1. MDX interferogram visualization

In a previous lab we saw a preliminary use of the display tool mdx. In this segment, we will explore the features of mdx in greater detail to highlight its utility.

mdx stands for “multi-dataset display using X11” and was written as a special purpose display tool to visualize multiple large data sets associated with radar data processing. mdx takes into account the fact that the data sets are typically very large, often multiple gigabytes, and therefore impractical to read into memory all at once. Therefore, in its design, it keeps track of the coordinates of the display that the user controls with the GUI motif-based user interface, and only reads, scales, maps to color, and displays those data within the window. When the user scrolls the window to another location, mdx keeps track of what is old and what is new in the new position and recalculates and redisplays only the new data. In this way it can be very fast to explore very large data sets. On zooming out, mdx subsamples the data file, so even zooming out over large areas allows for relatively fast panning. This feature, which one would expect to find in many visualization tools, is unique to mdx, and makes it a very powerful tool for working with ISCE data.

2. Command line options

mdx has a large array of command-line arguments to control the data sets that are displayed and how they are displayed. Typing “mdx” at the command prompt with no command arguments returns a help message with a usage message and all the possible options. We will describe the most often used of these options later, but first we will describe a python script called “mdx.py” that allows users quick and easy use of mdx for commonly displayed data types. Typing “mdx.py” at the command prompt with no command arguments returns a help message describing its usage in full.

```
> mdx.py
  Usage:

  mdx.py   filename [-wrap wrap] ... [-z zoom -kml output.kml]

  where

  mdx.py  : displays one or more data files simultaneously by
            specifying their names as input. The maximum number,
            of images that can be displayed depends on the machine
            architecture and  mdx limits. If displayed (no -kml flag)
            the images don't need to have the same extension, but need
            to have same width.

  filename: input file containing the image metadata.
            Metadata files must be of format filename.{xml,rsc}
            and must be  present in the same directory as filename.
            Different formats (xml,rsc) can be mixed.

  -wrap   : sets display scaling to wrap mode with a modules of Pi.
            It must follow the filename to which the wrap is applied.

  ...  : the command can be repeated for different images.

  -z   : zoom factor (+ or -) to apply to all layers. It's optional
            and can appear anywhere in the command sequence and must
            appear only once.

  -kml : only for geocoded images it creates a kml file with all the
            input images overlaid. Each layer can be turn on or off in
            Google Earth. It's optional and can appear anywhere in the
            command sequence and must appear only once. The images don't
            need to be co-registered.

  Examples:
  mdx.py 01_02.int                              # Standard way to run mdx.py
  mdx.py 03_04.int 05_06.int -z -8              # Display two images; zoom out by 8
```

```
  mdx.py 03_04.geo -z 8 05_06.geo -kml fileout.kml # Create a kml file named
fileout.kml with two
                                                   # layers, one per image. Both
images are zoomed in
                                                   # by a factor of 8
  mdx.py 03_04.int 05_06.int -wrap 6.28            # Display two images. Wrap the
second modulo 2Pi


  or

  mdx.py -ext

  to see the supported image extensions.
```

3. MDX and file names

The common form of a data filename in ISCE is *prefix.ext* where *ext* is one of a number of standard extensions that describe a particular format or type of data. See Output File Formats discussion in the Datasets tutorial for a complete description of the extensions used in ISCE. Typing “mdx.py -ext” at the command prompt with no other command arguments displays a list of all the extensions that mdx.py understands, including a few heritage extensions not used by ISCE.

```
> mdx.py -ext
version =  1.0.0
Supported extensions:
unw
byt
flg
scor
slc
int
geo
flat
mph
cpx
msk
cor
hgt
hgt_holes
rect
amp
dem
dte
dtm
```

mdx.py takes as arguments a list of filenames, one for each of the images you want to display. Since all ISCE data files are flat files, ie. binary files of data arranged in rows and columns with no headers or other descriptive information, their corresponding xml metadata file is needed to describe the image dimensions. The metadata filename is formed in ISCE by adding “.xml” to the end of the filename. The format of the data is also encoded in the metadata. mdx.py constructs a metadata filename from the data filename provided on the command line, then reads all the descriptive information from the metadata file to determine how to display the data.

4. Layers of data

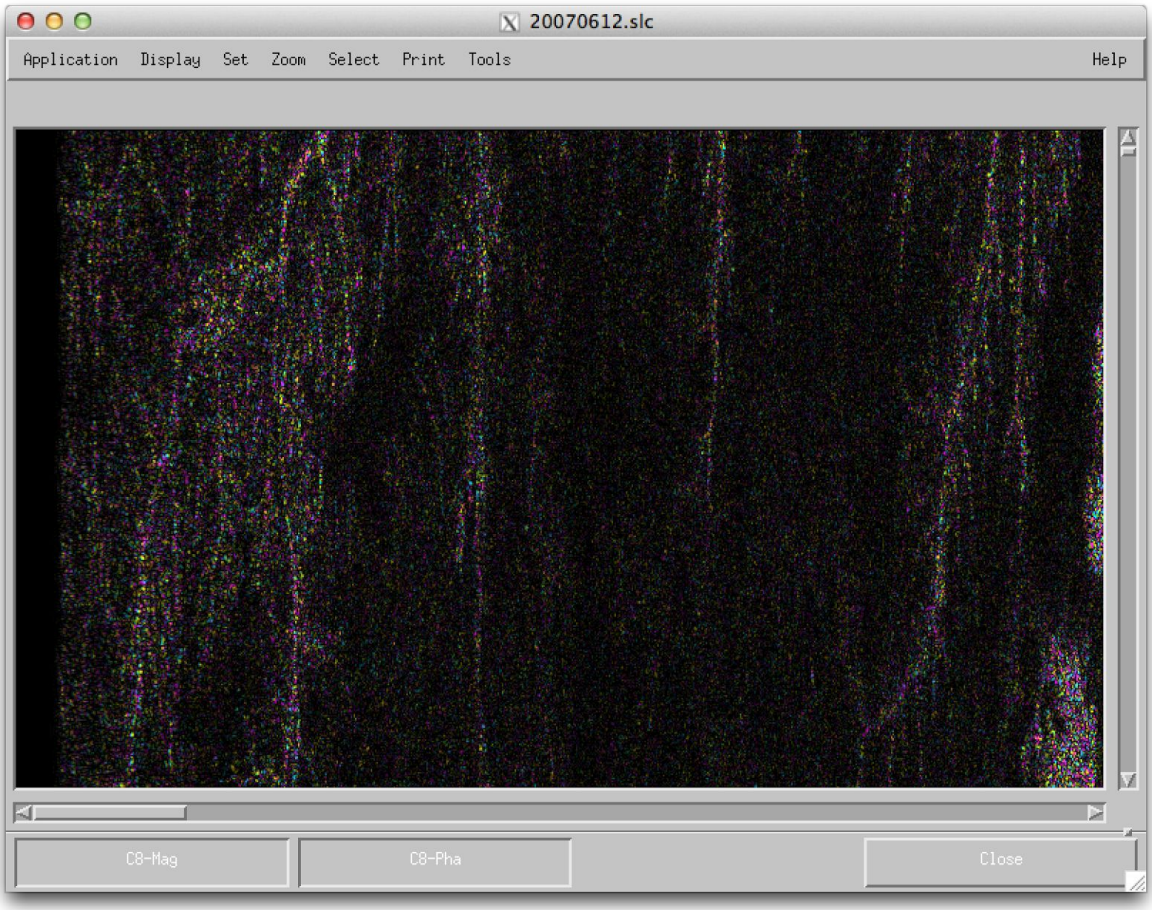
Let's look at a few examples. The output of ISCE after running the insarApp.py application contains a pair of complex images that form the interferogram, typically known as the "master" and "slave," though the name of the file can be arbitrary. In what follows, the master and slave filenames are denoted by the dates of acquisition. You might want to compare these images to see how well they align after motion compensation and image formation.

```
> cd /home/ubuntu/data/sites/lab3_alos/20070612_20090802
```

This site is known as the Afar region in Africa. A599 refers to the Ascending track number and 0230 refers to the frame number. The dates 20070612 and 20090802 are the dates on which acquisitions of this area were made, and we choose to create a directory called `20070612_20090802` as the location to process the data. The input data files are contained in directories `20070612` and `20090802` individually. Let's display one of the processed images: the "single look complex" slc files.

```
> mdx.py 20070612.slc
```

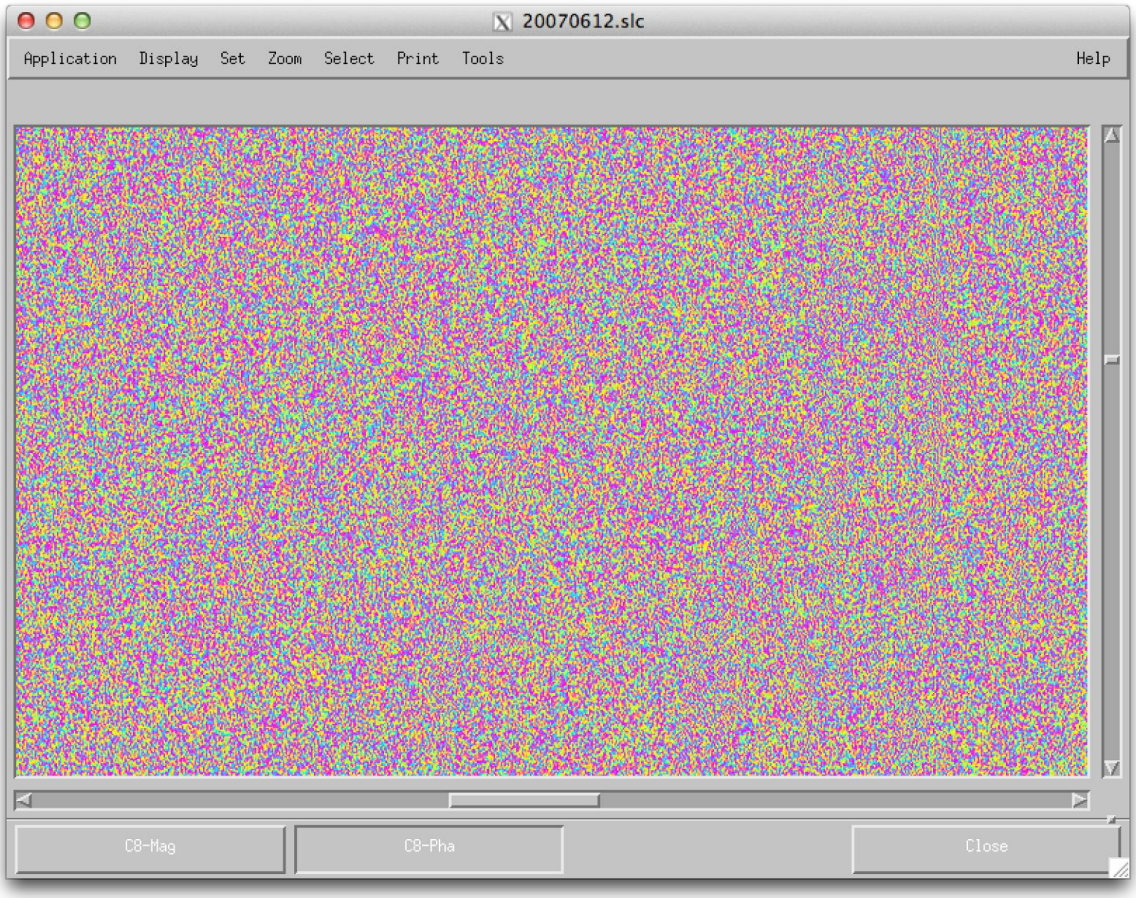
will display this file as 2 layers, each layer with a clickable button to select it.



Note the name associated with these buttons. An `slc` is a complex data type, and the natural way to display each complex pixel is through its magnitude and phase. So for this slc, mdx.py will construct a layer that is the magnitude (`C8-Mag`) and another layer that is the phase (`C8-Pha`). The `C8` designation means that the slc file is stored as a complex 8-byte data type (4-byte real, 4-byte imaginary; mdx understands complex integers as well). For more on complex numbers and data formats, see this document on Number Formats.

Since we specified one slc file as input to mdx.py, there are 2 layers, C8-Mag, C8-Pha. By clicking on either layer, it will select just that layer. Click back and forth on the C8-amp and C8-Pha layers to see that while there is structure in the amp layer associated with the ground reflectivity, the phase layer is completely random. This is because the radar is a coherent system and the phase is representative of the “speckle” pattern created by random collections of scatterers in a resolution element. The phase is a completely random process. The amplitude is also subject to speckle, which is why it is so noisy, but the intrinsic reflectivity of the surface shows through.

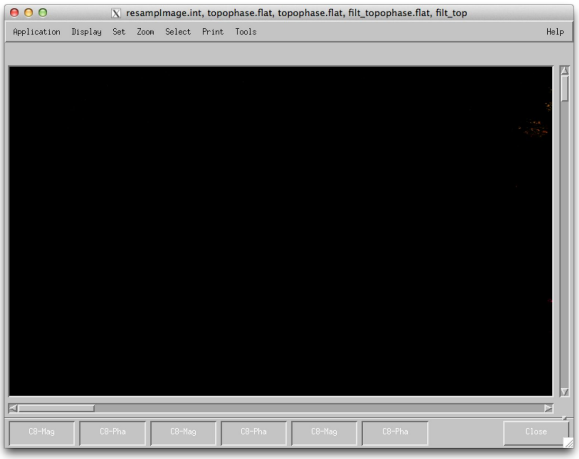
Click on the C8-Pha layer and scroll around using the scroll bars. Try to match the scroll locations as shown below. The phase should still look random!

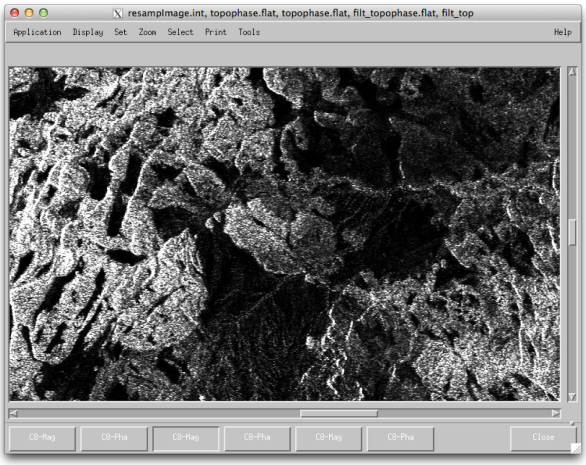


It is hard to imagine looking at this random phase that there is information present in the data. But because the randomness has to do with the spatial variability of the scattering on the ground, if another acquisition is made at a different time from nearly the same vantage point, then the randomness will be essentially the same, so in the difference, i.e. in the interferogram, this randomness cancels and a beautiful interference pattern emerges. Let's examine the various interferograms that ISCE insarApp.py produces.

```
> mdx.py resampImage.int topophase.flat filt_topophase.flat
```

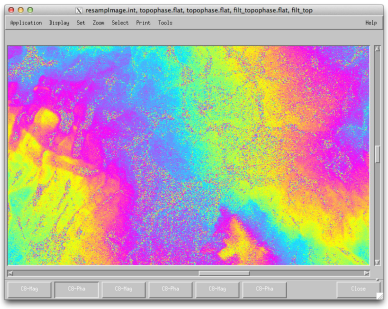
will show the original interferogram, one flattened to remove topography, and another filtered to smooth out the noise. These are also complex data types, so there will be 6 layers - 3 pairs of C8-Mag, C8-Pha layers. Note that the display comes up looking very dark. This is because this corner of the images is indeed low reflectivity water. Also each of the three amplitude layers is multiplied together to merge them for display, so they become even darker.



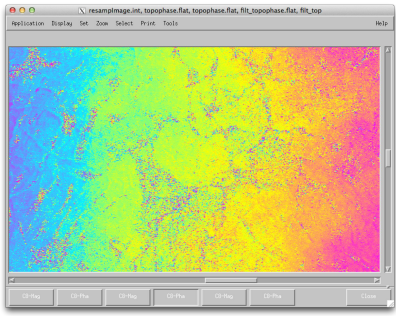


If you click individually on any one amp layer, you will see they are brighter, and if you scroll around to different areas, you'll see much brighter surface features in the image, as shown in the side-by-side comparison above.

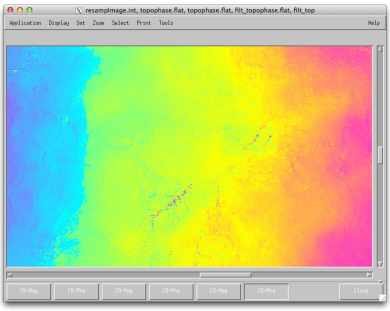
Now click on the three C8-Pha layers to see how they change with different stages of processing. Note that the random phase of the slc file has yielded to a phase difference pattern than is quite well behaved and non-random, through the magic of interferometry.



`resampImage.int - basic interferogram`



`topophase.flat - interferogram with topography removed`

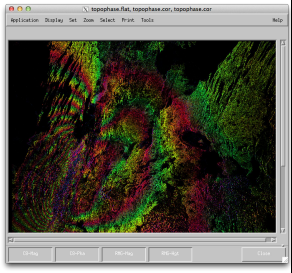
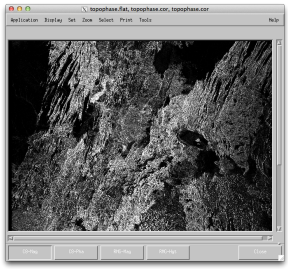
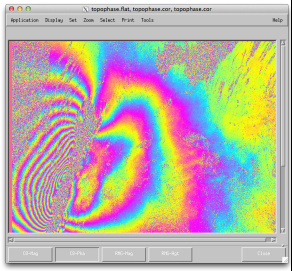
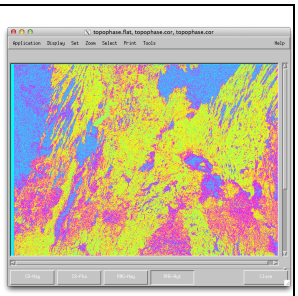


`filt_topophase.flat - filtered version to smooth out the noise`

Note that if you would like to go back to an overlay of data, for example overlaying the amp and pha layers of a particular image, you must click on one layer then "middle click" (for a 3-button mouse) on the other layer. ("middle" click on a 1-button mouse depends on your system; for a mac, often setting the X11 preference to "emulate three button mouse" and holding down the "Option key" while clicking will emulate a middle-click.)

You can also mix data types. You can examine the interferograms and the correlation file to compare the phase noise to the correlation.

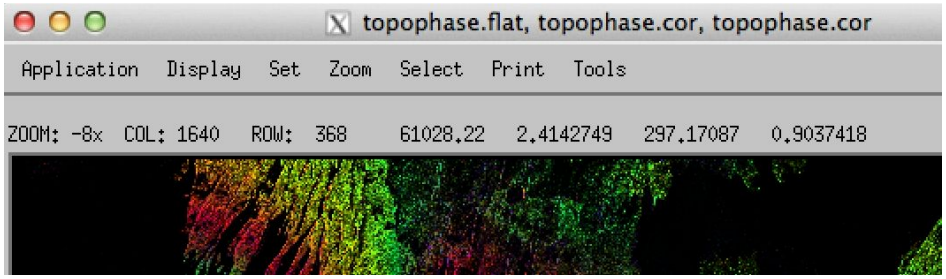
```
> mdx.py -z -8 topophase.flat topophase.cor -wrap 1.2
```

			
merged layers of above command for topophase.flat and topophase.cor	amplitude layer only of topophase.flat	phase layer only of topophase.flat	correlation layer only of topophase.cor, with a color cycle wrap of 0 - 1.2

In this case, mdx.py displays the `C8-Mag` and `C8-Pha` layers of the topophase.flat followed by the `RMG-Mag` displaying again the magnitude kept in the correlation file layer and `RMG-Hgt` displaying the correlation layer. An `RMG` file is a ROI_PAC legacy line-interleaved format file still used in ISCE. Click back and forth on the magnitude layers. They should be completely coregistered. Then click back and forth on the phase and correlation layers. Note the correlation is high when the phase is smoother. The figure above illustrates the various layers. Note also in this example that we specified a zoom factor of 1/8 (“zoom out”) on the command line “`-z -8`”. As shown in the first tutorial, zooming can also be accomplished with the mdx windows.

5. Bonus features

A couple of other features of mdx and mdx.py that you should know about:
1) by clicking on any pixel in the image, the values of all layers at that pixel will be displayed at the top of the display window, above the image but below the menu bars.
2) the menu bars allow the user to interactively control a number of features, including the zoom factor



mdx.py is a convenient way to quickly display files that are produced by ISCE. But the mdx program itself has a considerable amount of command-line flexibility that mdx.py does not try to capture. You can see a glimpse of the syntax expected by mdx by observing the command line for mdx that mdx.py constructs:

```
> mdx.py topophase.flat
Running: mdx topophase.flat -c8 -s 5194
.
.
.
```

By playing with mdx.py and observing the constructed mdx commands, and by reading the usage message for mdx (two pages!), you can get a feel for how to display files with many different characteristics simultaneously and efficiently using the mdx executable command (as opposed to the mdx.py script) directly. But mdx.py should satisfy the needs of most users.