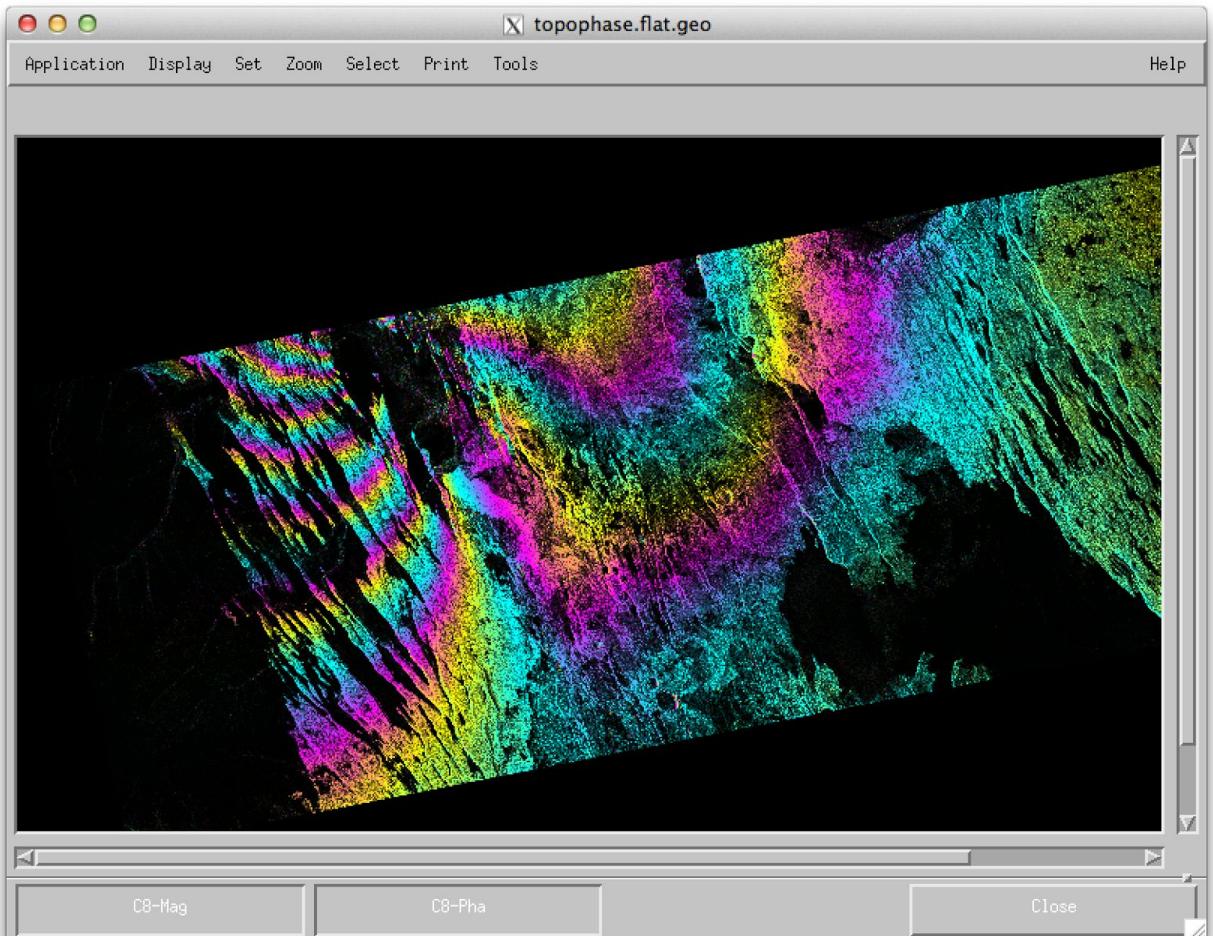
```
> insarApp.py --steps --start='formslc' insarApp_1patch.xml
```

(When `--start` or `--end` is specified, the `--steps` switch is technically not needed.) At the end of this process, you will have completed the full processing run. Now we can play with another processing option: unwrapping.

4. Turning on unwrapping

Now that we've completed the full processing run (for a 1-patch subset of the full image to speed up the demo), we can see that the output contains a geocoded interferogram, but it is not unwrapped.

```
> mdx.py topophase.flat.geo
```



We can see some nice deformation fringes on the top left of the image. To be able to exploit this signature in further analysis such as stack processing, we need to unwrap this image. Having run `--steps` previously, we can restart the process at the unwrapping stage by modifying the `insarApp_1patch.xml` file to include the unwrap option. This change has been made in the file named `insarApp_1patch_unwrap.xml`. The additional lines added to the `insarApp_1patch_unwrap.xml` file for unwrapping are the following (we are using the unwrapper called icu, which is one of a few options available in isce):

```
<property name="unwrap">
```

```
        <value>True</value>
    </property>
    <property name="unwrapper name">
        <value>icu</value>
    </property>
```

We can now run just the unwrapper to the end of the workflow.

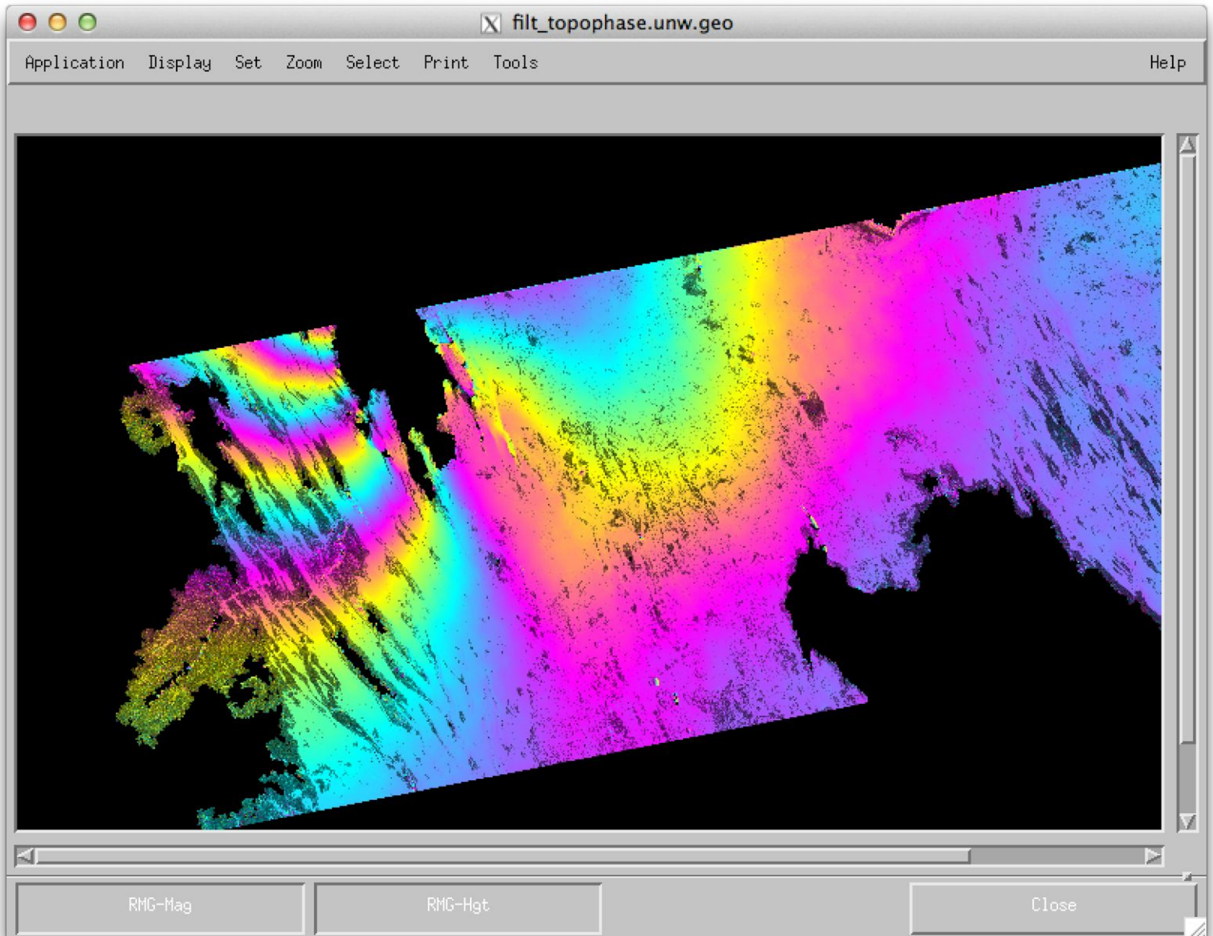
```
> insarApp.py --steps --start='unwrap' insarApp_1patch_unwrap.xml
```

When this is finished, we can see additional files in the directory.

```
> ls -1tr
.
.
.
insarApp_1patch_unwrap.xml
insar.log
filt_topophase.unw
filt_topophase.unw.xml
geo.log
topophase.cor.geo
topophase.cor.geo.xml
filt_topophase.flat.geo
filt_topophase.flat.geo.xml
topophase.flat.geo
topophase.flat.geo.xml
phsig.cor.geo.xml
phsig.cor.geo
los.geo
los.geo.xml
resampOnlyImage.amp.geo
resampOnlyImage.amp.geo.xml
dem.crop
filt_topophase.unw.geo
filt_topophase.unw.geo.xml
catalog
isce.log
insarProc.xml
```

Note that in addition to `topophase.flat.geo`, there are a number of other files. The geocoded unwrapped data is in `filt_topophase.unw.geo`. Using `mdx.py` to display, then changing the scale to 4*PI color wrap (right-click on Pha button and set wrap to 12.56), you should see:

```
> mdx.py filt_topophase.unw.geo
```



In comparing to the image above, you note that the phase colors are much smoother (due to a filtering operation applied before unwrapping), and the phase is no longer subject to a restriction to the interval from 0 to 2 PI. The black regions are places the unwrapper failed to unwrap, either due to no data on the periphery, or low correlation.

You have now successfully completed the insarApp.py workflow exploring a number of options, including unwrapping. If you are interested in seeing this entire scene rather than the 1-patch subset, feel free to start from the beginning deleting the “number of patches” attribute in the .xml file.

In a later lab, you will apply your skills to preparing a stack for time-series processing.