The 2011 GBT Raster Scan Data Set

- Isaac Shivvers, Pragaash Ponnusamy -

Goals of this presentation:

- Describe a relatively simple subsample of GBT SETI data
  - useful case study for others?
- Discuss what I found fruitful working with this data
  - template searches, clustering, outlier detection, et cetera

The Raw Data

- Data collection led by Andrew, Dan
- 11 hours of GBT observations
- Covered full Kepler field between 1.1 and 1.9 GHz
- Continuously moving telescope, "raster scan" pattern
- All data saved as raw baseband voltages with timestamps

Signal Search

- GPU-accelerated code produced power spectra
- Effective integration times of ~4.8 seconds
- Effective resolution of ~3 Hz
- Searched for all detections with signal-to-noise ratio > 10.0
  - Rolling window search through data
  - Sensitive to signals $\sim$3 - 400Hz wide
  - Correcting for Doppler drifts between ±10 Hz.

First Round RFI Rejection

- Beam of GBT (in arcminutes) = $\frac{13.01}{\nu_{GHz}}$
  - Beam FWHM in our data: 7 to 12 arcminutes
- Given the scan rate of telescope: $t_{max} \approx 16$ seconds
  - Identify any signal with duration $t > t_{max}$ as RFI
  - $\sim$90% of signals
  - $\sim 1 \times 10^6$ signals pass this test

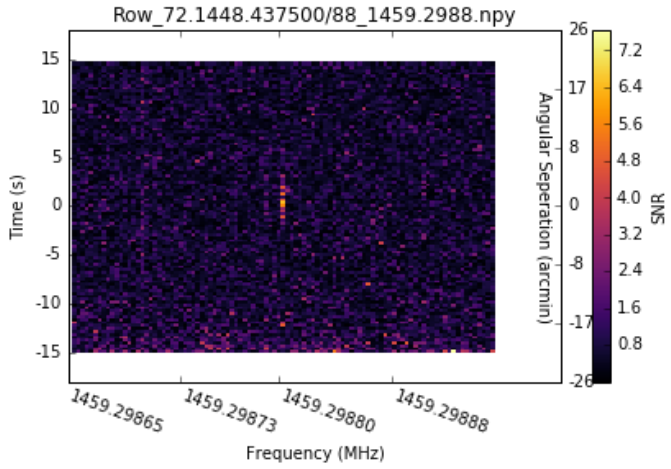
Data Storage

- Location, frequency, and S/N recorded for every signal
- 100px × 100px cutout recorded for 10^6 signals that pass 1^{st} round RFI rejection
  - ~30 seconds and ~300 Hz wide
  - called "waterfall plots"
  - time and position on sky degenerate

Waterfall Plots

In [25]:
```
allsignals = parse_catalog()
```

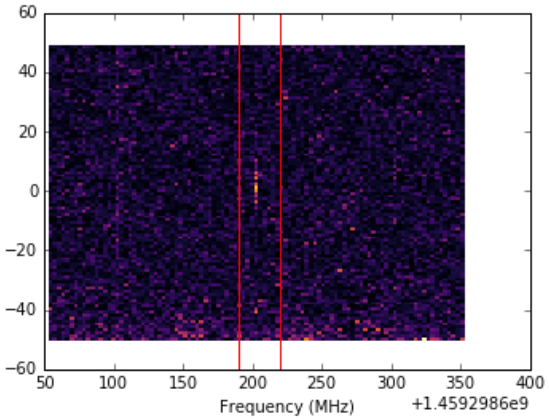
```
rando_wf = allsignals[np.random.randint(allsignals.shape[0])]
plotone( rando_wf['path'] )
print 'File:',rando_wf['path']
print 'RA: %f\nDEC: %f'%(rando_wf['ra'], rando_wf['dec'])
```

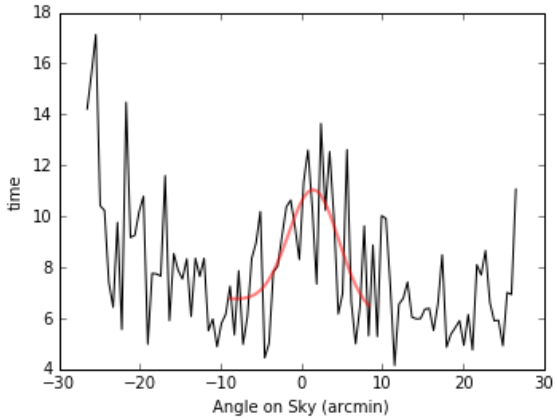



```
File: Row_72.1448.437500/88_1459.2988.npy
RA: 299.503500
DEC: 41.242400
```

```
plot_beacon( rando_wf['path'] )
```

```
deshifted by 0.0
Expected Beam FWHM: 8.91524063354
Signal FWHM: 7.29615608468
```





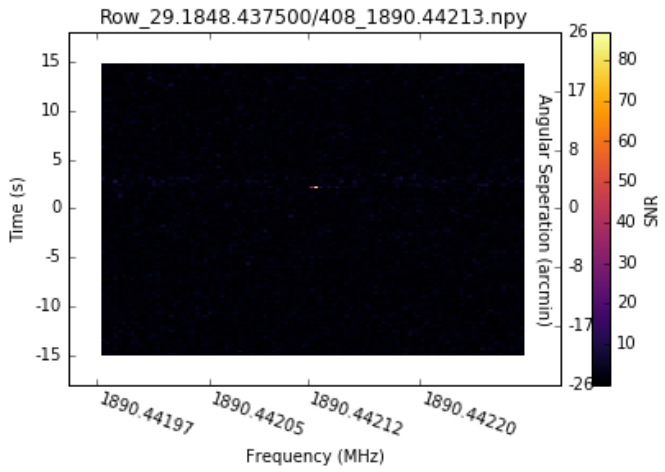
The Problem:

- we have $\sim 10^6$ 100px $\times$ 100px monochromatic images
- TASK: Find the "interesting" ones!

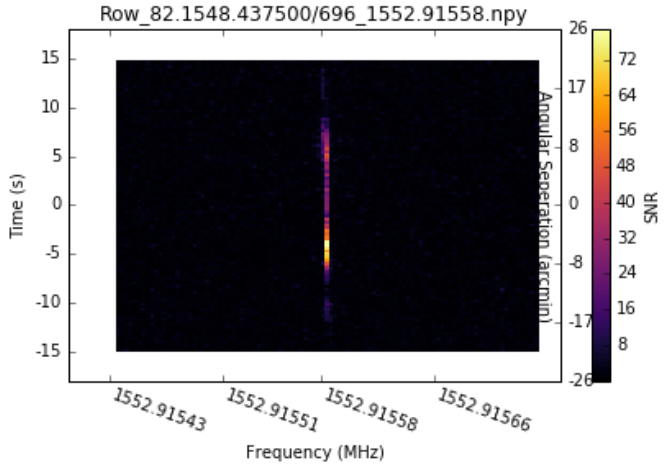
Simple methods:

- cross-correlate against templates of interest
  - for example, "pulses" or "beacons"

```
pulse_viewer.next()
```



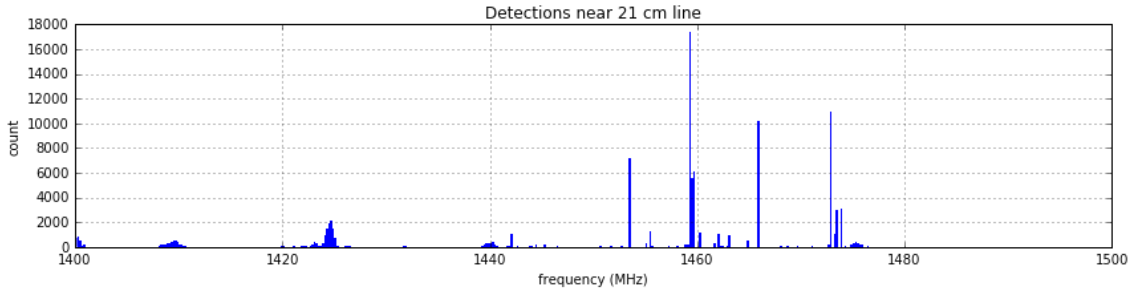
```
beacon_viewer.next()
```



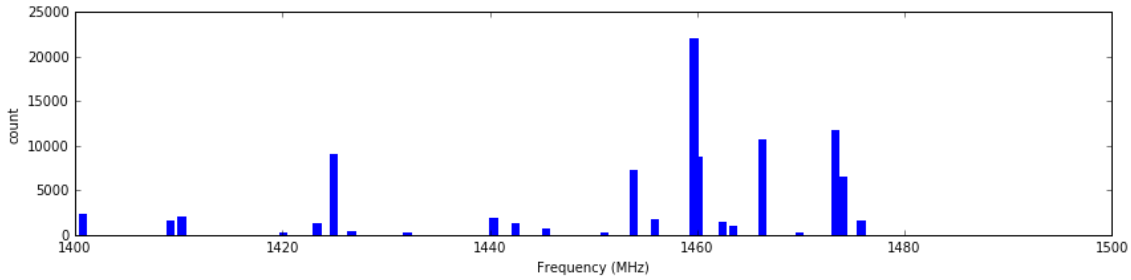
Machine Learning Methods

- Clustering in small dimensions
  - Frequency, RA, Decl, S/N, Drift Rate, et cetera
- Frequency is by far the most important for identifying RFI

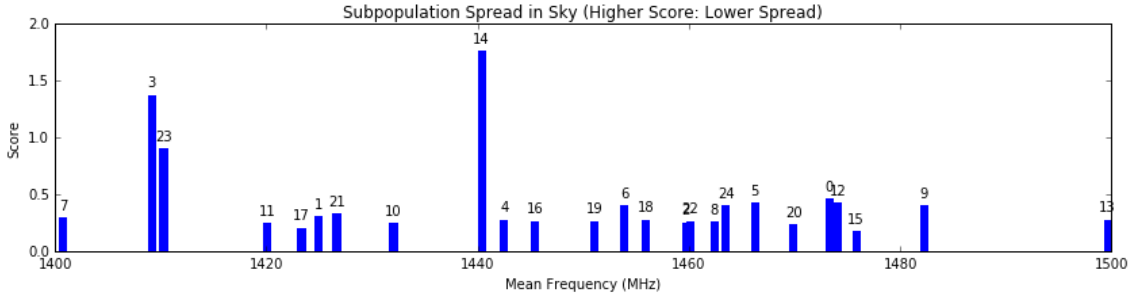
100 MHz of detections, from 1.4 GHz to 1.5 GHz



1D Clustering (K-means)



Are any frequency clusters localized on the sky?

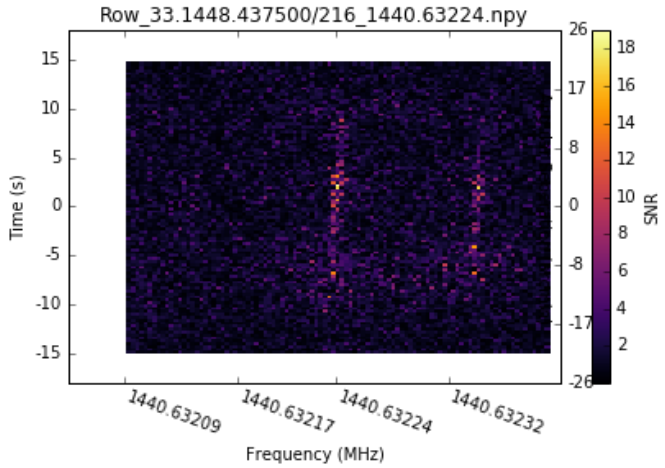



```
z: 0.0139
relative velocity: -4164 km/s
```

Let's have a look inside cluster 14

```
print 'Cluster 14: %d members' %len(clusterPaths)
sample = clusterPaths[np.random.randint(0,clusterPaths.size)]
plotone( sample )
```

```
Cluster 14: 1973 members
```



Still lots of sources!

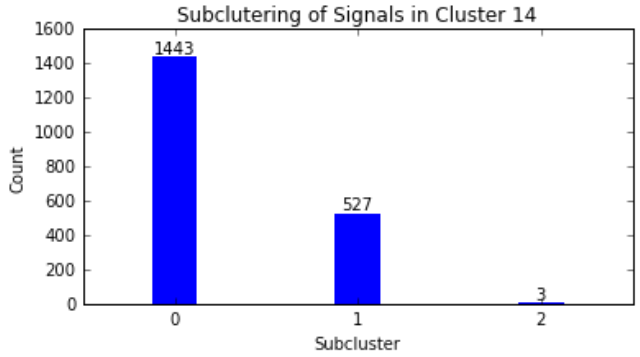
- Can't all be aliens: we are still dominated by RFI
- Let's go deeper
- subset of just cluster 14 is small enough to examine in detail

Image Dimensionality Reduction

- Native image data set has 100×100 dimensions
- Can bin data
  - Fast but limited; you're throwing information away
  - Can be powerful if know what you're looking for
- Or "whiten" images via Principal Component Analysis (PCA)
  - Top 30% of PCs for each image keeps ~92% of information
  - Reduce dimensionality by 1/3

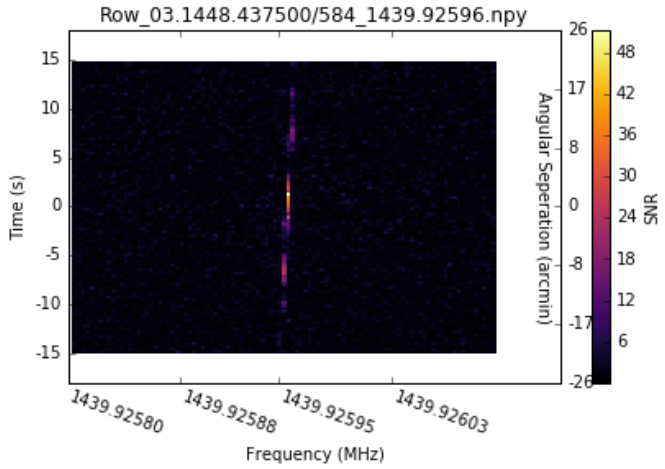
Clustering within the PCA-whitened images?

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
            connectivity=None, linkage='average',
            memory=Memory(cachedir=None), n_clusters=3, n_components=None,
            pooling_func=<function mean at 0x7f726419d578>)
```



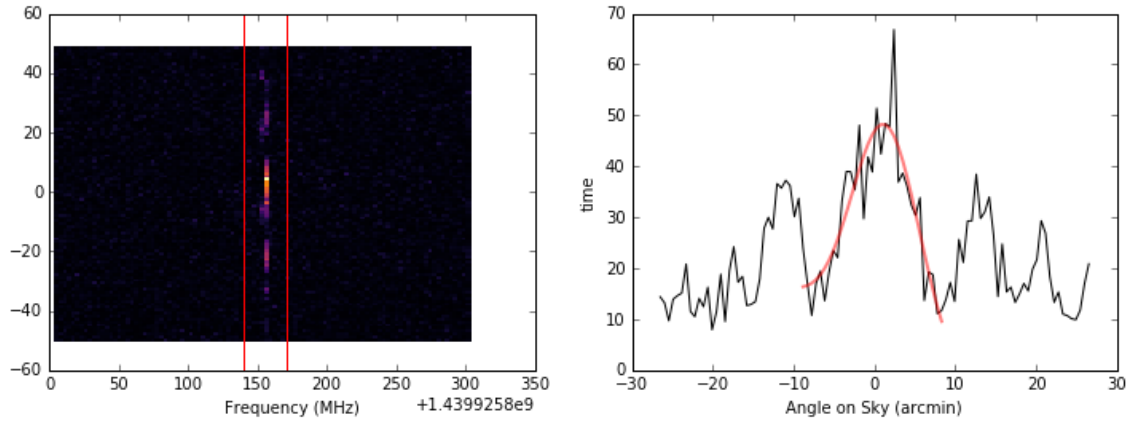
```
# Examine Agglomerative Cluster
K = 2
cpaths = AGCatalog[AGCatalog.cluster == K].path.values
print 'Cluster %d has %d members' %(K, cpaths.size)
cpath = cpaths[np.random.randint(0,cpaths.size)]
plotone(cpath)
```

```
Cluster 2 has 3 members
```




```
plot_beacon(cpath)
```

```
deshifted by 0.4
Expected Beam FWHM: 9.03518681055
Signal FWHM: 9.72750427176
```



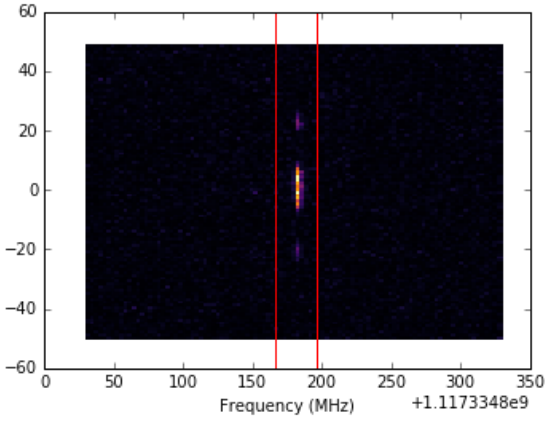
Scale it up?

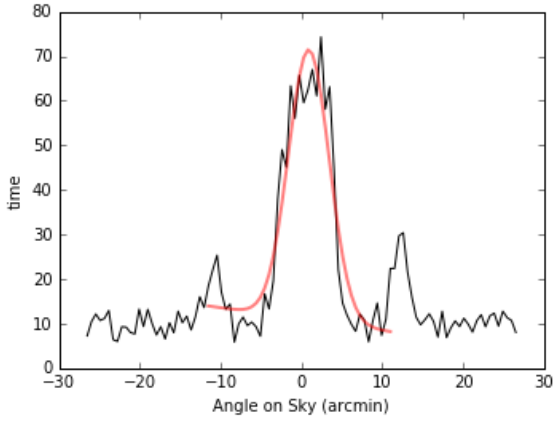
- I've applied many of the above steps to the full 1.1 - 1.9 GHz data set
- There are a lot of "interesting" signals

- DBSCAN on frequency & Doppler drift
  - label all clusters as RFI
  - examine only the outliers
- Local Outlier Factor (LOF)
  - assume aliens are rare
  - examine only the most-unique signals
- Nearest Neighbors
  - find signals most similar to any single identified signal of interest

For example, let's find other satellites via Nearest Neighbors

```
Row_03.1148.437500/584_1117.33498.npy
RA, Decl: 282.4062 41.6394
deshifted by 0.0
Expected Beam FWHM: 11.643777574
Signal FWHM: 5.93541753527
```






```
neighbor_viewer.next()
```

```
Row_03.1848.437500/584_1882.66502.npy
RA, Decl: 282.4062 41.6394
deshifted by 0.0
Expected Beam FWHM: 6.91041681263
Signal FWHM: 6.17586824365
```