SAR Processing Algorithm

We have seen that the phase history of an azimuth point scatterer forms a chirp signal much like that we used for our range modulation, with a chirp rate that depends on imaging geometry and a centroid that depends on pointing. The centroid represents a "carrier" frequency upon which the chirp is imposed.

Hence the two important quantities for azimuth signal representations are the chirp rate f_R and the Doppler centroid f_{DC}.

$$f_R = \frac{-2v^2}{\lambda r_{DC}}$$

$$f_{DC} = -\frac{2v^2 \epsilon_{DC}}{\lambda r_{DC}} = \frac{2v}{\lambda} \sin\theta_q = \frac{2v}{\lambda} \frac{x_{DC}}{r_{DC}}$$

To process the azimuth information and obtain the finest possible resolution, we simply create a matched filter as in the range case, and correlate.

Azimuth matched filter

The chirp rate and centroid, f_R and f_{DC}, for azimuth correspond to chirp slope and offset in range. As in the range case, we need to know the pulse length and sample rate to construct the reference signal.

Recall from our azimuth resolution discussion that we assumed we would process the illuminated

part of the swath:

$$\text{az beamwidth} = \frac{r\lambda}{\ell}$$

In the case of squinted geometries, we use the range to the Doppler centroid, r_{DC}, as the reference range, recalling that

$$r_{DC}^2 = r_0^2 + x^2$$

where x^2 is the offset along-track of the illuminated point at the antenna boresight from the zero-Doppler line. The time required for a point to traverse the beam is then

$$\tau_{az} = \frac{r_{DC}\lambda}{v\ell}$$

which is the azimuth pulse length. Our signal is sampled at the prf, so the length of the azimuth reference in points is

$$npts_{az} = \tau_{az} \cdot prf = \frac{r_{DC}\lambda\, prf}{v\ell}$$

We can now write down the phase history of the reference function:

$$\phi_{az}(t) = \pi \cdot f_R \cdot t^2 + 2\pi f_{DC} \cdot t \qquad -\frac{\tau_{az}}{2} < t < \frac{\tau_{az}}{2}$$

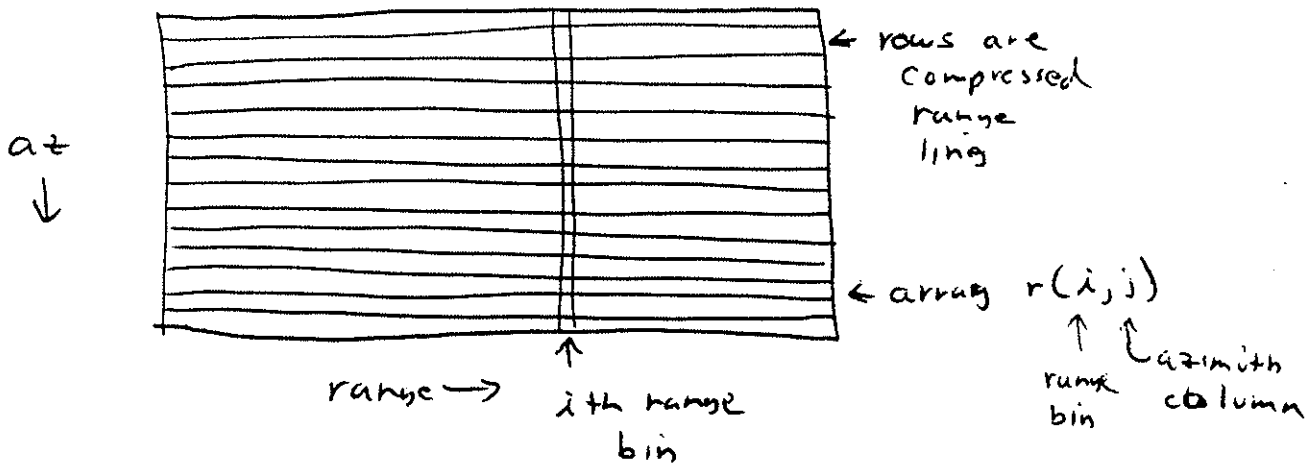
and once again

$$ref(t) = e^{j\,\phi_{az}(t)}$$

In discrete form, our code might look like

```
do i = -npts/2, npts/2
   t = i/prf
   phase = pi*frate**t**2 + 2*pi*fdc*t
   ref(i+npts/2+1) = cexp (cmplx(0., phase))
end do
```

So, to process the following range-compressed data set




```
do bin = 1, n                    ← loop over range bins
   range = r0 + bin * deltar     ← find correct range for bin
   rdc = sqrt(range**2 + (fdc*range*λ/2/v)**2)   ← get rdc
   frate =                       ← freq fdc for this bin
   fdc =
   <create reference function>
```

```
do line = 1, lines
    signal ( line) = r( bin, line)        } copy signal data into 1-D array for processing
end do

call fft ( signal , lines , -1)           ← transform signal
call fft (ref, lines, -1)                 ← transform reference

do line = 1, lines
   signal ( line) = signal (line) * conjg(ref(line))
end do

call fft ( signal , lines , 1)            ← inverse transform product

do line = 1, lines
   r(bin, line) = signal (line)           } copy back to 2-D array
end do

end bin loop
```

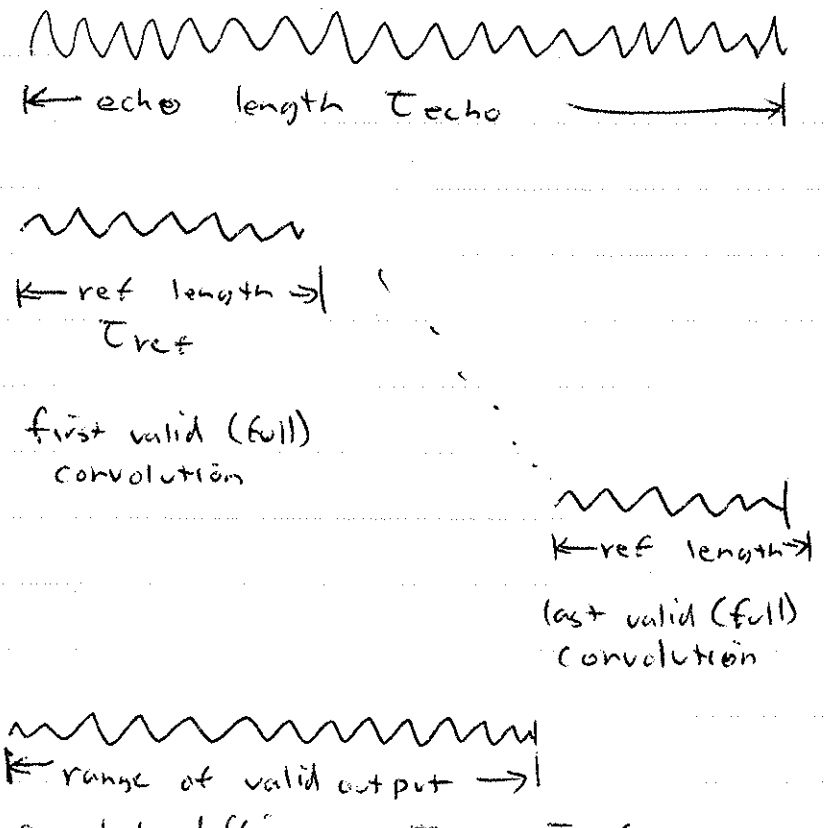
Writing out the $r(i,j)$ array saves the image.

Of course, remember to initialize arrays to zero when necessary within the loops.

Valid data and transform optimization

When processing either range or azimuth by

convolutional methods, the output processed data will usually have a different length than the input data because of "chirp overhead". Consider this convolution (correlation) of an echo signal with a reference function:



In discrete form,

$$n_{valid} = n_{echo} - n_{ref}$$

where n is the number of points in an array.

However, the length of the full convolution, including partially valid points, is the sum of the lengths

$$\tau_{full} = \tau_{echo} + \tau_{ref}$$

$$n_{full} = n_{echo} + n_{ref}$$

Finally, since we are using fft methods, we have the wrap-around due to the cyclic nature of the transform. Thus points repeat every n_{fft} points.

We had an example of this already when we looked at range compressing the ERS signal:

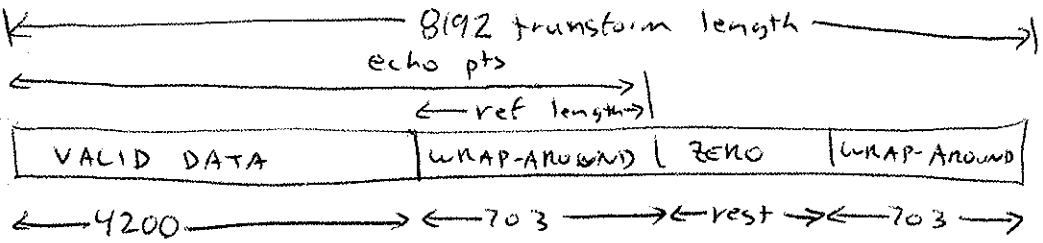
Recall we had 4096 data values in each record, of which 412 were header information. Since two values form each complex sample, the actual number of samples n_{echo} was

$$n_{echo} = (10218 - 412)/2$$
$$= 4903 \text{ points}$$

The ERS chirp was 703 complex samples ($n = \tau \cdot f_s$), so the number of valid points was

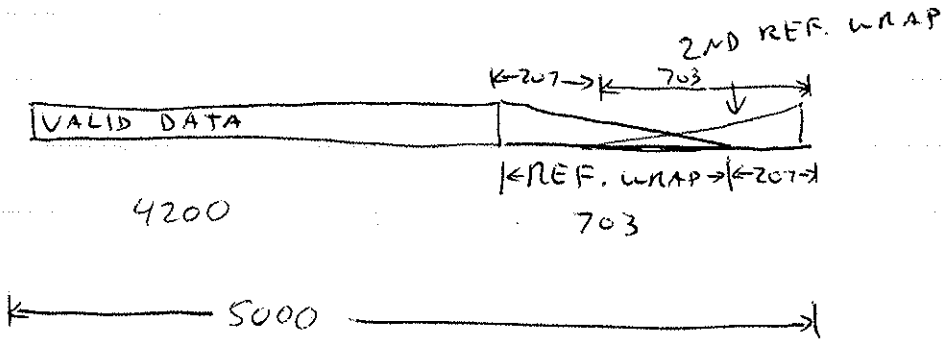
$$n_{valid} = 4903 - 703$$
$$= 4200$$

Our compressed lines then look like



An efficiency trick: Since we throw away all non-valid data, suppose we shorten our transform to 5000

and allow the wraparounds and zeros to overwrite each other. As long as the sum of valid data length plus reference remains less than the transform we are safe. In our example:



Note that this means we need to transform only the length of the original echo, $n_{valid} + n_{ref}$, rounded up to $2^{\sim}$ for most ffts, in most cases.

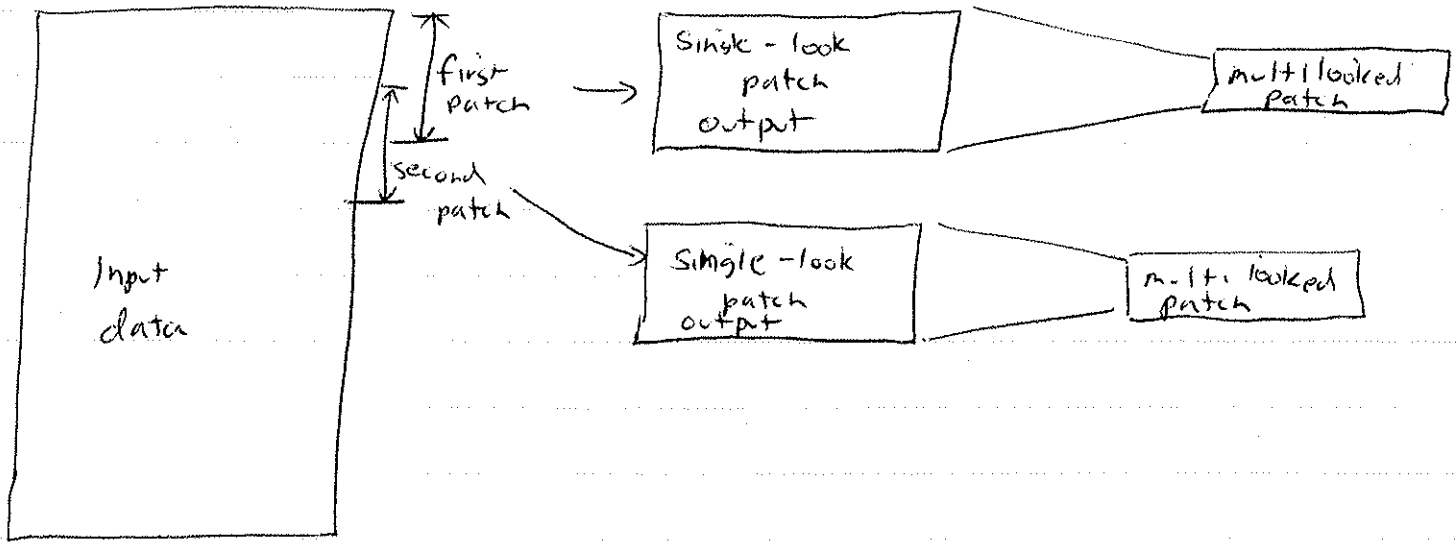
The same argument holds for azimuth with proper lengths substituted.

Multiple looks

We found that the resolution of a SAR is $\frac{\ell}{2}$ in azimuth, with the range resolution determined by transmitted bandwidth. For most systems, the azimuth resolution is finer than the range resolution. Thus if we wish to have undistorted maps with square pixels, we can average pixels in azimuth to create undistorted images, plus reduce statistical noise as well.

Once again we average powers in the output images, not complex values. The result of this azimuth multilooking is to reduce the image size in pixels, in azimuth.

Our focused SAR algorithm is then:



Note the offset from patch to patch in the input data set is less than the patch size. The offset is determined by the number of valid data samples.

Numerical example for patch offsets and looks

A/C radar, $v = 200$ m/s, prf = 400, $\delta_{range} = 5$ m, $\delta_{az} = \frac{1}{2}$ m ($\ell = 1$ m), range = 10 km, $\lambda = 2.5$ cm, zero Doppler.

Azimuth pulse length: $\frac{r_{DC} \lambda \, prf}{\ell v} = \frac{10^4 \cdot \frac{1}{40} \cdot 400}{1 \cdot 200} = 500$ pts

So let's read in 1024 lines and make each patch. The number of valid data points will be $1024 - 500 = 524$.

To equalize resolutions, we need to average 10 lines so

The final azimuth size will be 52.4 valid lines. We'll keep 52, corresponding to 520 input lines. So our offset scheme is