Prototype of Complete OTF Data Reduction Pipeline

P. Schloerb and C. Dammer

September 2019

This notebook is designed to illustrate the spectral line data reduction pipeline for On-the-fly (OTF) mapping at the LMT.

The example can be run as a single script.

The notebook is in python 3.

SETUP

Import Statements

In [1]:

```
# Python Imports
import numpy as np
import matplotlib.pyplot as pl
import subprocess
import netCDF4

# Line Data Reduction Imports
from spec import *
from ifproc import *
from spec_viewer import *
from grid import *
from line_reduction import *

# set up the grid geometry
theGrid = grid()

# Pipeline S/W Imports
from cube_reader import *
```

Set up parameters

The pipeline requires a number of parameters to be set in order to complete its task. These are all set up in the following code cell.

The specific parameters are:

- Raw Spectral Line Data Reduction Parameters
  - `data_path` - path to the data files
  - `obsnum` - ObsNum of the observation to process
  - `bank` - select the *bank* of the spectral line (always 0 for SEQUOIA at this point)
  - `tsys` - a value to use for the system temperature if you choose *NOT* to use the calibration scan data.
  - `list_of_pixels` - a python list of the pixel id numbers to be processed
  - `use_calibration` - a boolean to select whether to use the cal scan (True) or just multiply by a constant.
  - `baseline_order` - order of the polynomial to be removed from

each spectrum.
    - `line_integral_list` - a python list of lists to provide regions for integration over line in analysis.
    - `baseline_list` - a python list of lists to indicate the regions where the baseline will be fit.
    - `slice_list` - a python list to provide limits of spectrum to be prepared for the SpecFile.
- SpecFile Preparation Parameters
    - `netcdf_filename` - name of the SpecFile, to be written in NetCDF format
    - `rms_cut` - threshhold for excluding individual spectra when viewing the spectra in display example for the SpecFile
- OTF Gridding Process Parameters
    - `ProgramPath` - full path name for the C program that does the gridding
    - `FitsFileName` - file name for the output FITS file
    - `xextent` - dimension of the data cube along the "x" spatial axis
    - `yextent` - dimension of the data cube along the "y" spatial axis
    - `rmscutoff` - threshold for exclusion of spectra from gridding process
- Parameters for the Cube Reader display examples
    - `Cmax` - maximum value in 2D map display example
    - `xb`, `xe` - pixel limits on x axis in 2D map display example
    - `yb`, `ye` - pixel limits on y axis in 2D map display example
    - `v_val` - pixel number on spectral axis for 2D display example
    - `x_val` - pixel number on x axis for 2D display example
    - `y_val` - pixel number on y axis for 2D display example

In [2]:

```
data_path = './example_data/'
obsnum = 79448
bank=0          # SEQUOIA data for always in bank 0
tsys = 200.     # tsys to use in case use_calibration is False
list_of_pixels = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]  # pixe
ls to process for map

use_calibration = False

baseline_order = 2
```

```python
line_integral_list = [[-50,0]]

baseline_list = [[-200,-55],[5,150]]
# we are only going to write channels in this velocity range
slice_list = [-100,75]

# here is the name of the netcdf file for output
# please note that if the file exists, it will be overwritten!
netcdf_filename = './Pipeline.nc'

#ViewSpecFile Parameters
rms_cut = 10

# C Program Parameters
ProgramPath='./spec_driver_fits'
FitsFileName = './Pipeline.fits'
xextent = '600'
yextent = '600'
rmscutoff = '1.3'

# Cube_Reader Parameters
Cmax = 6
v_val=95
x_val=86
y_val=86
xb,xe=50,120
yb,ye=50,120

# check to see whether files exist, and remove if they do
if os.path.isfile(netcdf_filename) == True:
    os.remove(netcdf_filename)
if os.path.isfile(FitsFileName) == True:
    os.remove(FitsFileName)
```

READ AND PROCESS THE DATA FILES

With all parameters defined, we can make the first step of processing the Roach and IFProc data files. The processing function used here is located in *line_reduction.py*.

The function `read_obsnum_otf` reads and reduces all the spectra in the observaiton file. Currently it is hard coded to reduce these using an average spectrum computed from all the reference observations in the map. The function can be modified to do one of the other options for reference removal if needed.

In [3]:

```
# read the data from the relevant files for this obsnum
# note use of "data_path" to set the path to the data.  Default
path is '/data_lmt/'
I,S = read_obsnum_otf(obsnum,list_of_pixels,bank,use_calibration
,tsys=tsys,path=data_path)
```

```
found roach0_79448_1_0_IRC+10216_2018-11-16_114845.n
c
append roach0_79448_1_0_IRC+10216_2018-11-16_114845.
nc
found roach1_79448_1_0_IRC+10216_2018-11-16_114845.n
c
append roach1_79448_1_0_IRC+10216_2018-11-16_114845.
nc
found roach2_79448_1_0_IRC+10216_2018-11-16_114845.n
c
append roach2_79448_1_0_IRC+10216_2018-11-16_114845.
nc
found roach3_79448_1_0_IRC+10216_2018-11-16_114845.n
c
append roach3_79448_1_0_IRC+10216_2018-11-16_114845.
nc
found ifproc_2018-11-16_079448_01_0000.nc
before read npix = 16
from pixels npix = 16
from xlen npix = 16
TRACKING Sequoia PIXEL  10
Map Parameters: Ra Continuous
HPBW= 16.0 XLength=   400.0 YLength=   400.0 XStep=
```

```
1.00 YStep=  0.70
./example_data/ifproc/ifproc_2018-11-16_079448_01_00
00.nc does not have bs parameters
79448 is a Map observation
read_roach ./example_data/spectrometer/roach0/roach0
_79448_1_0_IRC+10216_2018-11-16_114845.nc
r:0 inp:0 pix:0 to:-0.030000
r:0 inp:1 pix:1 to:-0.030000
r:0 inp:2 pix:2 to:-0.030000
r:0 inp:3 pix:3 to:-0.030000
read_roach ./example_data/spectrometer/roach1/roach1
_79448_1_0_IRC+10216_2018-11-16_114845.nc
r:1 inp:0 pix:4 to:-0.030000
r:1 inp:1 pix:5 to:-0.030000
r:1 inp:2 pix:6 to:-0.030000
r:1 inp:3 pix:7 to:-0.030000
read_roach ./example_data/spectrometer/roach2/roach2
_79448_1_0_IRC+10216_2018-11-16_114845.nc
r:2 inp:0 pix:8 to:-0.030000
r:2 inp:1 pix:9 to:-0.030000
r:2 inp:2 pix:10 to:-0.030000
r:2 inp:3 pix:11 to:-0.030000
read_roach ./example_data/spectrometer/roach3/roach3
_79448_1_0_IRC+10216_2018-11-16_114845.nc
r:3 inp:0 pix:12 to:-0.030000
r:3 inp:1 pix:13 to:-0.030000
r:3 inp:2 pix:14 to:-0.030000
r:3 inp:3 pix:15 to:-0.030000
```

Example of Line Processing

Here we provide a short example of line analysis of the reduced data. A particular spectrum from the OTF data (located near the source position) is selected. Specifically, we select the 2661st spectrum from pixel 10.

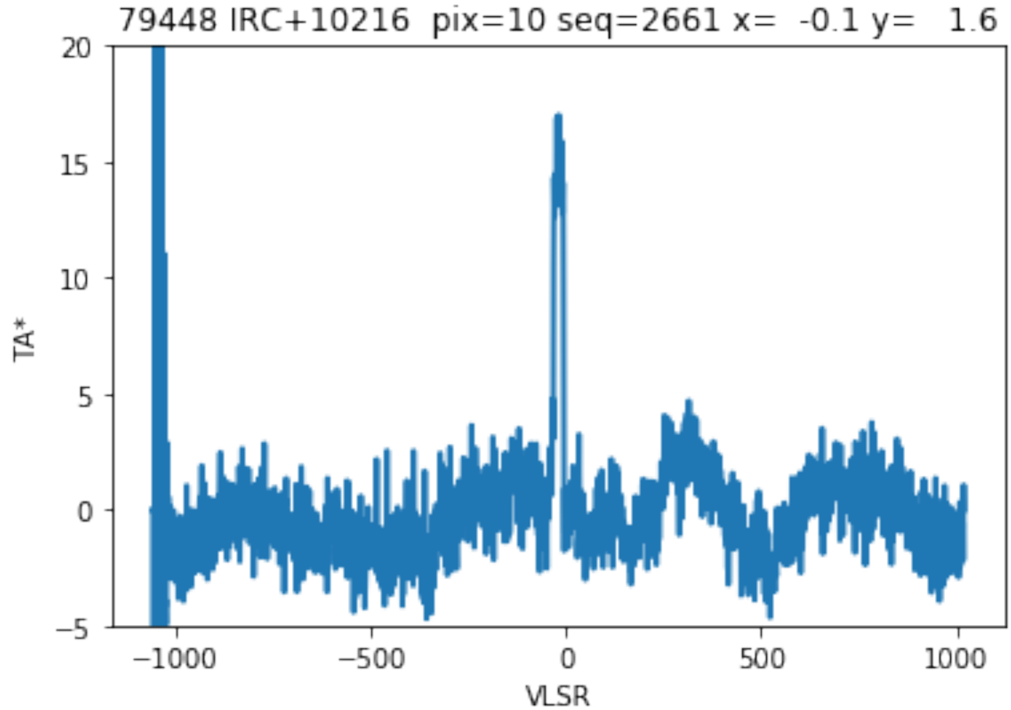
For the first figure, we create a LineData object, `ll`, from the specific spectrum in our `spec_bank` object. We just make a simple plot of the entire spectrum.

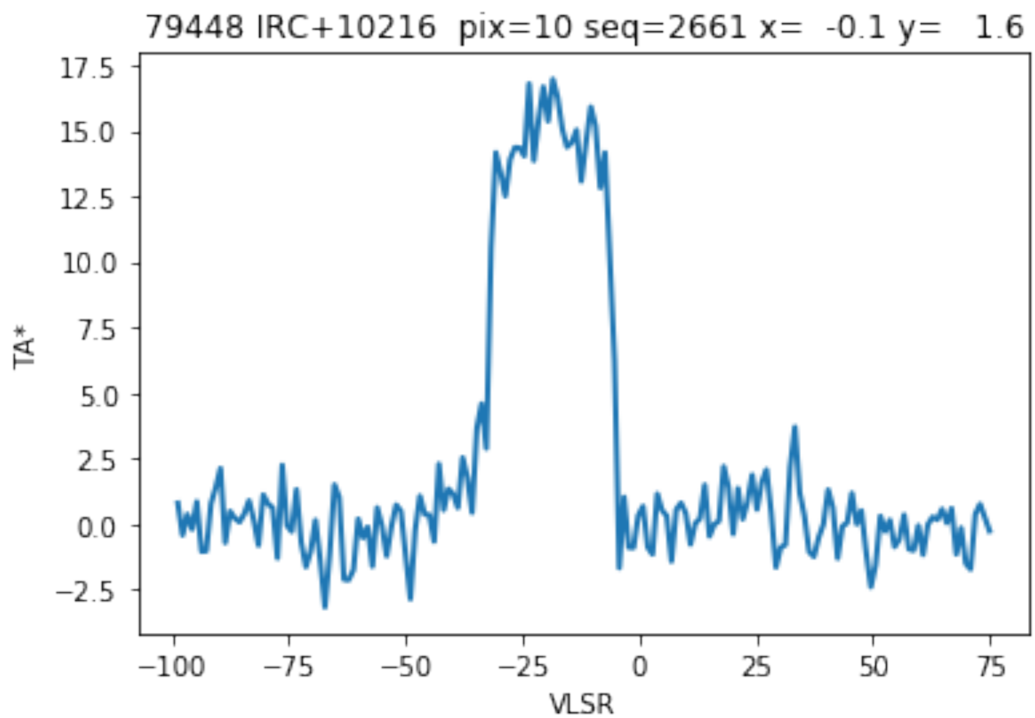
For the second figure, we actually do some processing. A Line object, `lll`, is created by taking a slice of the spectrum over the range specified by our `slice_list`. Then the baseline method in this class is used to remove the baseline. Finally, a simple plot is made.

In [4]:

```
# For fun, here is a trial run with one processed spectrum
ipix  = 10      # SEQUOIA PIXEL
ispec = 2661  # otf spectrum in sequence (IRC_79957 8069)
# print x,y values
print('x= ',S.roach[ipix].xmap[S.roach[ipix].ons][ispec],'  y= '
,S.roach[ipix].ymap[S.roach[ipix].ons][ispec])

# create a LineData object for line reduction
ll = LineData(I,bank,S.nchan,S.bandwidth,S.roach[ipix].reduced_s
pectra[ispec])
pl.plot(ll.xarray,ll.yarray)
pl.xlabel(ll.xname)
pl.ylabel('TA*')
pl.title('%d %s  pix=%d seq=%d x=%6.1f y=%6.1f'%(S.obsnum,S.sour
ce,ipix,ispec,S.roach[ipix].xmap[S.roach[ipix].ons][ispec],S.roa
ch[ipix].ymap[S.roach[ipix].ons][ispec]))
pl.ylim([-5,20])

# select a slice from the spectrum and load into a Line object f
or processing
# limits are specified by velocity
lll = ll.vslice(slice_list[0],slice_list[1])
# find channel numbers for baseline region
bblist,nblist = lll.xlist(baseline_list)
# remove the baseline
lll.baseline(bblist,nblist,baseline_order=1)
pl.figure()
pl.plot(lll.xarray,lll.yarray)
pl.xlabel(lll.xname)
pl.ylabel('TA*')
pl.title('%d %s  pix=%d seq=%d x=%6.1f y=%6.1f'%(S.obsnum,S.sour
ce,ipix,ispec,S.roach[ipix].xmap[S.roach[ipix].ons][ispec],S.roa
ch[ipix].ymap[S.roach[ipix].ons][ispec]))
```

```
x=  -0.1398945076734548   y=  1.596218470757127

Out[4]:

Text(0.5, 1.0, '79448 IRC+10216  pix=10 seq=2661 x=
-0.1 y=   1.6')
```





CREATING A SPECFILE

Set up Reduction Process

This step exists to (1) figure out the dimensions of the data array to be written to the SpecFile in NetCDF format; and (2) prepare header information for the spectral line data to be recorded.

The function `count_otf_spectra` just counts the total number of spectra to be processed. This value is used to set the one of the dimensions of the NetCDF data written in the SpecFile.

The next set of steps figures out the number of channels to be written for each spectrum, assuming that we limit the data to a slice of channels defined by the `vslice` method. This step results in an instance of a Line object, `L`, thaht contains all the useful header variables needed for further spectroscopic analysis.

In [5]:

```
# count the total number of spectra that will be processed and w
ritten to file
total_spectra = count_otf_spectra(S,list_of_pixels)

# make a dummy spectrum to count the channels after processing s
teps
LD = LineData(I,bank,S.nchan,S.bandwidth,np.zeros(S.nchan))
L = LD.vslice(slice_list[0],slice_list[1])
nchan_to_save = L.nchan
```

```
0 5739 5739
1 5739 11478
2 5739 17217
3 5739 22956
4 5738 28694
5 5738 34432
6 5738 40170
7 5738 45908
8 5739 51647
9 5739 57386
10 5739 63125
11 5739 68864
12 5738 74602
13 5738 80340
14 5738 86078
15 5738 91816
Total Number of Spectra = 91816
```

Writing the netCDF file

The SpecFile is a NetCDF file. The script in the next code cell processes each of the spectra in the OTF map and writes it into the file. SpecFiles are then used at the next step to actually grid the data into a data cube.

The beginning of the script sets up the NetCDF file for writing by defining and writing all the header variables.

For a single file, all the spectral information is the same, so we have a class that uses the scan data header to write a set of NetCDF header variables. This class has a method which, given a Line object, will write the header data to the netcdf file.

Then, in the main for loop, we process each pixel in turn considering all spectra for that pixel. Currently, the map coordinates are linked to the map mode used when the data were collected. However, this need not be the case. We could make maps in RA-Dec even if the data were obtained in Az-El.

For each spectrum, we process the data by fitting a baseline. It is useful to limit the size of the SpecFile that will be written by limiting the number of channels to a region around the line of interest. We do this by creating `L`, a LineData object with a single spectrum. Then, we use the LineData method `vslice` to create a Line object, `LL`, which is a slice of the full array of spectral data covering the region given by `slice_list`. `L` then contains the extracted data and the Line methods are used for computing the baseline. Finally, the baselined data are written to the SpecFile. Note that we followed the same steps to define the data to be written that were followed when we set up `nchan_to_save` in the previous code cell.

Eventually, this script will be written as a function with options to hide all the NetCDF stuff and make it easier to use.

In [6]:

```
# write the netCDF file

# open Dataset.  If the file exists, we stop here!
nc = netCDF4.Dataset(netcdf_filename, 'w', format='NETCDF4')
```

```
# dimension of number of spectra is from total number count
nc_dimension_nspec = nc.createDimension('nspec',total_spectra)
# dimension of nymber of channels in spectrum is from trial redu
ction step
nc_dimension_nchan = nc.createDimension('nchan',nchan_to_save)
# just doing 20 characters in string
nc_dimension_nlabel = nc.createDimension('nlabel',20)

# the Observation Header
nc_obsnum = nc.createVariable('Header.Obs.ObsNum','i4')
nc.variables['Header.Obs.ObsNum'][0] = S.obsnum

nc_source = nc.createVariable('Header.Obs.SourceName','c',('nlab
el',))
if len(S.source) > 19:
    nc_source[0:19] = S.source[0:19]
else:
    nc_source[0:len(S.source)] = S.source[0:len(S.source)]

nc_x_position = nc.createVariable('Header.Obs.XPosition','f4')
nc_y_position = nc.createVariable('Header.Obs.YPosition','f4')
if S.map_coord == 1:
    nc.variables['Header.Obs.XPosition'][0] = S.ifproc.source_RA
/np.pi*180.0
    nc.variables['Header.Obs.YPosition'][0] = S.ifproc.source_De
c/np.pi*180.0
else:
    nc.variables['Header.Obs.XPosition'][0] = 0.0
    nc.variables['Header.Obs.YPosition'][0] = 0.0


# using line header information derived from spec bank
ncl = NetCDFLineHeader(nc)
ncl.write_line_header_variables(L) # write using the result of t
rial run

nc_pix = nc.createVariable('Data.Pixel','i4',('nspec',))
nc_seq = nc.createVariable('Data.Sequence','i4',('nspec',))
nc_x = nc.createVariable('Data.XPos','f4',('nspec',))
nc_x.units = 'arcsec'
nc_y = nc.createVariable('Data.YPos','f4',('nspec',))
nc_y.units = 'arcsec'
nc_rms = nc.createVariable('Data.RMS','f4',('nspec',))
nc_rms.units = 'K'
```

```
nc_data = nc.createVariable('Data.Spectra','f4',('nspec','nchan'
))
nc_data.units = 'K'

count = 0
for ipix in list_of_pixels:
    i = S.find_pixel_index(ipix)
    n_spectra = len(S.roach[i].xmap[S.roach[i].ons])
    x_spectra = S.roach[i].xmap[S.roach[i].ons] # x coordinate
    y_spectra = S.roach[i].ymap[S.roach[i].ons] # y coordinate
    if I.map_coord == 0:
        gx,gy = theGrid.azel(S.elev/180.*np.pi,I.tracking_beam)
    else:
        parang = np.mean(S.roach[i].pmap[S.roach[i].ons]) # aver
age parang
        gx,gy = theGrid.radec(S.elev/180.*np.pi,parang,I.trackin
g_beam)

    for j in range(n_spectra):
        # process each spectrum
        L = LineData(I,bank,S.nchan,S.bandwidth,S.roach[i].reduc
ed_spectra[j])
        LL = L.vslice(slice_list[0],slice_list[1])
        bbase,nbase = LL.xlist(baseline_list)
        LL.baseline(bbase,nbase,baseline_order=baseline_order)

        # write the reduced line into the NetCDF file
        nc_data[count,:] = LL.yarray
        nc_rms[count] = LL.rms
        nc_pix[count] = ipix
        nc_seq[count] = j
        nc_x[count] = x_spectra[j]-gx[ipix]
        nc_y[count] = y_spectra[j]-gy[ipix]
        count = count + 1


nc.close()
print('netCDF Done')
```

```
netCDF Done
```

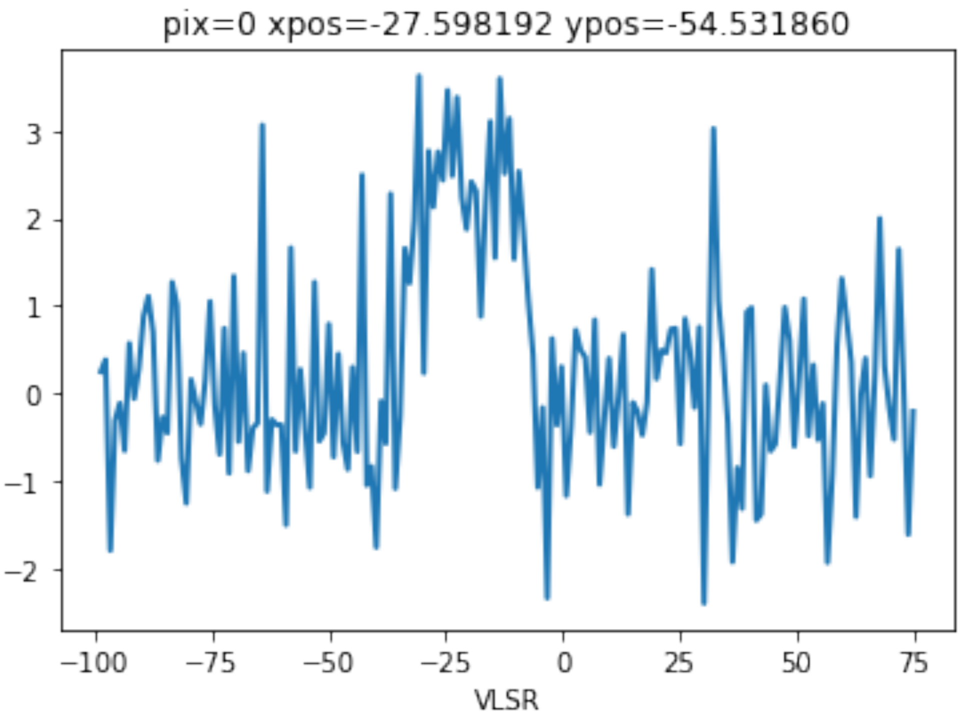
Simple Example of Reading back the SpecFile

The following code cell provides a simple illustration of reading some data back from the SpecFile we have just written.

In [7]:

```
# read back the netcdf file as a check
nc = netCDF4.Dataset(netcdf_filename, 'r', format='NETCDF4')
print('ObsNum = %d'%(nc.variables['Header.Obs.ObsNum'][0]))
print('Source = %s'%(netCDF4.chartostring(nc.variables['Header.O
bs.SourceName'][:])))

ncl = NetCDFLineHeader(nc)
ncl.read_line_header_variables(L)
print('number of channels= %d'%(L.nchan))
print('x axis name= %s'%(L.xname))

xpos = nc.variables['Data.XPos']
ypos = nc.variables['Data.YPos']
pix = nc.variables['Data.Pixel']
print('x extremes = %f %f'%(np.min(xpos),np.max(xpos)))
print('y extremes = %f %f'%(np.min(ypos),np.max(ypos)))

# plot a spectrum for one dump
ipos = 2636
pl.plot(nc.variables['Header.SpectrumAxis.CAXIS'],nc.variables['
Data.Spectra'][ipos][:])
pl.title('pix=%d xpos=%f ypos=%f'%(pix[ipos],xpos[ipos],ypos[ipo
s]))
pl.xlabel(L.xname)
nc.close()
```

```
ObsNum = 79448
Source = IRC+10216
number of channels= 172
x axis name= VLSR
x extremes = -244.573685 269.080872
y extremes = -256.818237 267.553284
```



VIEWING THE SPECFILE

The following code cells are being turned into a new class to provide methods for reading and displaying SpecFiles.

In the first code cell, below, we just read the data.

In [8]:

```
nc = netCDF4.Dataset(netcdf_filename, 'r', format='NETCDF4')
obsnum = nc.variables['Header.Obs.ObsNum'][0]
source_name = netCDF4.chartostring(nc.variables['Header.Obs.Sour
ceName'][:])

x_position =  nc.variables['Header.Obs.XPosition'][0]
y_position =  nc.variables['Header.Obs.YPosition'][0]

nchan = nc.variables['Header.Line.NChannels'][0]
chan = nc.variables['Header.Line.ChannelNumber'][:]
cdelt = nc.variables['Header.SpectrumAxis.CDELT'][0]
crpix = nc.variables['Header.SpectrumAxis.CRPIX'][0]
crval = nc.variables['Header.SpectrumAxis.CRVAL'][0]
ctype = netCDF4.chartostring(nc.variables['Header.SpectrumAxis.C
TYPE'][:])
caxis = nc.variables['Header.SpectrumAxis.CAXIS'][:]

print('number of channels= %d'%(nchan))

pixel = nc.variables['Data.Pixel'][:]
xpos = nc.variables['Data.XPos'][:]
ypos = nc.variables['Data.YPos'][:]
rms = nc.variables['Data.RMS'][:]
data = nc.variables['Data.Spectra'][:]

print('shape of OTF data %d %d'%(np.shape(data)))
nc.close()
```

```
number of channels= 172
shape of OTF data 91816 172
```

SpecFile Plotting Example 1

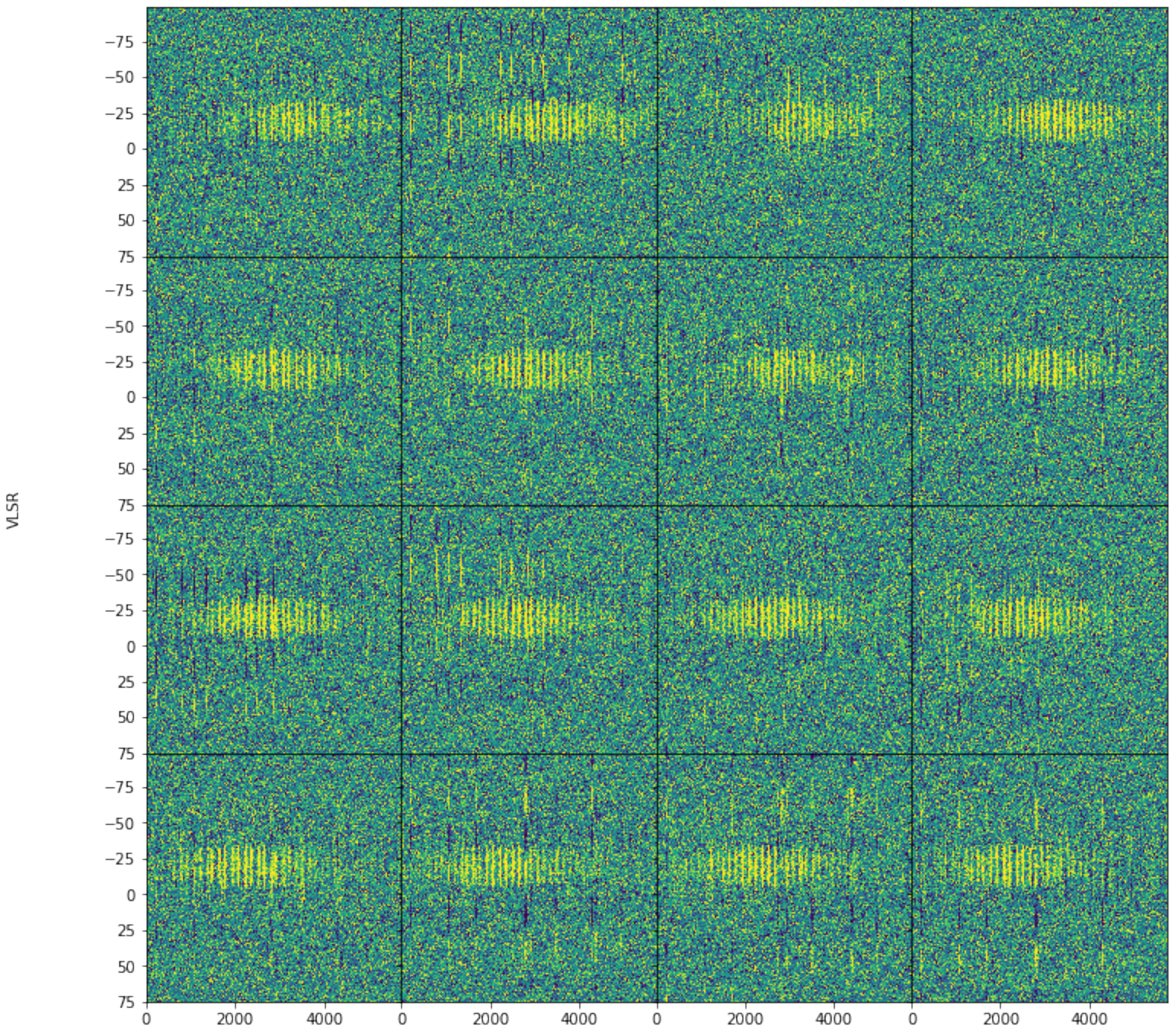
These are some example plots of the data in the SpecFile.

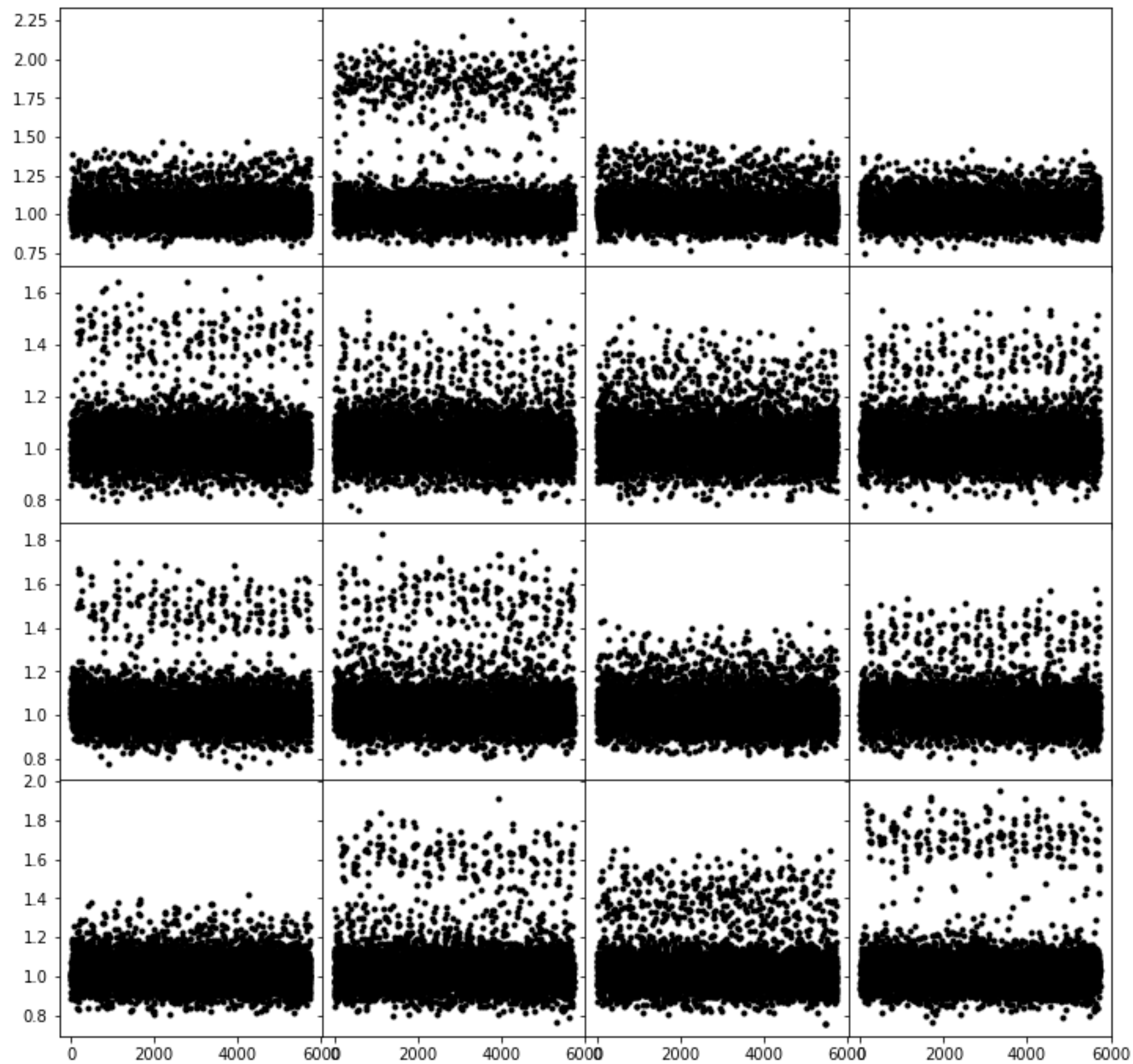
The first is a *waterfall* plot of all the spectra. In this case, it is easy to see the line from the source in each pixel of the array (as vertical yellow stripes). However, we can also see other features where individual spectra have poor baselines.

The second example shows a simple graph of the baseline RMS for each spectrum in each pixel. It is clear that some of the spectra have much poorer baselines. These are candidates for elimination when we construct our data cube.

In [9]:

```
fig1, ax1 = pl.subplots(4, 4, sharex='col', sharey='row',gridspec_kw={'hspace': 0, 'wspace': 0},figsize=(12,12))
fig2, ax2 = pl.subplots(4, 4, sharex='col', sharey='row',gridspec_kw={'hspace': 0, 'wspace': 0},figsize=(12,12))
fig1.text(0.02, 0.5, ctype, va='center', rotation='vertical')
j=0
k=0
for i in range(0,16):
    the_pixel = i
    rms_cut = 10
    pindex = np.where(pixel==the_pixel)[0]
    rindex = np.where(rms[pindex]<rms_cut)[0]
    #print()'Total Scans for pixel %d = %d   Good Scans = %d'%(the_pixel,len(pindex),len(rindex)))
    ax1[j,k].imshow(data[pindex[rindex]].transpose(),origin='lower',extent=[0.,float(len(rindex)),caxis[0],caxis[-1]],clim=[-2,2],aspect='auto')
    ax2[j,k].plot(rms[pindex[rindex]],'k.')
    if(k == 3):
        k=0
        j=j+1
    else:
        k=k+1
```


VLSR
−75
−50
−25
0
25
50
75
0
2000
4000



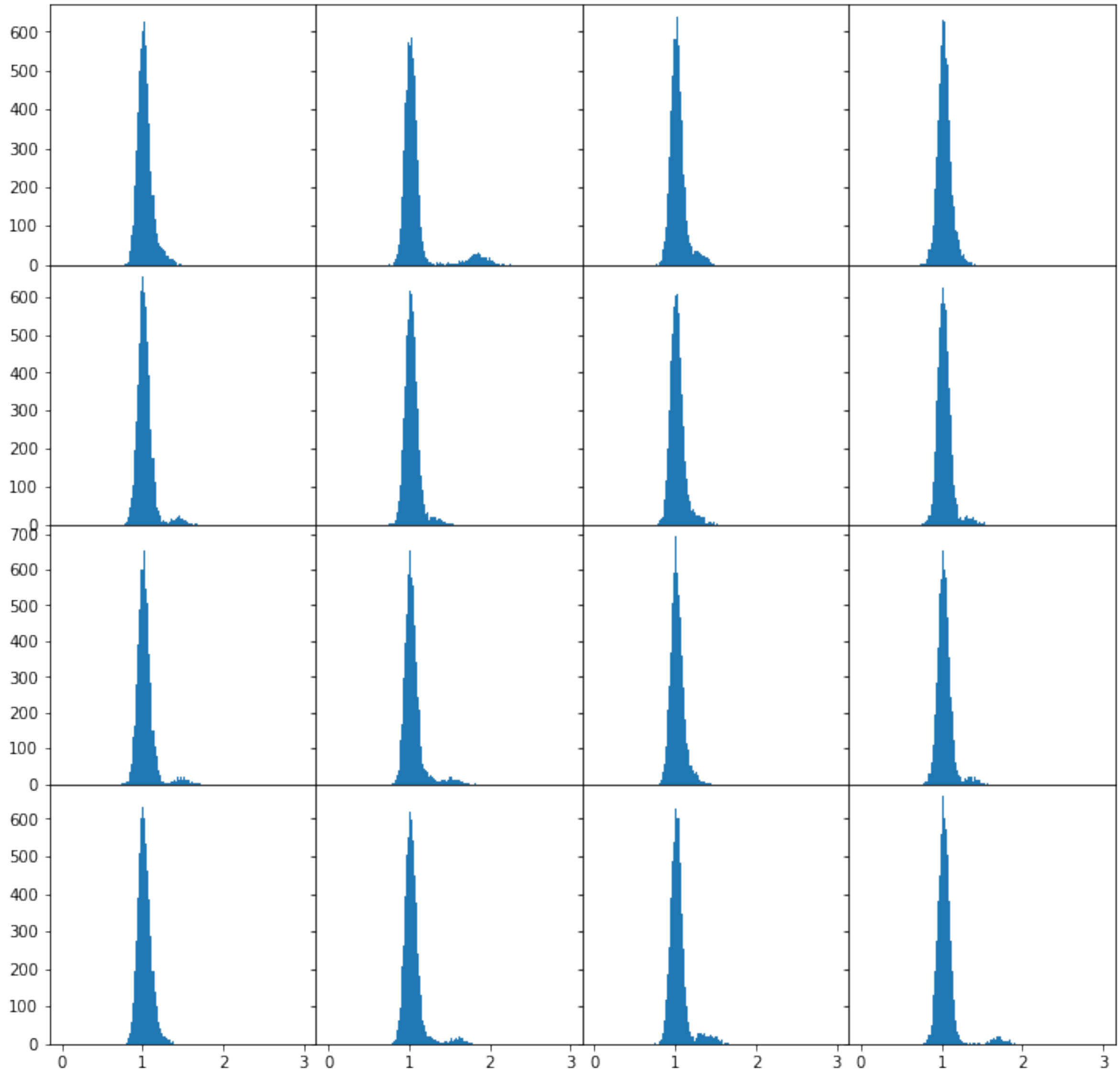
SpecFile Plotting Example 2

Here are two other examples. Ths first is a histrogram of the RMS values in the spectra for each pixel. Note that there is often a second peak, corresponding to "bad" spectra. We can use histograms like this to select rms values to be excluded when the data cube is prepared.

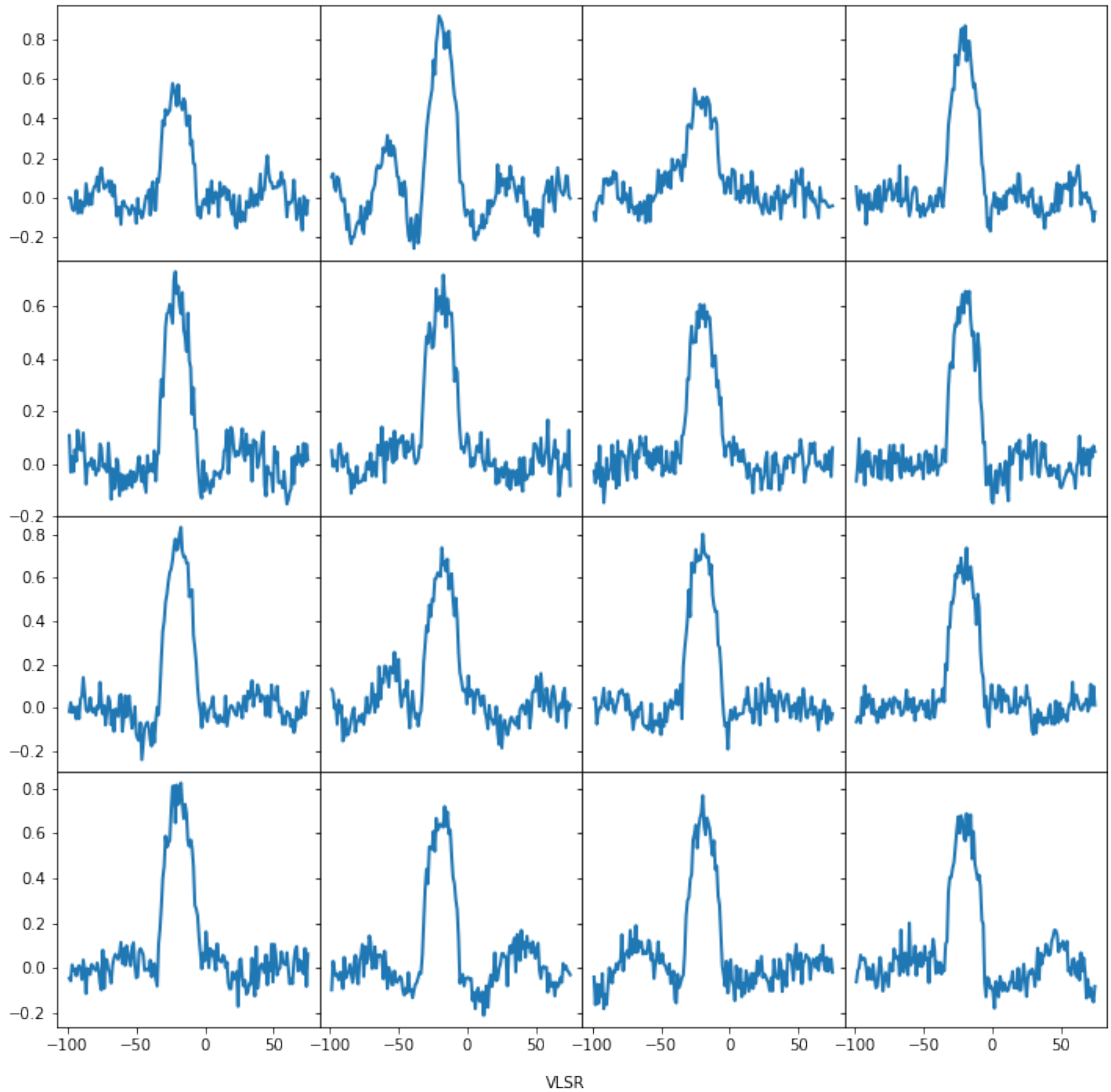
The second plot shows the mean of all the spectra. For this map, the line is apparent. We also see the effect of poor baselines on an average of all spectra.

In [10]:

```
fig3, ax3 = pl.subplots(4, 4, sharex='col', sharey='row',gridspe
c_kw={'hspace': 0, 'wspace': 0},figsize=(12,12))
fig4, ax4 = pl.subplots(4, 4, sharex='col', sharey='row',gridspe
c_kw={'hspace': 0, 'wspace': 0},figsize=(12,12))
fig4.text(0.5, 0.08, ctype, ha='center')
j=0
k=0
for i in range(0,16):
    the_pixel = i
    pindex = np.where(pixel==the_pixel)[0]
    rindex = np.where(rms[pindex]<rms_cut)[0]
    ax3[j,k].hist(rms[pindex[rindex]],bins=np.arange(0.,3.02,.02
))
    ax4[j,k].plot(caxis,np.mean(data[pindex[rindex]],axis=0))
    if(k == 3):
        k=0
        j=j+1
    else:
        k=k+1
```



600
500
400
300
200
100
0
700
0
1
2
3



0.8
0.6
0.4
0.2
0.0
−0.2
−100
−50
0
50
VLSR

OTF GRIDDING

The OTF gridding routine takes the spectral line data in the SpecFile and convolves it into a regular grid.

The OTF routine has been written in C. The executable program is run as a subprocess call. The program has a number of switches which allow parameters, such as file names, to be input to the program.

The set of command line arguments is:

- `-h` - help list of all commands (no argument) ... not necessarily up to date.
- `-i` - input file name
- `-o` - output file name
- `-l` - resolution_size; the actual λ/D (arcsec)
- `-c` - cell_size; nominally $\lambda/(2D)$ (arcsec)
- `-z` - rms cutoff value for inclusion in the cube (K)
- `-f` - specifies convolution filter (0=PILL BOX; 1=JINC; 2=GAUSSIAN)
- `-x` - x extent of the map (arcsec)
- `-y` - y extent of the map (arcsec)
- `-r` - maximum extent of convolution function in units of λ/D
- `-0` - the jinc "a" parameter
- `-1` - the jinc "b" parameter (also gaussian half power)
- `-2` - the jinc "c" parameter

In [11]:

```
with open('out.txt','w+') as outputfile:
    with open('err.txt','w+') as errorfile:
        exit_code=subprocess.call([ProgramPath,
                                   '-i',netcdf_filename,
                                   '-o',FitsFileName,
                                   '-x',xextent,
                                   '-y',yextent,
                                   '-z',rmscutoff],
                                  stdout=outputfile,
                                  stderr=errorfile)
        # reset stdout file to read from it
        outputfile.seek(0)
        # save output (if any) in variable
        standard_output=outputfile.read()
        print('STDOUT ****************************')
        print(standard_output)

        # reset stderr file to read from it
        errorfile.seek(0)
        # save errors (if any) in variable
        standard_error = errorfile.read()
        print('STDERR ****************************')
        print(standard_error)
print('Exit Code: %d'%(exit_code))
```

```
STDOUT ****************************
11 ./spec_driver_fits -i ./Pipeline.nc -o ./Pipeline
.fits
./Pipeline.nc
./Pipeline.nc
about to read ./Pipeline.nc
about to open file ./Pipeline.nc
file ./Pipeline.nc opened
Dimensions complete 91816 172
Header.Obs complete
Header.SpectrumAxis.CRVAL
Header.SpectrumAxis.CRPIX
Header.SpectrumAxis.CDELT
Header.SpectrumAxis.CTYPE
Header.SpectrumAxis.XAxis
Data.Spectra
Data.Spectra completed
```

```
Data.XPos
Data.YPos
Data.Pixel
Data.Sequence
Data.RMS
file: ./Pipeline.nc nspec= 91816 nchan= 172
allocated theData
completed read
n_cell= 7
r (as)   c
 0.00   1.0000
 0.16   0.9994
 0.33   0.9975
 0.49   0.9945
 0.66   0.9902
 0.82   0.9847
 0.99   0.9780
 1.15   0.9701
 1.32   0.9611
 1.48   0.9509
 1.65   0.9396
 1.81   0.9273
 1.98   0.9139
 2.14   0.8995
 2.31   0.8842
 2.47   0.8679
 2.64   0.8507
 2.80   0.8327
 2.96   0.8139
 3.13   0.7943
 3.29   0.7741
 3.46   0.7531
 3.62   0.7316
 3.79   0.7096
 3.95   0.6870
 4.12   0.6640
 4.28   0.6407
 4.45   0.6170
 4.61   0.5930
 4.78   0.5689
 4.94   0.5445
 5.11   0.5201
 5.27   0.4956
 5.44   0.4712
 5.60   0.4467
```

5.76	0.4224
5.93	0.3983
6.09	0.3744
6.26	0.3507
6.42	0.3274
6.59	0.3044
6.75	0.2818
6.92	0.2596
7.08	0.2379
7.25	0.2167
7.41	0.1961
7.58	0.1760
7.74	0.1565
7.91	0.1377
8.07	0.1196
8.24	0.1021
8.40	0.0853
8.56	0.0693
8.73	0.0540
8.89	0.0394
9.06	0.0256
9.22	0.0125
9.39	0.0002
9.55	-0.0113
9.72	-0.0220
9.88	-0.0320
10.05	-0.0412
10.21	-0.0497
10.38	-0.0574
10.54	-0.0643
10.71	-0.0706
10.87	-0.0761
11.04	-0.0810
11.20	-0.0852
11.36	-0.0887
11.53	-0.0916
11.69	-0.0939
11.86	-0.0956
12.02	-0.0967
12.19	-0.0973
12.35	-0.0974
12.52	-0.0970
12.68	-0.0962
12.85	-0.0949
13.01	-0.0933

13.18	-0.0912
13.34	-0.0889
13.51	-0.0862
13.67	-0.0833
13.84	-0.0800
14.00	-0.0766
14.16	-0.0730
14.33	-0.0692
14.49	-0.0652
14.66	-0.0612
14.82	-0.0570
14.99	-0.0528
15.15	-0.0486
15.32	-0.0443
15.48	-0.0400
15.65	-0.0358
15.81	-0.0316
15.98	-0.0274
16.14	-0.0233
16.31	-0.0193
16.47	-0.0155
16.64	-0.0117
16.80	-0.0081
16.96	-0.0046
17.13	-0.0013
17.29	0.0019
17.46	0.0049
17.62	0.0077
17.79	0.0103
17.95	0.0128
18.12	0.0150
18.28	0.0171
18.45	0.0190
18.61	0.0207
18.78	0.0222
18.94	0.0235
19.11	0.0246
19.27	0.0255
19.44	0.0263
19.60	0.0269
19.76	0.0273
19.93	0.0276
20.09	0.0277
20.26	0.0277
20.42	0.0275

20.59	0.0272
20.75	0.0268
20.92	0.0262
21.08	0.0256
21.25	0.0249
21.41	0.0240
21.58	0.0231
21.74	0.0221
21.91	0.0211
22.07	0.0200
22.24	0.0189
22.40	0.0177
22.56	0.0165
22.73	0.0153
22.89	0.0140
23.06	0.0128
23.22	0.0115
23.39	0.0103
23.55	0.0091
23.72	0.0079
23.88	0.0067
24.05	0.0055
24.21	0.0044
24.38	0.0033
24.54	0.0023
24.71	0.0013
24.87	0.0004
25.04	-0.0005
25.20	-0.0014
25.36	-0.0022
25.53	-0.0029
25.69	-0.0036
25.86	-0.0042
26.02	-0.0047
26.19	-0.0053
26.35	-0.0057
26.52	-0.0061
26.68	-0.0064
26.85	-0.0067
27.01	-0.0069
27.18	-0.0071
27.34	-0.0072
27.51	-0.0073
27.67	-0.0074
27.84	-0.0073

28.00	-0.0073
28.16	-0.0072
28.33	-0.0071
28.49	-0.0070
28.66	-0.0068
28.82	-0.0066
28.99	-0.0063
29.15	-0.0061
29.32	-0.0058
29.48	-0.0055
29.65	-0.0053
29.81	-0.0049
29.98	-0.0046
30.14	-0.0043
30.31	-0.0040
30.47	-0.0037
30.64	-0.0033
30.80	-0.0030
30.96	-0.0027
31.13	-0.0024
31.29	-0.0021
31.46	-0.0018
31.62	-0.0015
31.79	-0.0012
31.95	-0.0010
32.12	-0.0007
32.28	-0.0005
32.45	-0.0003
32.61	-0.0001
32.78	0.0001
32.94	0.0003
33.11	0.0005
33.27	0.0006
33.44	0.0008
33.60	0.0009
33.76	0.0010
33.93	0.0011
34.09	0.0011
34.26	0.0012
34.42	0.0012
34.59	0.0013
34.75	0.0013
34.92	0.0013
35.08	0.0013
35.25	0.0013

```
35.41   0.0013
35.58   0.0013
35.74   0.0012
35.91   0.0012
36.07   0.0012
36.24   0.0011
36.40   0.0011
36.56   0.0010
36.73   0.0010
36.89   0.0009
37.06   0.0008
37.22   0.0008
37.39   0.0007
37.55   0.0007
37.72   0.0006
37.88   0.0005
38.05   0.0005
38.21   0.0004
38.38   0.0004
38.54   0.0003
38.71   0.0003
38.87   0.0002
39.04   0.0002
39.20   0.0002
39.36   0.0001
39.53   0.0001
39.69   0.0001
39.86   0.0001
40.02   0.0000
40.19   0.0000
40.35   0.0000
40.52  -0.0000
40.68  -0.0000
40.85  -0.0000
41.01  -0.0000
41.18  -0.0000
41.34  -0.0000
41.51  -0.0000
41.67  -0.0000
41.84  -0.0000
42.00   0.0000
axes initialized
Cube Completed
Weighting Completed
Weight of 0.000000 0.000000 is 33.238815
```

0	74.989	0.07
1	73.973	-0.00
2	72.957	-0.12
3	71.941	0.03
4	70.925	-0.11
5	69.909	-0.12
6	68.893	-0.13
7	67.877	-0.11
8	66.861	0.21
9	65.845	0.08
10	64.829	0.06
11	63.813	0.20
12	62.798	0.32
13	61.782	0.09
14	60.766	-0.15
15	59.750	0.37
16	58.734	0.18
17	57.718	-0.12
18	56.702	0.15
19	55.686	0.11
20	54.670	0.10
21	53.654	-0.04
22	52.638	0.42
23	51.622	-0.03
24	50.607	-0.06
25	49.591	-0.07
26	48.575	-0.10
27	47.559	-0.17
28	46.543	-0.29
29	45.527	-0.21
30	44.511	-0.17
31	43.495	-0.28
32	42.479	-0.20
33	41.463	0.04
34	40.447	-0.12
35	39.431	-0.10
36	38.415	-0.22
37	37.400	0.06
38	36.384	0.14
39	35.368	0.02
40	34.352	-0.11
41	33.336	0.31
42	32.320	0.00
43	31.304	-0.18
44	30.288	-0.15

45	29.272	-0.10
46	28.256	0.10
47	27.240	0.03
48	26.224	-0.01
49	25.208	-0.03
50	24.193	0.07
51	23.177	0.08
52	22.161	0.13
53	21.145	-0.06
54	20.129	0.32
55	19.113	0.19
56	18.097	-0.09
57	17.081	-0.20
58	16.065	0.07
59	15.049	0.19
60	14.033	0.04
61	13.017	0.14
62	12.002	0.02
63	10.986	-0.27
64	9.970	-0.35
65	8.954	0.03
66	7.938	-0.29
67	6.922	0.06
68	5.906	-0.01
69	4.890	-0.07
70	3.874	-0.27
71	2.858	-0.10
72	1.842	0.07
73	0.826	0.06
74	-0.190	-0.38
75	-1.205	0.02
76	-2.221	-0.24
77	-3.237	-0.14
78	-4.253	0.86
79	-5.269	6.02
80	-6.285	10.72
81	-7.301	14.10
82	-8.317	14.77
83	-9.333	15.13
84	-10.349	14.63
85	-11.365	14.22
86	-12.381	14.58
87	-13.397	14.55
88	-14.412	14.20
89	-15.428	14.23

90	-16.444	14.20
91	-17.460	14.01
92	-18.476	14.28
93	-19.492	14.05
94	-20.508	14.46
95	-21.524	14.13
96	-22.540	14.11
97	-23.556	13.89
98	-24.572	13.82
99	-25.588	13.66
100	-26.603	14.07
101	-27.619	13.71
102	-28.635	13.70
103	-29.651	13.07
104	-30.667	11.97
105	-31.683	9.62
106	-32.699	4.51
107	-33.715	2.37
108	-34.731	1.39
109	-35.747	0.24
110	-36.763	-0.05
111	-37.779	0.12
112	-38.795	-0.28
113	-39.810	0.14
114	-40.826	0.02
115	-41.842	0.08
116	-42.858	0.17
117	-43.874	0.05
118	-44.890	-0.06
119	-45.906	-0.12
120	-46.922	0.10
121	-47.938	0.07
122	-48.954	0.00
123	-49.970	0.03
124	-50.986	-0.18
125	-52.002	0.03
126	-53.017	-0.24
127	-54.033	-0.28
128	-55.049	0.04
129	-56.065	0.01
130	-57.081	-0.09
131	-58.097	-0.10
132	-59.113	0.15
133	-60.129	-0.07
134	-61.145	0.08

```
 135  -62.161   0.07
 136  -63.177   0.14
 137  -64.193   0.09
 138  -65.208   0.21
 139  -66.224  -0.12
 140  -67.240   0.05
 141  -68.256   0.04
 142  -69.272   0.14
 143  -70.288  -0.07
 144  -71.304   0.43
 145  -72.320  -0.08
 146  -73.336   0.18
 147  -74.352  -0.12
 148  -75.368  -0.06
 149  -76.384  -0.20
 150  -77.400  -0.02
 151  -78.415  -0.13
 152  -79.431   0.09
 153  -80.447  -0.01
 154  -81.463   0.14
 155  -82.479   0.16
 156  -83.495   0.15
 157  -84.511  -0.02
 158  -85.527  -0.02
 159  -86.543   0.23
 160  -87.559   0.04
 161  -88.575   0.01
 162  -89.591   0.01
 163  -90.606   0.01
 164  -91.622  -0.28
 165  -92.638  -0.18
 166  -93.654  -0.17
 167  -94.670   0.06
 168  -95.686  -0.13
 169  -96.702   0.01
 170  -97.718  -0.15
 171  -98.734  -0.13

STDERR ****************************

Exit Code: 0
```

VIEWING THE DATA CUBE

With the FITS data cube written, we may now wish to view the result. Fortunately, *astropy* provides useful routines to read FITS data cubes and display them in an appropriate World Coordinate System. The CubeReader class provides a simple interface to these routines as well as some methods to manipulate the cube.

Read the Cube

The first code cell illustrates simply reading the cube and showing the header.

The cube axes are labeled 'x', 'y', and 'v', and methods use these labels to extract data from the cube.

In [12]:

```
# Open FITS file using CubeReader
Cube = CubeReader(FitsFileName,1,2,3)
# print the header
Cube.PrintHeader()
# print some information about the cube
Cube.GetInfo()
```

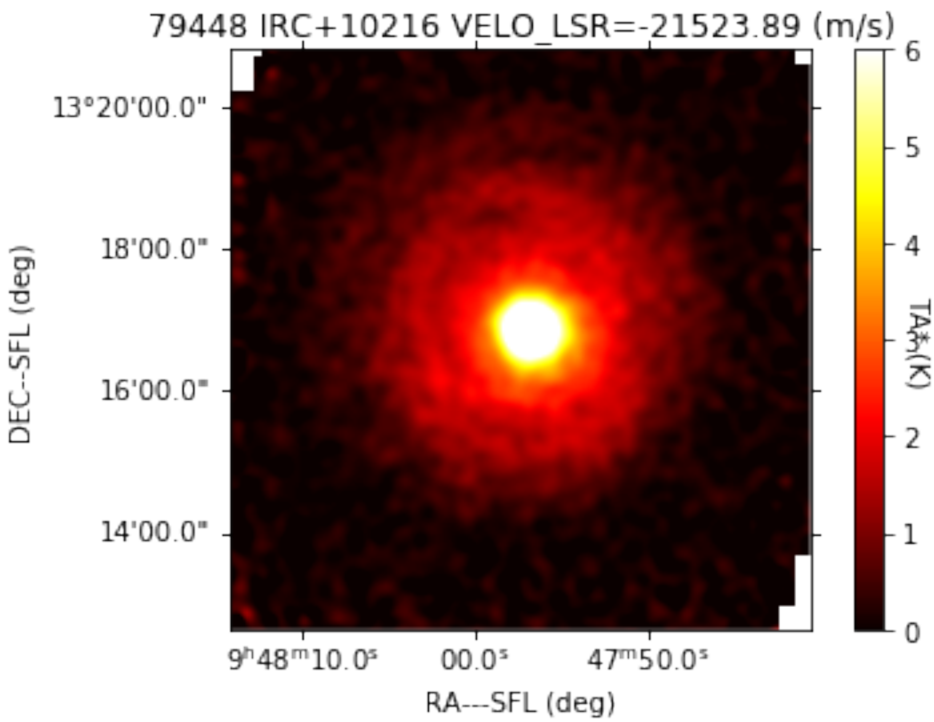
```
SIMPLE  =                    T / file does conform t
o FITS standard
BITPIX  =                  -32 / number of bits per
data pixel
NAXIS   =                    3 / number of data axes
NAXIS1  =                  173 / length of data axis
1
NAXIS2  =                  173 / length of data axis
2
NAXIS3  =                  172 / length of data axis
3
EXTEND  =                    T / FITS dataset may co
ntain extensions
COMMENT   FITS (Flexible Image Transport System) for
mat is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; b
ibcode: 2001A&A...376..359H
TELESCOP= 'LMT     '           /
OBJECT  = 'IRC+10216'          /
OBSNUM  =                79448 /
BUNIT   = 'K       '           /
CTYPE1  = 'RA---SFL'           /
CRVAL1  =             146.9892 / deg
CDELT1  =         -0.001944444 / deg
CRPIX1  =                  86. /
CUNIT1  = 'deg     '           /
CTYPE2  = 'DEC--SFL'           /
CRVAL2  =             13.27877 / deg
CDELT2  =          0.001944444 / deg
CRPIX2  =                  86. /
CUNIT2  = 'deg     '           /
CTYPE3  = 'VELO_LSR'           /
CRVAL3  =             74988.61 / m/s
CDELT3  =            -1015.921 / m/s
CRPIX3  =                   0. /
CUNIT3  = 'm/s     '           /
EQUINOX =                2000. /
RADESYS = 'FK5     '           /
Filename: ./Pipeline.fits
No.    Name      Ver    Type      Cards   Dimensions
Format
  0  PRIMARY       1 PrimaryHDU      30   (173, 173,
172)   float32
```

View a Map from a 2D slice of data from the cube

The next example shows the 2D map of a single spectral channel in the cube. Note that this is a slice of the "v" axis at channel `v_val`.

In [13]:

```
Cube.Slice('v',v_val,colormap=pl.cm.hot,maximum=Cmax,xbegin=xb,xend=xe,ybegin=yb,yend=ye)
```

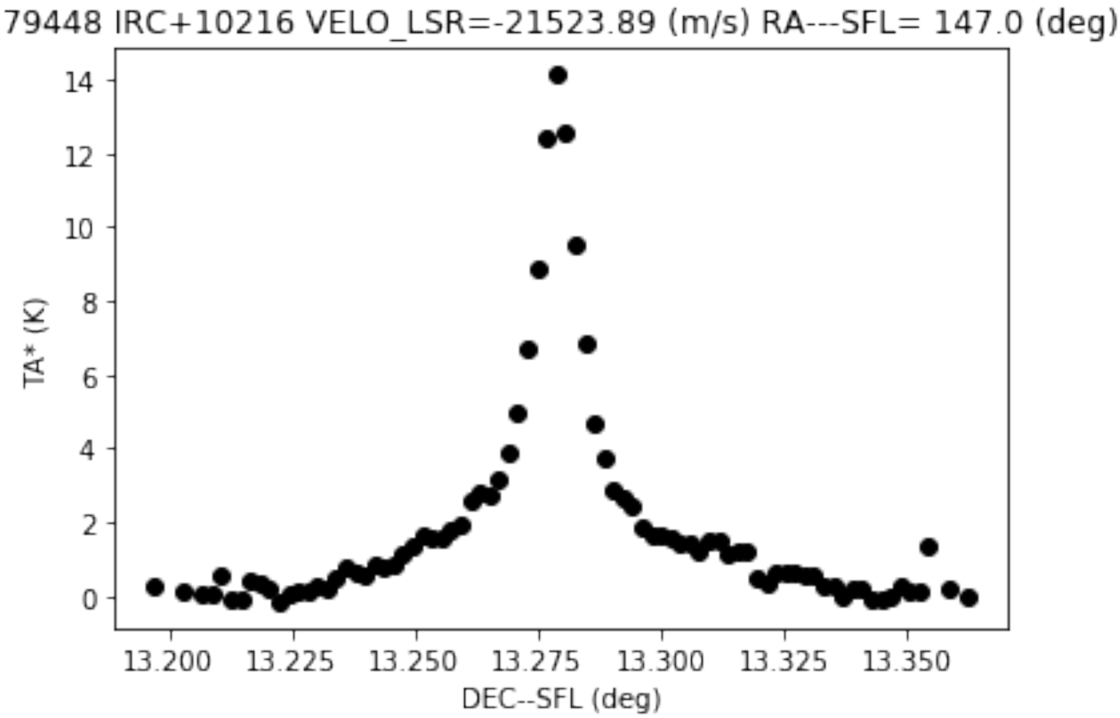


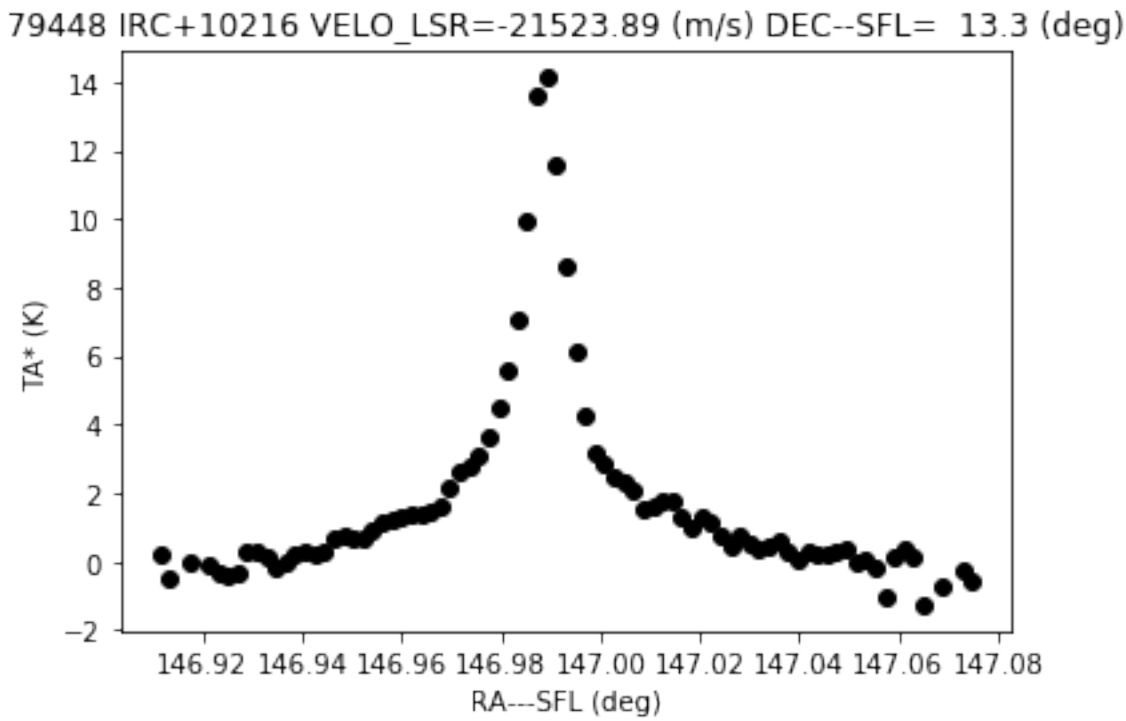
View Line of Data through the Cube

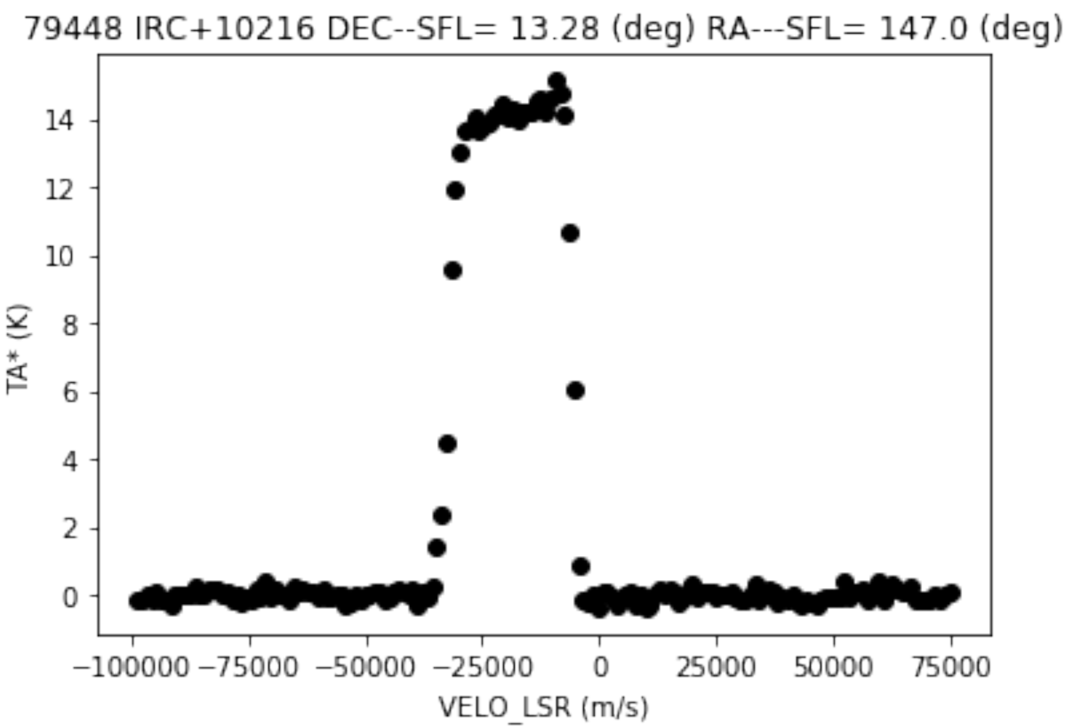
These examples show a single line of data extracted from the cube in each of the three possible directions.

In [14]:

```
# spectral channel v_val plotted for all declinations at ra give
n by x_val
Cube.ShowSpectrum('x',x_val,'v',v_val)

# spectral channel v_val plotted for all ra's at declination giv
en by y_val
Cube.ShowSpectrum('y',y_val,'v',v_val)

# all spectral channels plotted for a single pixel: ra given by
x_val; dec given by y_val
Cube.ShowSpectrum('x',x_val,'y',y_val)
```




79448 IRC+10216 VELO_LSR=-21523.89 (m/s) DEC--SFL= 13.3 (deg)
TA* (K)
RA---SFL (deg)
146.92
146.94
146.96
146.98
147.00
147.02
147.04
147.06
147.08


79448 IRC+10216 DEC--SFL= 13.28 (deg) RA---SFL= 147.0 (deg)
TA* (K)
VELO_LSR (m/s)
−100000
−75000
−50000
−25000
0
25000
50000
75000