orion-muse

William Henney

October 15, 2015

Contents

1 TODO Important things to follow up

1.1 [Fe III] 5270

- This shows lots of wonderful structure in the jet source regions
  - HH529 and counterjet?
- Also strong in NW extension of HH202
- Shows jet that can be maybe linked with HH203/204
- Our WFC3 469N filter shows a small field of this around Orion S
- MUSE spectrum starts at 4595, so we should also have [Fe III] 4702, 4658, which are of similar brightness to 5270, plus a host of weaker ones - see the Manuel notes.
  - Yes, they are seen - 4658 is the best

1.2 O II complex at 4650

- 4649 and 4651 are severely blended
  - Disentangling these is vital for getting an O II density diagnostic
- We have to sum over a large area to get enough s/n to fit for all the O II components
- We could maybe use the mean wavelength of the 4649+51 blend as a proxy for the 4649/51 ratio, but we would have to correct for the kinematics, using perhaps [O III] or, better yet, an unblended line of the same multiplet
  - But the only one is 4676 and that is too weak
  - We can't really use 4639+42 because that is blended with N II 4643 and N III 4641

1.3 [C I] 8727.13

- This has a different morphology than anything else!
- Redshifted filament pointing down SSW from Orion S

1.4 [Ar III] 7135.78

- This is excellent for velocity mapping
- The velocity resolution is better at longer wavelengths
- You can see blue-shifted and red-shifted flows easily

1.5 Weaker lines that might be interesting

- Ne I 8892.22 - similar to O I
- Ca I] 9052.16 but 9095.09 is missing so maybe something else
- [O I] 5577.31 need to remove sky line but then there are some interesting little spots like HH 201
- 5906 very weak line - I had classified it as Si I 5906.22, 5906.15, 5906.418, 5906.92 but this seems very unlikely since it is not seen in any of the low-ionization parts, only near the Trapezium, most strongly in the SW compact bar

1.6 Using the absorption lines

- The trouble here is that many of the best He II lines are bluer than the spectral range
- He II 4686 works well
  - Deepest in th2A
  - One problem is that it is phase-dependent in th1C
- He II 5411 has some potential, but is contaminated by [Fe III]
- O II 4650 is seen in absorbtion right on th2A, but in the nebula it is swamped by the ORL emission lines
- C IV 5801.35, 5811.97 are clearly seen in th1C spectrum and much weaker in th2A, absent in other trapezium stars

  - Unfortunately, they are very weak in the nebula
  - Requires integration over 15x15 arcsec box to have much s/n
- N III 6633.9 is very interesting
  - Has absorption depth of 0.08 in Orion S region
  - But not seen in any OB stars
  - Seen in th1E, which is G2V spectral type
    * (which Olivares et al 2013) say is not bound to Trapezium
    * And is also eclipsing binary (Morales-Calderón et al 2012)
    * But absorption depth there is only 0.05
  - Conclusion must be that we are seeing scattered light from an embedded yellow supergiant that is leaking out.
  - Some T Tauri stars show a Cr I line at 6630 but that has a very low EW ~0.01 A (Apenzeller et al 1986, Table III)
  - There are also lines around 6480 and 6490 in the nebular scattered light
    * Some stars show a strongish line around 6495
- Not an absorption line, but also possibly stellar
  - Wide (50 Angstrom) bup seen around 8600 Angstrom

1.7 DONE Looking at the strange broad NIR emission bump

- Take the difference or ratio between 8570 A and 8552 A
- Results are very disappointing
  - We see a vague form of the nebula in the ratio image
  - It looks similar to the continuum image, but not exactly the same
  - It doesn't look much like scattered continuum
  - And there is this instrumental tartan pattern superimposed on it
  - And it is also very noisy
  - Certainly not worth bothering with
  - Ratio image is even worse

```
from astropy.io import fits
import numpy as np

cubehdu, = fits.open('muse-hr-data-wavsec6.fits')
continuum = np.nanmean(cubehdu.data[441:446], axis=0)
bump = np.nanmean(cubehdu.data[455:471], axis=0)
fits.PrimaryHDU(header=cubehdu.header, data=bump-continuum).writeto('bump8600-diff.fits
fits.PrimaryHDU(header=cubehdu.header, data=bump/continuum).writeto('bump8600-ratio.fit
```

1.8 Longer wavelength lines

7001.92	O I	3	
7002.23	O I	3	blend
7065.28	He I	2	
7135.78	[Ar III]	1	super strong
7155.14	[Fe II]	4	
7231.34	C II	3	
7236.42	C II	3	
7254.15	O I	3	Also 7254.45, 7254.53
7281.35	He I	3	
7290.3	?	4	possibly [Fe II]
7318.39	[O II]	1	Also 7319.99
7329.66	[O II]	1	Also 7330.73
7377.83	[Ni II]	4	
7411.61	[Ni II]	5	
7442.30	N I	5	
7452.54	[Fe II]	4	
7468.31	N I	4	
7751.10	[Ar III]	1	
7816.13	He I	4	
7890.07	Ca I]	4	
7900	Sky	4	Lots of sky lines
8000	Sky	4	in this spectral
8100	Sky	4	range
8189	Fe I?	4	ID uncertain
8200.36	N I?	5	very weak
8210.72	N I	5	
8216.34	N I	4	
8223.14	N I	4	Strongest component
8243	?	4	O I? or Fe II?
8240+	H I	4	Lots of Paschen lines
8437.96	H I	3	Pa 18
8446.36	O I	2	And 8444.25, 8444.76–
8467.25	H I	2	Pa 17
8502.48	H I	2	Pa 16
8545.38	H I	2	Pa 15
8578.69	[Cl II]	3	
8598.39	H I	2	Pa 14
8600	Bump?	4	Maybe scattered stellar
8616.95	[Fe II]	3	
8665.02	H I	2	Pa 13
8680.28	N I	4	Strongest component
8683.40	N I	4	
8686.15	N I	4	
8703.25	N I	4	
8711.70	N I	4	
8718.83	N I	5	very weak
8727.13	[C I]	4	Different!

2 Extracting lines for comparison with WFC3

- The interesting sections are wavsec0 to wavsec3

2.1 Extract subsets of the full cubes

2.1.1 Transposed cubes

- These are taken from the section cubes that I made down here
  - Origina axis order is RA, Dec, Wav
- They will look like longslit spectra in DS9 - at least that is the idea

1. **TODO** Stack of horizontal slits: Wav, RA, Dec
   - 1 2 3 -> 3 1 2

```
MDIR=~/Source/Montage/bin
$MDIR/mTranspose muse-hr-data-wavsec0.fits muse-hr-hslit-stack-wavsec0.fits 3 1 2
```

   - Note that this takes a long time to run - even longer on my laptop
   - Best to do it in a terminal rather than with babel
   - They are too big (17GB each), so I have moved tham to a `BigFiles/` folder, which I am not synching with my laptop, only with hypatia
2. **TODO** Stack of vertical slits: Wav, Dec, RA
   - 1 2 3 -> 3 2 1
3. Bin them up to make them easier to use
   (a) 1 arcsec pixels: 5x5 binning in RA and Dec
   - First, work directly on the original cube sections
   - FITS order is RA, Dec, Wav
   - Python order is Wav, Dec, RA
   - Original NY, NX is 1476, 1766
     - If we chop off just 1 pixel on each axis, then we are divisible by 5
   - Work on one image at time to simplify things and reduce memory footprint

```
from __future__ import print_function
import sys
import numpy as np
from astropy.io import fits

def rebin_xyimage(im, mx=5, my=5):
    ny, nx = im.shape
    # Shape of new rebinned array
    nny, nnx = ny//my, nx//mx
    # Shave a bit off original array so shape is multiple of m
    # And then concertina each axis to be 4-dimensional
    im4d = im[:nny*my, :nnx*mx].reshape((my, nny, mx, mmx))
    # Average along the mx, my axes
    return np.nanmean(im4d, axis=(0, 2))


def rebin_hdu(hdu, m=(5, 5)):
    mx, my = m                  # FITS axis order
    nv, ny, nx = hdu.data.shape  # Python axis order
    nny, nnx = ny//my, nx//mx
    newdata = np.empty((nv, nny, nnx))
    for k in range(nv):
        print('Rebinning plane {}/{}'.format(k+1, nv))
        newdata[k] = rebin_xyimage(hdu.data[k], mx, my)
    newhdu = fits.PrimaryHDU(header=hdu.header, data=newdata)
    # New pixel deltas are bigger
    newhdu.header['CDELT1'] *= mx
    newhdu.header['CDELT2'] *= my
    # pix=0.5 is left edge of first pixel
    newhdu.header['CRPIX1'] = 0.5 + (newhdu.header['CRPIX1'] - 0.5)/mx
    newhdu.header['CRPIX2'] = 0.5 + (newhdu.header['CRPIX2'] - 0.5)/my

    return newhdu

if __name__ == '__main__':
    try:
        infilename = sys.argv[1]
        m = sys.argv[2:4]
    except IndexError:
        print('Usage:', sys.argv[0], 'FITSFILE BINX BINY')
```

```
hdu = fits.open(fn)['DATA']
suffix = '-rebin{:02d}x{:02d}'.format(*m)
outfilename = infilename.replace('.fits', suffix+'.fits')
rebin_hdu(hdu, m).writeto(outfilename)
```

- These should all be run in an interactive shell
- First, test it with a small file

```
python rebin_datacube.py
```

2.1.2 Spectral windows for each WFC3 filter

- In principal the calibration can be done with just integrating the spectrum over the filter T reponse.
- But we really need to fit Gaussians to the lines
- Note that pysynphot requires Python 2.7
- Also requires PYSYN_CDBS environment variable to be set
  - On linux server

```
export PYSYN_CDBS=/fs/nil/other0/will/CDBS
```

1. List of HST filters to use Export this table to `all-filters-input.tab` with `C-c t e` after any modification.

Instrument	Filter
wfc3	f469n
wfc3	f487n
wfc3	f502n
wfc3	f547m
wfc3	fq575n
wfc3	f656n
wfc3	f658n
wfc3	fq672n
wfc3	f673n
wfc3	fq674n
wfpc2	f502n
wfpc2	f547m
wfpc2	f631n
wfpc2	f656n
wfpc2	f658n
wfpc2	f673n
acs	f658n
acs	f660n
acs	f435w
acs	f555w
acs	f775w
acs	f850lp

Send all tables to linux server

```
rsync -aPq *.tab nil:/fs/nil/other0/will/orion-muse
```

```
def bp_fullname(instrument, filter_):
    if instrument.lower() == 'wfc3':
        return 'wfc3,uvis1,'+filter_.lower()
    elif instrument.lower() == 'acs':
        return 'acs,wfc1,'+filter_.lower()
    elif instrument.lower() == 'wfpc2':
        return 'wfpc2,'+filter_.lower()
    else:
        raise NotImplementedError('Unknown instrument: ' + instrument)
```

2. **DONE** [1/1] Print out the mean wavelength and rectangular width of each filter

```
import pysynphot
from astropy.table import Table
def bp_fullname(instrument, filter_):
    if instrument.lower() == 'wfc3':
        return 'wfc3,uvis1,'+filter_.lower()
    elif instrument.lower() == 'acs':
        return 'acs,wfc1,'+filter_.lower()
    elif instrument.lower() == 'wfpc2':
        return 'wfpc2,'+filter_.lower()
    else:
        raise NotImplementedError('Unknown instrument: ' + instrument)
float_fmt = '{:.2f}'
intab = Table.read('all-filters-input.tab', format='ascii.tab')
outtab = [['Filter', 'Wav0', 'dWav'], None]
for row in intab:
    fn = bp_fullname(row['Instrument'], row['Filter'])
    bp = pysynphot.ObsBandpass(fn)
    outtab.append([fn, float_fmt.format(bp.avgwave()), float_fmt.format(bp.rectwid
```

- ☒ Test that this works on linux server

3. Decomposing the components that go into the throughput curve

   This is done in `wfc3-throughput-components.py`

   Use the STScI python install for a change

   ```
   source ~/.bash_profile
   ur_setup
   export PYSYN_CDBS=/Users/will/Dropbox/CDBS
   python wfc3-throughput-components.py
   ```

   This is a plot of all the components that multiply together to make the filter throughput:

   hst_ota Optical Telescope Assembly. I think this is the primary mirror efficiency, accounting for fraction of circular area that is obscured by secondary. Roughly constant at about 0.65

   wfc3_uvis_ccd1 Efficiency of CCD, roughly constant at ~ 0.87

   - Note that CCD2 is extremely similar. The difference is less than 0.5%

wfc3_uvis_owin Outer window transmission, roughly 0.95

wfc3_uvis_cor Correction based on white dwarf photometry. Roughly 1.18 but falling to red.

wfc3_uvis_mir1 Internal camera mirror efficiency, roughly 0.9

wfc3_uvis_mir2 Another mirror efficiency, roughly 0.9

wfc3_uvis_iwin Internal window, roughly 0.95

wfc3_pom_001 Pick Off Mirror (45 deg mirror that diverts light into instrument), roughly 0.88

wfc3_uvis_f547m The filter itself, roughly 0.85

Multiplying them all together gives the total transmission of `0.364878642843`, which matches what is expected for the total bandpass.

(a) **DONE** Variation with time

- According to Appendix B of the pysynphot docs we can ask for the filter throughput for a particular MJD using e.g., 'wfc3,uvis1,f658n,mjd#49486'
- Today's Julian date is `57307`
- Orion S observations were around MJD=55933
- Plot various dates in `wfc3-throughput-evolution.py`

```
source ~/.bash_profile
ur_setup
export PYSYN_CDBS=/Users/will/Dropbox/CDBS
python wfc3-throughput-evolution.py
```

Upshot is that there is no discernible difference with time, and also that the Quantum Yield Correction makes no difference at the shortest wavelengths that we are interested in (4700 Angstrom), `2.7.10`

4. **TODO** Converting surface brightness to predicted counts

- Note that `bp.primary_area` is given as 45238.93416, which must be in sq cm. This is same as `45238.9342117`
- The units of the muse data is given as '10**(-20)*erg/s/cm**2/Angstrom'
  - This must be per pixel, I assume
  - Each pixel is 0.2 arcsec square, so this is `9.40175455274e-13` steradian

- ☐ We have summed this in wavelength, but we really should have multiplied by lambda first to convert from energy to photon units
  - And while we are at it we can use the C_{WFC3} to put it in electron/s

5. Air to vacuum wavelength conversion
   - This depends on refractive index of air, given by the following function
   - To convert air -> vacuum we multiply the wavelengths by the refractive index
   -

```
from astropy import units as u
def air_refractive_index(wav):
    """Equation (65) of Greisen et al 2006 for the refractive index of air
at STP.  Input wavelength 'wav' should be in microns or in any
'astropy.units' unit. It does not matter if 'wav' is on air or vacuum
scale

    """
    try:
        # Convert to microns if necessary
        wavm = wav.to(u.micron).value
    except AttributeError:
        # Assume already in microns
        wavm = wav
    return 1.0 + 1e-6*(287.6155 + 1.62887/wavm**2 + 0.01360/wavm**4)
```

6. **TODO** [5/5] Process spectral windows for each filter
   - This could be the last step that we would have to run on the server
   - If the files are small enough then they can be copied over to the macs
   - Each of the following snippets is run interactively on the server

   (a) **DONE** Imports

```
from astropy.io import fits
from astropy import wcs
from astropy.table import Table
import pysynphot
import numpy as np
```

(b) **DONE** Read FITS cube

```
hdulist = fits.open('DATA/DATACUBEFINALuser_20140216T010259_78380e1d.fits')
```

(c) **DONE** Set up a vacuum wavelength scale

```
from astropy import units as u
def air_refractive_index(wav):
    """Equation (65) of Greisen et al 2006 for the refractive index of air
at STP.  Input wavelength 'wav' should be in microns or in any
'astropy.units' unit. It does not matter if 'wav' is on air or vacuum
scale

    """
    try:
        # Convert to microns if necessary
        wavm = wav.to(u.micron).value
    except AttributeError:
        # Assume already in microns
        wavm = wav
    return 1.0 + 1e-6*(287.6155 + 1.62887/wavm**2 + 0.01360/wavm**4)

w = wcs.WCS(hdulist['DATA'].header)
NV, NY, NX = hdulist['DATA'].data.shape
# construct array of observed air wavelengths (at image center to be safe)
_, _, wavs = w.all_pix2world([NX/2]*NV, [NY/2]*NV, np.arange(NV), 0)
# Make dimensional
wavs *= u.m
# Convert to vacuum scale
wavs *= air_refractive_index(wavs)
```

(d) **DONE** Read in the table of filters

```
intab = Table.read('all-filters-input.tab', format='ascii.tab')
```

(e) **DONE** Extract the windows for each filter

```
def bp_fullname(instrument, filter_):
```

```
        if instrument.lower() == 'wfc3':
            return 'wfc3,uvis1,'+filter_.lower()
        elif instrument.lower() == 'acs':
            return 'acs,wfc1,'+filter_.lower()
        elif instrument.lower() == 'wfpc2':
            return 'wfpc2,'+filter_.lower()
        else:
            raise NotImplementedError('Unknown instrument: ' + instrument)
    for row in intab:
        bpname = bp_fullname(row['Instrument'], row['Filter'])
        bp = pysynphot.ObsBandpass(bpname)
        # extend a full rectwidth either side of the average wavelength to fit it
        wav_window = bp.avgwave() + bp.rectwidth()*np.array([-1, 1])
        # Add in the units (all are in Angstrom I hope)
        assert bp.waveunits.name == 'angstrom'
        wav_window *= u.Angstrom
        # convert to air wavelengths to agree with the WCS
        wav_window /= air_refractive_index(wav_window)
        # Now convert to fractional pixel coordinates
        _, _, [k1, k2] = w.all_world2pix([0, 0], [0, 0], wav_window.to(u.m), 0)
        # smallest slice that covers the window
        wavslice = slice(int(k1), int(k2) + 2)
        # tuple of slices for the 3 cube axes (in numpy array order: V, Y, X)
        cubeslices = [wavslice, slice(None, None), slice(None, None)]

        newhdr = hdulist['DATA'].header.copy()
        newhdr.update(w.slice(cubeslices).to_header())

        # Make a new HDUlist for the windowed spectrum and write it out
        fits.HDUList(
            [fits.PrimaryHDU(header=hdulist[0].header, data=None),
             fits.ImageHDU(header=newhdr, data=hdulist['DATA'].data[cubeslices])
            ]
        ).writeto('muse-hr-window-{}-{}.fits'.format(row['Instrument'], row['Filt
```

7. Cleaning up the window FITS files for DS9 For some reason, ds9 does not like the wavelength WCS, so we will try and fix it:

   - Put the physical scales in the CDELTi instead of in the PCi_j
   - Put it in angstrom instead of m

- That's it to start with

```
import sys
from astropy.io import fits
def clean_up_wav_wcs(filename):
    hdulist = fits.open(filename, mode='update')
    for hdu in hdulist:
        if hdu.header.get('CUNIT3') == 'm':
            # Change to Angstrom
            hdu.header['PC3_3'] *= 1e10
            hdu.header['CRVAL3'] *= 1e10
            hdu.header['CUNIT3'] = 'Angstrom'
            # And move scales to CDELT
            for i in '123':
                CDELTi = 'CDELT'+i
                # Sanity check
                assert hdu.header.get(CDELTi) == 1.0
                PCi_j = 'PC{0}_{0}'.format(i)
                hdu.header[CDELTi], hdu.header[PCi_j] = hdu.header[PCi_j], hdu.hea
    hdulist.flush()


if __name__ == '__main__':
    try:
        fn = sys.argv[1]
        clean_up_wav_wcs(fn)
    except IndexError:
        print('Usage:', sys.argv[0], 'FITSFILE')
```

Export with `C-u C-c C-v C-t`

Test it on the WFC3 f656n file

```
python clean_up_wav_wcs.py muse-hr-window-wfc3-fq674n.fits
```

That seemed to work

```
python clean_up_wav_wcs.py muse-hr-window-wfc3-f487n.fits
```

8. **DONE** Convert from erg/cm2/s/Angstrom to electron/s

- The fundamental equation is $R_j = C_{WFC3} \int \lambda I_\lambda T_\lambda d\lambda$
  - Where $C_{WFC3} = 0.0840241$ if λ is in Å
- So, we need to multiply by lambda when we do the flattening
- Also, we need to get from MUSE's flux-per-pixel to brightness (per-steradian)
  - This means we divide by the MUSE pixel area of 9.40175455274e-13 sr
- AND we need to multiply by the MUSE bin width in Å
- Question is, do we apply this normalization to the `transwin` cubes?
  - Best not, so as to minimze churn of large files on Dropbox

```
from astropy import units as u
WFC3_CONSTANT = 0.0840241
MUSE_FLUX_UNITS = 1e-20
MUSE_PIXEL_AREA_SR = (0.2*u.arcsec).to(u.radian)**2
```

9. Fold the spectra through each filter to get simulated images This does not have to be done on the server any more

```
from __future__ import print_function
import sys
from astropy.io import fits
from astropy import wcs
from astropy.table import Table
import pysynphot
import numpy as np
from astropy import units as u
def air_refractive_index(wav):
    """Equation (65) of Greisen et al 2006 for the refractive index of air
at STP.  Input wavelength 'wav' should be in microns or in any
'astropy.units' unit. It does not matter if 'wav' is on air or vacuum
scale

    """
    try:
        # Convert to microns if necessary
```

```
        wavm = wav.to(u.micron).value
    except AttributeError:
        # Assume already in microns
        wavm = wav
    return 1.0 + 1e-6*(287.6155 + 1.62887/wavm**2 + 0.01360/wavm**4)

def bp_fullname(instrument, filter_):
    if instrument.lower() == 'wfc3':
        return 'wfc3,uvis1,'+filter_.lower()
    elif instrument.lower() == 'acs':
        return 'acs,wfc1,'+filter_.lower()
    elif instrument.lower() == 'wfpc2':
        return 'wfpc2,'+filter_.lower()
    else:
        raise NotImplementedError('Unknown instrument: ' + instrument)
from astropy import units as u
WFC3_CONSTANT = 0.0840241
MUSE_FLUX_UNITS = 1e-20
MUSE_PIXEL_AREA_SR = (0.2*u.arcsec).to(u.radian)**2


def bandpass_flatten(instrument, bpname):
    filename = 'muse-hr-window-{}-{}.fits'.format(instrument, bpname)
    hdulist = fits.open(filename)
    hdu = hdulist['DATA']
    w = wcs.WCS(hdu.header)
    NV, NY, NX = hdu.data.shape
    # construct array of observed air wavelengths (at image center to be safe)
    _, _, wavs = w.all_pix2world([NX/2]*NV, [NY/2]*NV, np.arange(NV), 0)
    # Make dimensional
    wavs *= u.m
    # Convert to vacuum scale
    wavs *= air_refractive_index(wavs)

    # Get bandpass for filter
    fn = bp_fullname(instrument, bpname)
    bp = pysynphot.ObsBandpass(fn)
    # Calculate transmission curve at the observed wavelengths
    T = bp(wavs.to(u.Angstrom).value)
    # Weight by transmission curve and save that
```

```
    hdu.data *= T[:, None, None]
    hdulist.writeto(filename.replace('-window-', '-transwin-'), clobber=True)
    # Integrate over wavelength, already weighted by transmission curve. But
    # now need to multiply by wavelength, put in brightness units, and
    # convert to WFC3 electron/s/pixel
    hdu.data *= WFC3_CONSTANT*MUSE_FLUX_UNITS/MUSE_PIXEL_AREA_SR
    hdu.data *= wavs.to(u.Angstrom).value[:, None, None]
    hdu.data = hdu.header['CDELT3']*np.sum(hdu.data, axis=0)
    hdu.header['BUNIT'] = 'electron/s/(0.03962 arcsec)**2'
    hdulist.writeto(filename.replace('-window-', '-image-'), clobber=True)

if __name__ == '__main__':
    try:
        instrument, bpname = sys.argv[1:]
        bandpass_flatten(instrument, bpname)
    except IndexError:
        print('Usage:', sys.argv[0], 'INSTRUMENT FILTER')
```

New example of use, using STSCI python on laptop

```
source ~/.bash_profile
ur_setup
export PYSYN_CDBS=/Users/will/Dropbox/CDBS
python filter-flatten.py wfc3 fq575n
```

Check the same one using the Anaconda py27 on hypatia, but make sure that we are using the same version of CDBS

```
source activate py27
export PYSYN_CDBS=/Users/will/Dropbox/CDBS
python filter-flatten.py wfc3 fq575n
```

Exactly the same, which is a heartening.

Now do it for all the filters

```
source activate py27
export PYSYN_CDBS=/Users/will/Dropbox/CDBS
FILTERS="f469n f487n f502n f547m fq575n f656n f658n fq672n f673n fq674n"
for f in $FILTERS; do
    echo Flattening $f
```

```
    python filter-flatten.py wfc3 $f
done
```

10. Comparing profiles by eye

    - Took some profiles by eye on WFC3 and MUSE images of F547M
      - Cannot compare them in DS9 because the spatial axis is written in pixels, which are different sizes
    - `muse-f547m-cut.dat`
    - `wfc3-f547m-cut.dat`

2.2 DONE [4/4] Compare the real and predicted count-rate images on a common grid

- We want to put everything on the MUSE pixel grid, since that will make smaller files by a factor of `25.4818555174`
- We could use
  1. astrodrizzle
     - acs-ramp-filters.org
  2. montage
     - `https://montage-wrapper.readthedocs.org`
     - I had already done that in the t-squared project
     - And I hadn't even used the python bindings

2.2.1 DONE Check that I have a working Montage installation

- I already have version 3.3 but version 4 is out
- Cloning from github into `file:///Users/will/Source/Montage/`
- Compiled with `make -j8` - that was fast!

2.2.2 DONE Testing out Montage

```
PATH=$PATH:~/Source/Montage/bin
err=$(mProjectPP --help)
echo $err
```

2.2.3 DONE Script to resample WFC3 image onto MUSE grid

- Header for MUSE full frame grid: `muse-full-frame.hdr`
- This does not expand the WFC3 image beyond its original borders
  - So we will have to extract a section of the MUSE image for comparison
  - On the other hand, it does maintain the same reference pixel
    - But with different values of CRPIX because the image lower left corner is different
    - So it has the same values of CRVAL
    - This will make it easy to slice the MUSE image
- Smoothing needs to be improved
  - No smoothing is too little
  - The `s120` images that I already have are too much (this was 1.2 arcsec I assume)
  - In the t2 notes I calculated FWHM of 4 pixels = 0.8 arcsec
  - Actually 0.7 arcsec was better - this is now done in the orion-t2.org notes

```
F=$1
MDIR=~/Source/Montage/bin
TDIR=~/Work/RubinWFC3/Tsquared
$MDIR/mProjectPP -h 0 -X $TDIR/full_${F}-s070.fits wfc3-resample-muse-$F.fits muse-full
```

Test with a single image

```
time sh wfc3-resample-to-muse.sh F547M 2>&1
```

Do all of the images

```
FILTERS="f469n f487n f502n f547m fq575n f656n f658n fq672n f673n fq674n"
for f in $FILTERS; do
    echo "Resampling $f"
    time sh wfc3-resample-to-muse.sh $f
done
```

2.2.4 DONE [2/2] Crop MUSE image to the WFC3 field

- ☒ This is the final step required before we can do things like take ratio maps or calculate 2d histogram images
- ☒ *[2015-10-15 Thu]* Also, write out the integrated spectrum times filter throughput for the cropped region

```
import sys
import numpy as np
from astropy.io import fits
from astropy.wcs import WCS, WCSSUB_SPECTRAL
import astropy.units as u

def crop_muse_to_wfc3(fid):
    """Cut out a section of the MUSE image to match the WFC3 field"""
    wname = 'wfc3-resample-muse-{}.fits'.format(fid)
    mname = 'muse-hr-image-wfc3-{}.fits'.format(fid)
    whdu = fits.open(wname)[0]
    mhdu = fits.open(mname)['DATA']
    # Also get the spectral data cube multiplied by filter throughput
    shdu = fits.open(mname.replace('-image-', '-transwin-'))['DATA']
    wcs_w = WCS(whdu.header).celestial
    wcs_m = WCS(mhdu.header).celestial
    # Check that the two images have the same reference point in RA, DEC
    assert np.all(wcs_w.wcs.crval == wcs_m.wcs.crval)
    # And that the pixel scales are the same
    assert np.all(wcs_w.wcs.cdelt == wcs_m.wcs.cdelt)
    assert np.all(wcs_w.wcs.pc == wcs_m.wcs.pc)

    # The shapes of the two grids: (nx, ny) in FITS axis order
    shape_w = np.array([whdu.header['NAXIS1'], whdu.header['NAXIS2']])
    shape_m = np.array([mhdu.header['NAXIS1'], mhdu.header['NAXIS2']])

    # The difference in CRPIX values tells us the start indices (i, j)
    # for the crop window on the MUSE grid. Note that this is in
    # zero-based array indices
    start = wcs_m.wcs.crpix - wcs_w.wcs.crpix
    # The stop indices for the crop window
    stop = start + shape_w
```

```
# Shift 1 pixel to the right to do a coarse alignment correction
start[0] += 1
stop[0] += 1

# Check that these are within bounds of the original MUSE grid
assert np.all(start >= 0.0)
assert np.all(stop < shape_m)

# Crop the MUSE data array to the start:stop indices, remembering
# that python axis order is backwards with respect to FITS axis
# order
mhdu.data = mhdu.data[start[1]:stop[1], start[0]:stop[0]]

# And copy the WFC3 wcs into the new MUSE header
mhdu.header.update(wcs_w.to_header())

# Write out the new cropped MUSE image
oname = mname.replace('-image-', '-cropimage-')
mhdu.writeto(oname, clobber=True)

# Finally, as a bonus, calculate the 1-D average spectrum from the cube
spec = np.nanmean(shdu.data[:, start[1]:stop[1], start[0]:stop[0]], axis=(-1, -2))

# Convert from 1e-20 flux-per-pixel to surface brightness units (flux per sr)
pixel_area_sr = np.product(np.abs(wcs_m.wcs.cdelt))*(u.deg.to(u.radian))**2
spec *= 1e-20/pixel_area_sr
# extract only the spectral part of the cube's WCS
wcs_s = WCS(shdu.header).sub([WCSSUB_SPECTRAL])
oshdu = fits.PrimaryHDU(header=wcs_s.to_header(), data=spec)
oshdu.header['BUNIT'] = 'erg/s/cm**2/sr/Angstrom'
# Fix up the wavelngth units to angstrom
oshdu.header['CDELT1'] *= 1e10
oshdu.header['CRVAL1'] *= 1e10
oshdu.header['CUNIT1'] = 'Angstrom'
# And record the window from the MUSE full field that was extracted
oshdu.header['MUSE_X1'] = start[0] + 1, 'Extracted window: start X pixel'
oshdu.header['MUSE_X2'] = stop[0] + 1, 'Extracted window: stop X pixel'
oshdu.header['MUSE_Y1'] = start[1] + 1, 'Extracted window: start Y pixel'
oshdu.header['MUSE_Y2'] = stop[0] + 1, 'Extracted window: stop Y pixel'
oshdu.writeto(mname.replace('-image-', '-cropspec1d-'), clobber=True)
```

```
    return oname


if __name__ == '__main__':
    try:
        filter_id = sys.argv[1]
    except:
        print('Usage:', sys.argv[0], 'FILTER')

    print(crop_muse_to_wfc3(filter_id))
```

```
python crop_muse.py f547m
```

```
FILTERS="f469n f487n f502n f547m fq575n f656n f658n fq672n f673n fq674n"
for f in $FILTERS; do
    time python crop_muse.py $f
done
```

2.3 Plot the cropped 1D spectra

```
from __future__ import print_function
import sys
from astropy.io import fits
from astropy.wcs import WCS
from astropy import units as u
from matplotlib import pyplot as plt
import seaborn as sns

def plot_1d_spec_from_fits(fn, ax, fontsize=None):
    """Plots spectrum from filename `fn` onto pre-existing axis `ax`"""
    hdu = fits.open(fn)[0]
    spec = hdu.data/1e-3
    w = WCS(hdu.header)
    nwav = len(spec)
    wavs, = w.all_pix2world(range(nwav), 0)
    wavs *= u.m.to(u.Angstrom)
    #ax.plot(wavs, spec, drawstyle='steps-mid')
    ax.bar(wavs, spec, align='center', linewidth=0)
    ax.set_xlim(wavs.min(), wavs.max())
```

```
    ax.set_xlabel('Observed Air Wavelength, Angstrom', fontsize=fontsize)
    ax.set_ylabel('Filter Throughput x Brightness\n 0.001 erg/s/cm^2/sr/Angstrom', font


if __name__ == '__main__':
    try:
        filt = sys.argv[1]
    except IndexError:
        print('Usage:', sys.argv[0], 'FILTER')
    fig, ax = plt.subplots(1, 1)
    fn = 'muse-hr-cropspec1d-wfc3-{}.fits'.format(filt)
    plot_1d_spec_from_fits(fn, ax)
    # ax.set_yscale('log')
    # ax.set_ylim(1e-7, None)
    fig.savefig(sys.argv[0].replace('.py', '-test-{}.pdf'.format(filt)))
```

```
FILTERS="f469n f487n f502n f547m fq575n f656n f658n fq672n f673n fq674n"
for f in $FILTERS; do
    python specplot1d_utils.py $f
done
```

2.4 Visualizations of throughput calibration quality

The final count-rate images to be compared are

Smoothed WFC3 wfc3-resample-muse-FILTER.fits

Cropped MUSE muse-hr-cropimage-wfc3-FILTER.fits

2.4.1 Ratios of the images

- This will allow us to see how important misalignment is, and if there are any spatial trends

```
from astropy.io import fits

filters_ = ["FQ575N", "FQ672N", "FQ674N", "F673N", "F469N",
            "F487N", "F656N", "F658N", "F547M", "F502N"]

def divide_fits_images(name1, name2, outname):
    hdu1 = fits.open(name1)[0]
```

```
    hdu2 = fits.open(name2)['DATA']
    fits.PrimaryHDU(header=hdu1.header, data=hdu1.data/hdu2.data).writeto(outname, clob

if __name__ == '__main__':
    for f in filters_:
        divide_fits_images(
            'wfc3-resample-muse-{}.fits'.format(f),
            'muse-hr-cropimage-wfc3-{}.fits'.format(f),
            'wfc3-over-muse-calib-ratio-{}.fits'.format(f)
        )
```

2.4.2 Two-d histogram of WFC3 vs MUSE-predicted count rates

```
from __future__ import print_function
import numpy as np
from astropy.io import fits
from astropy.convolution import convolve, Gaussian2DKernel
from matplotlib import pyplot as plt
import seaborn as sns
from specplot1d_utils import plot_1d_spec_from_fits

maxcount = {
    "fq575n": 0.4,
    "fq672n": 0.6,
    "fq674n": 0.75,
    "f673n" : 2.5,
    "f469n" : 0.5,
    "f487n" : 10.0,
    "f656n" : 40.0,
    "f658n" : 11.0,
    "f547m" : 7.0,
    "f502n" : 20.0,
}
GAMMA = 2.0

cmap = sns.light_palette((260, 50, 30), input="husl", as_cmap=True)
# cmap = plt.cm.gray_r

def histogram_calib_images(f, vmax=1.0):
    name1 = 'wfc3-resample-muse-{}.fits'.format(f)
```

```
name2 = 'muse-hr-cropimage-wfc3-{}.fits'.format(f)
pltname = 'wfc3-vs-muse-calib-{}.pdf'.format(f)
hdu1 = fits.open(name1)[0]
hdu2 = fits.open(name2)['DATA']
hduc = fits.open('wfc3-resample-muse-f547m.fits')[0]
x, y = hdu2.data, hdu1.data
xmin, xmax = ymin, ymax = 0.0, vmax
ew = y/hduc.data
# mask out silly values
m = np.isfinite(x) & np.isfinite(y/x) & (np.abs(np.log10(y/x)) < 1.0)
H, xedges, yedges = np.histogram2d(x[m], y[m], 200,
                                   [[xmin, xmax], [ymin, ymax]],
                                   weights=y[m])
# Fit a straight line
mm = m & (x > 0.05*xmax) & (y > 0.05*ymax) & (x < 0.5*xmax) & (y < 0.5*ymax) & (np
# First, linear fit y(x) = m x + c
y_x_linfit = np.polyfit(x[mm], y[mm], 1, w=y[mm])
# Second, linear fit x(y) = m y + c
x_y_linfit = np.polyfit(y[mm], x[mm], 1, w=y[mm])
# Convert this from x(y) -> y(x)
# If x = m y + c, then y = (1/m) x - c/m
y_x_altfit = np.array([1./x_y_linfit[0], -x_y_linfit[1]/x_y_linfit[0]])
# Take average and spread of these two fits
y_x_bestfit = 0.5*(y_x_linfit + y_x_altfit)
y_x_errfit = 0.5*np.abs(y_x_linfit - y_x_altfit)

pbest =  np.poly1d(y_x_bestfit)

# H = convolve(H, Gaussian2DKernel(1.0))
fitcolor = (1.0, 0.5, 0.0)
fitlabel = "y = ({:.2f} +/- {:.2f}) x + ({:.2f} +/- {:.2f})".format(
    y_x_bestfit[0], y_x_errfit[0], y_x_bestfit[1], y_x_errfit[1])
fig, ax = plt.subplots(1, 1)
ax.imshow((H.T)**(1.0/GAMMA), extent=[xmin, xmax, ymin, ymax],
          interpolation='none', aspect='auto', origin='lower',
          cmap=cmap, alpha=1.0)
ax.plot([0.0, x[m].max()], [0.0, x[m].max()], '-', alpha=1.0,
        lw=1, c='w', label=None)
ax.plot([0.0, x[m].max()], pbest([0.0, x[m].max()]), '-', alpha=1.0,
        lw=1, c=fitcolor, label=fitlabel)
```

```
leg = ax.legend(loc='upper left', title='Linear Fit', frameon=True, fancybox=True)
leg.get_title().set_fontsize('small')
ax.set_ylabel(
    'Observed WFC3 {} count rate, electron/s/pixel'.format(f.upper()))
ax.set_xlabel(
    'MUSE-predicted WFC3 {} count rate, electron/s/pixel'.format(f.upper()))
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)

# Now do 1-D histogram of the deviations from the model
ratio = y/pbest(x)
if f == 'f547m':
    # Divide into high and low continuum counts
    s = 'counts'
    mlo = (y < y[m].mean()) & m
    mhi = (y >= y[m].mean()) & m
else:
    # Divide into high and low EW
    s = f.upper() + '/F547M'
    mlo = (ew < np.median(ew[m])) & m
    mhi = (ew >= np.median(ew[m])) & m

assert mlo.sum() > 0, f

# inset axis at the top left
ax2 = fig.add_axes([0.2, 0.55, 0.25, 0.25])
ax2.hist(ratio[mlo], bins=100, range=(0.5, 1.5),
         normed=True, weights=y[mlo], alpha=0.7, label='Low '+s)
ax2.hist(ratio[mhi], bins=100, range=(0.5, 1.5),
         normed=True, weights=y[mhi], color='red', alpha=0.3, label='High '+s)
ax2.set_xlim(0.5, 1.5)
# leave more space at top
y1, y2 = ax2.get_ylim()
y2 *= 1.2
ax2.set_ylim(y1, y2)
ax2.legend(loc='upper left', fontsize='xx-small')
ax2.tick_params(labelleft=False, labelsize='x-small')
ax2.set_xlabel('(Observed Counts) / (Linear Fit)', fontsize='xx-small')
ax2.set_ylabel('Weighted PDF Histograms', fontsize='xx-small')
```

```
#    ax2.set_title('PDF', fontsize='x-small')

    # inset axis at the bottom right
    ax3 = fig.add_axes([0.6, 0.2, 0.3, 0.3])
    fn = 'muse-hr-cropspec1d-wfc3-{}.fits'.format(f)
    plot_1d_spec_from_fits(fn, ax3, fontsize='xx-small')
    ax3.tick_params(labelsize='xx-small')
    ax3.set_title(f.upper())


    fig.set_size_inches(7, 7)
    fig.savefig(pltname)

    return [pltname, fitlabel]


if __name__ == '__main__':
    for f, vmax in maxcount.items():
        print(histogram_calib_images(f, vmax))
```

```
python histocalib.py
```

2.4.3 Summary of calibration results

- Calibration constant is unity in most cases!
  - Exceptions are
    F469N slope = 0.94
    F673N slope = 0.97
    F547M intercept = -0.11
- Also no evidence of trend with EW

2.5 Useful scripts

```
open -n -a SAOImage\ DS9 --args -title $DS9
sleep 1
xpaset -p $DS9 view buttons no
xpaset -p $DS9 frame delete all

xpaset -p ds9 frame new
xpaset -p ds9 fits $PWD/muse-hr-data-wavsec3.fits
```

3 Exploring the data cubes

3.1 Original data locations

At CRyA in `/fs/nil/other0/will/orion-muse/DATA`

LR 1.25 Angstrom sampling: DATACUBEFINALuser_20140216T010259_cf767044.fits

HR 0.85 Angstrom sampling: DATACUBEFINALuser_20140216T010259_78380e1d.fits

3.1.1 LR cube

- Dimensions:
  - NV = 3818
  - NY = 1476
  - NX = 1766
- Scales:
  - Spatial: 0.2 arcsec
  - Wavelength: 1.25 Ang

1. Reading in the cube

   ```
   from astropy.io import fits
   from astropy import wcs
   import numpy as np
   hdulist = fits.open('DATA/DATACUBEFINALuser_20140216T010259_cf767044.fits')
   cube = hdulist['DATA']
   ```

   Note that this does not read the full data cube (40 GB) into memory unless we need to do something with it.

2. Extracting the Orion S region

   - To start with, we will look at a 300x300 box centered on (1050, 550)
   - This is more or less the quad filter region

   ```
   subcube = cube.data[:, 400:700, 900:1200]
   spec = np.nansum(np.nansum(subcube, axis=-1), axis=-1)
   spechdu = fits.PrimaryHDU(header=cube.header, data=spec.reshape((3818, 1, 1)))
   spechdu.writeto('subcube-spec.fits')
   ```

- So this gives the summed spectrum of the region
- Note that I did a reshape on the array so that wavelength is still the 3rd FITS axis. So that the header WCS keywords don't need changing

```
rsync -avzP nil:/fs/nil/other0/will/orion-muse/subcube-spec.fits .
```

- The spectrum shows up as s single pixel in ds9, but you can see a graph of it by using a region

3.1.2 HR cube

- Exactly the same, except that NV = 5614
  - Wavelength scale: 0.85 angstroms
  - CRPIX3 = 1
  - CRVAL3 = 4595.
- First try at dividing it up: do it by wavelength
  - Divide into 8 parts of length 702
    - Last one will be 700
  - Size will be 0.702 1.476 1.766 4 = 7.32 GB

Section	CRVAL3
0	4595.00
1	5191.70
2	5788.40
3	6385.10
4	6981.80
5	7578.50
6	8175.20
7	8771.90

```
from astropy.io import fits
from astropy import wcs
import numpy as np

hdulist = fits.open('DATA/DATACUBEFINALuser_20140216T010259_78380e1d.fits')
datcube = hdulist['DATA']
```

```
errcube = hdulist['STAT']
sections = np.arange(8, dtype=int)
NV = 702
k1_list = sections*NV
k2_list = k1_list + NV
wav0_list = datcube.header['CRVAL3'] + datcube.header['CD3_3']*NV*sections
for section, k1, k2, wav0 in zip(sections, k1_list, k2_list, wav0_list):
    fn = 'muse-hr-data-wavsec{}.fits'.format(section)
    hdr = datcube.header.copy()
    hdr['CRVAL3'] = wav0
    hdr['NAXIS3'] = NV
    print('Writing', fn)
    fits.PrimaryHDU(header=hdr, data=datcube.data[k1:k2]).writeto(fn)
```

3.1.3

3.1.4 Heliocentric correction

Again, these snippets need to be run on the CRyA server where the big data cubes are

1. Looking for keywords in the top-level header

```
hdr = hdulist[0].header
hdr.tofile('HRcube.hdr', sep='\n', padding=False)
```

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                    8 / number of bits per data pixel
NAXIS   =                    0 / number of data axes
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
DATE    = '2014-11-13T08:54:24' / file creation date (YYYY-MM-DDThh:mm:ss UT)
ORIGIN  = 'TEST    '           / European Southern Observatory
TELESCOP= 'ESO-VLT-U4'         / ESO <TEL>
INSTRUME= 'MUSE    '           / Instrument used.
RA      =            83.780509 / [deg] 05:35:07.3 RA (J2000) pointing
DEC     =             -5.39556 / [deg] -05:23:44.0 DEC (J2000) pointing
EQUINOX =                2000. / Standard FK5
RADECSYS= 'FK5     '           / Coordinate system
EXPTIME =                   5. / Integration time
```

```
MJD-OBS =       56704.04374097 / Obs start
DATE-OBS= '2014-02-16T01:02:59.219' / Observing date
UTC     =               3770. / [s] 01:02:49.000 UTC
LST     =            21901.85 / [s] 06:05:01.850 LST
PI-COI  = 'UNKNOWN '           / PI-COI name.
OBSERVER= 'UNKNOWN '           / Name of observer.
PIPEFILE= 'DATACUBE_FINAL.fits' / Filename of data product
BUNIT   = '10**(-20)*erg/s/cm**2/Angstrom'
DATAMD5 = '69173383d3718d3ddb46e187f4cc2954' / MD5 checksum
OBJECT  = 'M42-lr  '           / Original target.
CHECKSUM= 'NcfSOcZPNcdPNcZP'   / HDU checksum updated 2014-11-12T22:17:16
DATASUM = '0       '           / data unit checksum updated 2014-11-12T22:17:16
HIERARCH ESO OBS AIRM =     5. / Req. max. airmass
HIERARCH ESO OBS AMBI FWHM = 2. / Req. max. seeing
... ETC ...
```

So, this does have the info that we need: RA, DEC, MJD-OBS in particular