Assignment 7 - Map Velocity Gradients

Code by: **Lorenzo Salgado**
Project Members: **Ethan James Sykes, Valentina La Torre, Alyssa Eakman**
AST-016 - **Prof. Danilo Marchesini**

Due: **05/11/2021**

This is a continuation of the work done in Assignment 6 where we produced figures of an observed galaxy's spectra, supported by the MaNGA data and tutorials.

In this assignment, we will use the previously observed galaxy and redshifted spectra (H_{α}, and two neighboring **NII**) to evaluate the performance of paramaters in a given model for flux. These measured parameters will span the 74X74 spectral range of the provided galaxy image and will be used to produce speed map gradient figure of the galaxy. Using this speed gradient, we can measure the line-of-sight velocity of the galaxy, necessary when considering the consistent expansion of the universe due to entropy.

We are using the same MaNGA data used in assignment 6 to map flux values for different spectra across the face of a galaxy, however now the observed spectra range will be reduced to +-200 Angstrom around the redshifted H_{α} and subsequent observed values also reduced to fit the same range. This is done to narrow down our spectra of interest around H_{α}, $[NII]_{a}$, $[NII]_{b}$ values.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from astropy.io import fits
from scipy.optimize import curve_fit
```

Part 1: Model Fitting

This is the first part to this Lab assignment where we will load in previously used galaxy data, and will apply a fitting measure for a given model with 6 parameters. Across the $74x74$ spectral grid, the curve_fit procedure will output the best coefficients values for the 6 model parameters - producing 12 matrices from 6 parameters and their error. There parameters are peak values of H_alpha, Niia, Niib, sigma, z, and C for the given model:

$$f = C \ | \ A_{H_{\alpha}} e^{-[\lambda_{obs} - \lambda H_{\alpha,rest}(1+z)]^2/2\sigma^2} \ | \ A_{[NII]_a} e^{-[\lambda_{obs} - \lambda_{[NII]_{a,rest}}(1+z)]^2/2\sigma^2} \ | \ A_{[NII]_b} e^{-[\lambda_{obs} - \lambda_{[NII]_{b,rest}}(1+z)]^2/2\sigma^2}$$

Where $A_{H_{\alpha}}, A_{[NII]_a}, A_{[NII]_b}$ are the values at the peaks of each respective emission line; C is the value of the continuum; $\lambda H_{\alpha,rest}, \lambda_{[NII]_{a,rest}}, \lambda_{[NII]_{b,rest}}$ are the rest frame wavelengths of the respective emission lines; z is the redshift at each spacial position; σ is the dispersion of the Gaussian used to model the emission line; and λ_{obs} is the observed wavelength.

Defining the Model:

In [39]:
```
def f_model(wave_obs, Halpha, Niia, Niib, sigma, z, C):

    # inputs
    #########
          # wave_obs - masked wavelength vector at pos (x,i)
          # Halpha - value at peak of H_alpha emission line
          # Niia - value at peak of Niia emission line
          # Niib - value at peak of Niib emission line
          # sigma- Dispersion of Gaussian
          # z - redshift value of the galaxy
          # C - Continuum value
    # output
    #########
          # y data, is a model of flux

    # Values from assignment 6 table, at rest.
    ha_r, Niia_r, Niib_r = 6564.61, 6549.86, 6585.27

    # flux model formula is fit with 3 gaussian functions
    flux = C+ Halpha * np.exp(  -(wave_obs-ha_r*(1+z) )**2 / (2*sigma**2))
    + Niia * np.exp(    -(  wave_obs-Niia_r*(1+z)  )**2 / (2*sigma**2))
    + Niib * np.exp(    -(  wave_obs-Niib_r*(1+z)  )**2 / (2*sigma**2))
    return flux
```

Loading in Data:

From: https://www.sdss.org/dr13/manga/manga-tutorials/

In [40]:
```
cube = fits.open('manga_8244_12704\manga-8244-12704-LINCUBE.fits')


# Re-order FLUX, IVAR, and MASK arrays from (wavelength, DEC, RA) to (RA, DEC, wavelength).
flux = np.transpose(cube['FLUX'].data, axes=(2, 1, 0))
ivar = np.transpose(cube['IVAR'].data, axes=(2, 1, 0))
mask = np.transpose(cube['MASK'].data, axes=(2, 1, 0))
wave = cube['WAVE'].data
flux_header = cube['FLUX'].header
```

Before iterating through each position in our $74x74$ spacial grid, we want to reduce the flux, ivar, mask, and wave vectors from 6732 Å to 200 Å; $H_{\alpha,redshift} \pm 100$.

This value is:

> $II_{\alpha,redshift} = \lambda_{H_{\alpha},rest} * (1 + redshift)$

Where $\lambda_{H_\alpha,rest} = 6564\ \text{Å}$ and where *redshift* $= 0.115137$.

This resultant wavelength vector is obtained by the following process:

- Find index in wavelength vector at which redshifted H_α wavelength occurs.
- Use slicing operators [:,:,index] to reduce range of wavelengths for flux, ivar, mask, and wave.
- Save the updated matrices in new variables.

So we want to modify shape from (74,74,6732) to (74, 74, 200). Our wavelength vector's used indices would be [7219, 7420].We applied the slicing operator in python to select the relevant section of the provided matrices for flux, ivar, mask, and wave.

In [41]:
```
h_alpha_redshift = 6563*(1+0.115137)

# will take length of index of all wavelength values less than
# H_a_rest to determine where index occurs.
ind_l, = np.where(wave <=(h_alpha_redshift)-100)
ind_u, = np.where(wave <=(h_alpha_redshift)+100)
# get index of  +/-100A from H_alpha_redshifted
lower = (len(ind_l))
upper = (len(ind_u))
print(upper-lower) #200 angstrom

wave_new = wave[lower:upper]
flux_new = flux[:, :, lower:upper]
ivar_new = ivar[:, :, lower:upper]
error_new = np.sqrt(1/ivar_new)
mask_new = mask[:, :, lower:upper]
print(np.shape(wave_new))
print(np.shape(flux_new))
```

```
200
(200,)
(74, 74, 200)
```

```
<ipython-input-41-e3bcf24b73a6>:15: RuntimeWarning: divide by zero encountered in true_divide
  error_new = np.sqrt(1/ivar_new)
```

The above reduces the relevant range of spectra that will be used to produce a speed map gradient to a shape of $(74x74x200)$ rather than $(74x74x6375)$

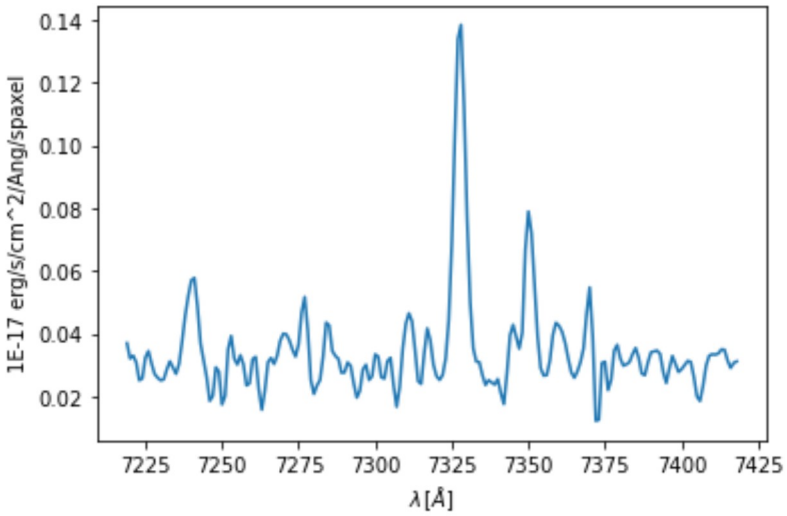
Apply updated mask matrix to new matrices 'centered' at H_α $\pm$100

This will be done first at spectral position [32, 14] (x=32, y=14) before expanding for the whole spectral range.

In [42]:
```
do_not_use = [(mask_new[32,14] & 2**10) == 0]

# using splice operators for pos x=32, y=14 on reduced range matrices.
val_m_cr = flux_new[32, 14, :]
err_m_cr = error_new[32,14,:]

## THEN apply mask
val_cr = val_m_cr[tuple(do_not_use)]
print(np.median(val_cr))
err_cr = err_m_cr[tuple(do_not_use)]

plt.plot(wave_new,val_cr)
plt.xlabel('$\lambda \, [\AA]$')
plt.ylabel(flux_header['BUNIT'])
```

```
0.03134951
```

Out[42]:
```
Text(0, 0.5, '1E-17 erg/s/cm^2/Ang/spaxel')
```



Apply model fit procedure for best-fit parameters at position $x = 32, y = 14$ with initial parameter estimates given to the model

In [58]:
```
# defining initial values for curve_fit parameter fitting
C = np.median(val_cr)
z = 0.115137 # redshift of galaxy
sigma = 3 # Angstrom
halpha = np.amax(val_cr)-np.median(val_cr)# peak

ha_max = halpha*1.1
Niia = halpha / 3.5 # estimated peak
Niia_max = halpha*0.4
Niib = halpha / 2 # estimated peak
Niib_max = halpha*.7
p0_n = [halpha, Niia, Niib, sigma, z, C]
max_bounds=((0,0,0,0,0,-np.inf),(ha_max, Niia_max, Niib_max, 5, 0.2, np.inf))

# apply model for fluxes
popt1,pcov = curve_fit(f_model, wave_new, val_cr, p0=p0_n, bounds=max_bounds)
print('\n flux fit:')
print('Halpha: ', popt1[0] ,'\nNiia:   ',popt1[1], '\nNiib:   ',popt1[2],
      '\nsigma: ',popt1[3],'\nz:      ',popt1[4], '\nC:      ', popt1[5])

# apply model for errors
print('\n error fit:')
# popt2,_ = curve_fit(f_model, wave_new, err_cr, p0=p0_n)
popt2 = np.diag(np.sqrt(pcov))
print('\nHalpha: ', (popt2[0]) ,'\nNiia:   ',popt2[1], '\nNiib:   ',popt2[2],
      '\nsigma:  ',popt2[3],'\nz:      ',popt2[4], '\nC:      ', popt2[5])
```

```
 flux fit:
Halpha:  0.10899778005497109
Niia:    6.352816822271547e-07
Niib:    0.07105863388722754
sigma:  1.7566353693373902
z:       0.11623273696366757
C:       0.03220431113116283

 error fit:

Halpha:  0.006460838429525823
Niia:    3.105751819884129e-17
Niib:    2.0373227876954407e-16
sigma:   0.1208817826893661
z:       1.8265621582638444e-05
C:       0.000672300855470229
<ipython-input-58-46a886492cef>:24: RuntimeWarning: invalid value encountered in sqrt
  popt2 = np.diag(np.sqrt(pcov))
```

Initialize Paramater / Error Matrices and save calculated values from above in position [32,14].

In [63]:
```
## Saving Parameter Values (Halpha, Niia, Niib, sigma, z, C) + (Halpha_err, Niia_err, Niib_err, sigma, z, C)
## Each matrix is identical in shape - 74 x 74

ha, ha_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)
na, na_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)
nb, nb_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)
sgm, sgm_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)
zm, zm_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)
Cm, Cm_err = np.full(shape=(74,74),fill_value=np.PZERO), np.full(shape=(74,74),fill_value=np.PZERO)

ha[32,14] = popt1[0]
ha_err[32,14] = popt2[0]

na[32,14] = popt1[1]
na_err[32,14] = popt2[1]

nb[32,14] = popt1[2]
nb_err[32,14] = popt2[2]

sgm[32,14] = popt1[3]
sgm_err[32,14] = popt2[3]

zm[32,14] = popt1[4]
zm_err[32,14] = popt2[4]

Cm[32,14] = popt1[5]
Cm_err[32,14] = popt2[5]

print(np.shape(Cm_err))
```

```
(74, 74)
```

In [64]:
```
np.savetxt(fname='z_map.HDF5', X=zm)
```

We save the matrices in HDF5 files that can be loaded in for use in the second part of the lab. This avoids the time-intensive curve-fitting process as we expand the above process to the whole spectral range $x = [0, 74]; y = [0, 74]$

Now applying for all spectral positions, x = [0,74]; y=[0,74]

We want to iterate through each spacial position (74x74) in our modified shrunken flux matrix of wavelength values [H_alpha wavelength +- 100]

Iterate through masked arrays ONLY if there are more than 10 non_zero points in vector.

This step is an expansion of the procedure carried out above. 74x74 = 5476 iterations maximum. Values calculated in this cell are saved locally in HDF5 files for simple accessing for step 2.

In [65]:

```
for i in range(0,len(mask)):
    for j in range (0, len(mask)):
        # access flux, error, mask values at i,j == x,y coordinates
        val_m = flux_new[i, j, :]
        err_m = error_new[i, j, :]
        m_f = mask_new[i,j,:]
        mask_val = [(m_f & 2**10) == 0]

        val_f = val_m[tuple(mask_val)]
        err_f = err_m[tuple(mask_val)]
        wav_f = wave_new[tuple(mask_val)]

        if (len(val_f) > 10): # determines valid minimum good values
            # carrying on with the curve fit

            # intial guess conditions, different for every [i,j]
            C = np.median(val_f)
            z = 0.115137                                # redshift of galaxy
            sigma = 3                                   # Angstrom
            halpha = np.amax(val_f)-np.nanmedian(val_f)# peak
            halpha_max = halpha*1.1
            Niia = halpha*(1/3.5)                          # estimated peak
            Niia_max = halpha*0.4
            Niib = halpha*.5                            # estimated peak
            Niib_max = halpha*.7
            p0_n = [halpha, Niia, Niib, sigma, z, C]    # intitial parameter array


            ydata = val_f+err_f

            # apply curve_fit
            if (abs(np.median(val_f))>0):
                popt_f,pcov = curve_fit(f_model, wav_f, ydata, p0=p0_n,
                                        bounds=((0,0,0,-5,0,-np.inf),
                                                (halpha_max, Niia_max,
                                                 Niib_max, 5, 0.2, np.inf)),
                                        maxfev=100000)

                pcov_e = np.diag(np.sqrt(pcov))
                # fill previously defined 74x74 NaN matrix with respective parameter values
                ha[i,j] = popt_f[0]
                ha_err[i,j] = pcov_e[0]

                na[i,j] = popt_f[1]
                na_err[i,j] = pcov_e[1]

                nb[i,j] = popt_f[2]
                nb_err[i,j] = pcov_e[2]

                sgm[i,j] = popt_f[3]
```

```
                zm[i,j] = popt_f[4]
                zm_err[i,j] = pcov_e[4]

                Cm[i,j] = popt_f[5]
                Cm_err[i,j] = pcov_e[5]

        else: # does not apply curve fit
            ('')
# saves completed matrices in HDF5 files
np.savetxt(fname='ha_map.HDF5', X= ha)
np.savetxt(fname='ha_err_map.HDF5', X=ha_err)
np.savetxt(fname='niia_map.HDF5', X=na)
np.savetxt(fname='niia_err_map.HDF5', X=na_err)
np.savetxt(fname='niib_map.HDF5', X=nb)
np.savetxt(fname='niib_err_map.HDF5', X=nb_err)
np.savetxt(fname='sigmap.HDF5', X=sgm)
np.savetxt(fname='sigmap_err.HDF5', X=sgm_err)
np.savetxt(fname='z_map.HDF5', X=zm)
np.savetxt(fname='z_err_map.HDF5', X=zm_err)
np.savetxt(fname='C_map.HDF5', X=Cm)
np.savetxt(fname='C_err_map.HDF5', X=Cm_err)
```

```
<ipython-input-65-c9ab1692c399>:37: RuntimeWarning: invalid value encountered in sqrt
  pcov_e = np.diag(np.sqrt(pcov))
```

The above expands our initial single position curve-fitting and model evaluation procedure. This procedure has produces 12 HDF5 files, 2 for each model parameter and associated error value across the 74x74 spectral range of our galaxy. These files, once in the directory, ensure that the curve-fitting procedure does not require to be run every time. Per the most recent run on May 9th, running this cell requires ~47 minutes of compilation time.

Below, these files will be loaded in and used to produce the velocity map of the galaxy.

Part 2: Producing Speed Maps

Below, we will use the z_map created above to define a velocity map across the spectral range observed and measured.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# load the parameters
C = np.loadtxt('C_map.HDF5')
C_err = np.loadtxt('C_err_map.HDF5')
A_Ha = np.loadtxt('ha_map.HDF5')
Ha_err = np.loadtxt('ha_err_map.HDF5')
A_Niia = np.loadtxt('niia_map.HDF5')
Niia_err = np.loadtxt('niia_err_map.HDF5')
A_Niib = np.loadtxt('niib_map.HDF5')
Niib_err = np.loadtxt('niib_err_map.HDF5')
sigma = np.loadtxt('sigmap.HDF5')
sigma_err = np.loadtxt('sigmap_err.HDF5')


# we use the z-map generated to produce velocity maps
zm = np.loadtxt('z_map.HDF5')
zmap = zm
```

Initialize the 74x74 matrices for velocity map values.

In [9]:
```
# NAN are invalid values overwritten by accepted values
v_map = np.full(shape=(74,74),fill_value=np.nan)
v_im = np.full(shape=(74,74),fill_value=np.nan)

c = 2.9792458e5 # speed of light
v_map = (zmap-0.115137) * c
```

Iterate through each 74x74 position and evaluate on basis of given conditional statements.

A is the ratio of the measured H_α peak over the H_α error.

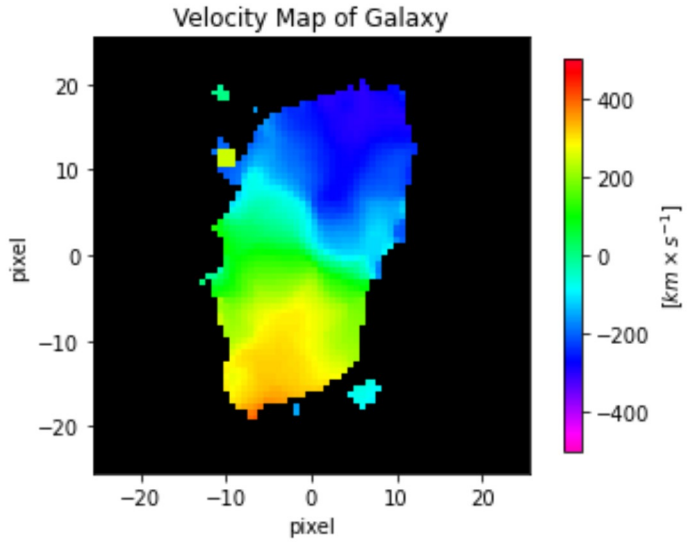
In [4]:
```
# starting points in loop are reduced to surrounding our range of interest
for i in range(18,len(zmap)-19):
    for j in range(0,len(zmap)):
        A = (A_Ha[i,j] // Ha_err[i,j])
        Nb = (A_Niib[i,j]//Niib_err[i,j])
        C_e = (5*C_err[i,j])
        if (A>3) & (Nb > 3) &(A_Ha[i,j]>C_e):
            if (v_map[i,j] > -500) & (v_map[i,j] < 500):
                v_im[i,j] = v_map[i,j]
```

```
<ipython-input-4-4b48b8580ae7>:4: RuntimeWarning: invalid value encountered in double_scalars
  A = (A_Ha[i,j] // Ha_err[i,j])
<ipython-input-4-4b48b8580ae7>:5: RuntimeWarning: invalid value encountered in double_scalars
  Nb = (A_Niib[i,j]//Niib_err[i,j])
```

In [5]:
```
im = v_im.T

extent = [-25.5000148, 25.5000148, -25.5000148, 25.5000148]
cmap = (plt.cm.get_cmap('gist_rainbow_r'))

ax = plt.imshow(im, extent=extent, cmap=cmap, vmin=-500, vmax=500, origin='lower', interpolation='none')
cmap.set_bad('k',1.)
plt.colorbar(ax, shrink=0.9, label='[$km \\times s^{-1}$]')
plt.title('Velocity Map of Galaxy')
plt.xlabel('pixel')
plt.ylabel('pixel')
```

```
<ipython-input-5-64abf96489ff>:7: MatplotlibDeprecationWarning: You are modifying the state of a globally registered colo
rmap. In future versions, you will not be able to modify a registered colormap in-place. To remove this warning, you can
make a copy of the colormap first. cmap = copy.copy(mpl.cm.get_cmap("gist_rainbow_r"))
  cmap.set_bad('k',1.)
```

Out[5]:
```
Text(0, 0.5, 'pixel')
```

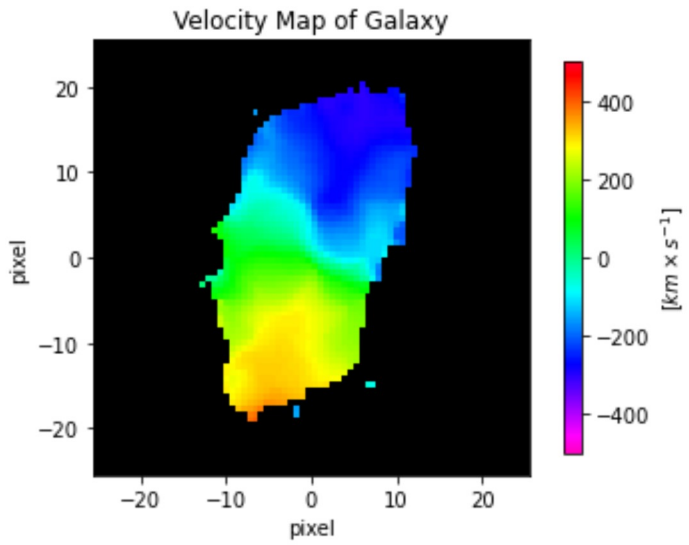


The above velocity map is *nearly* up to what we aimed to produce. We want to remove the 'noise' found at the edges of the galaxy map. To do this, we implement python's splice tools (i.e., [:,]) to replace undesired points with NAN values.

In [6]:
```
# filtering 'bad points'
v_im[50:len(v_im),0:len(v_im)-35] = np.nan # bottom right
v_im[0:25,48:len(v_im)] = np.nan # top left
v_im[36:len(v_im),0:len(v_im)-59] = np.nan # bottom right
```

In [7]:
```
im = v_im.T
ax = plt.imshow(im, extent=extent, cmap=cmap, vmin=-500, vmax=500, origin='lower', interpolation='none')
plt.colorbar(ax, shrink=0.9, label='[$km \\times s^{-1}$]')
plt.title('Velocity Map of Galaxy')
plt.xlabel('pixel')
plt.ylabel('pixel')
```

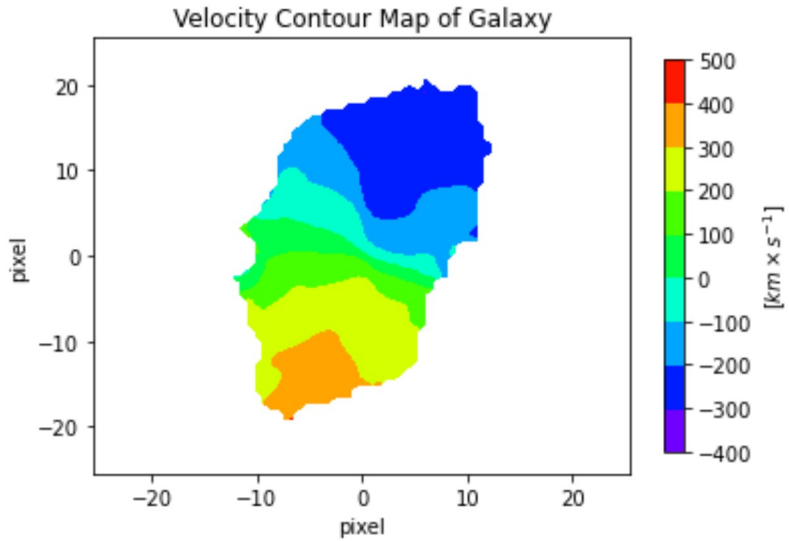
Out[7]:
```
Text(0, 0.5, 'pixel')
```



The measured speed map above indicates the velocity is greatest along the line of sight in the bottom red region and the gradient decreases spanning upwards along the face of the galaxy. We can explain the observed difference in velocity through the fact that, while photons travel at light-speed, the galaxy's non-uniform expansion is felt greatest at the lower speed ranges and least toward the upper velocity ranges. Hence, it seems that the 'top' of the galaxy is moving away from us at a greater intensity than the 'bottom' of the galaxy.

In [12]:
```
fig, ax = plt.subplots()
cs = ax.contourf(im, cmap=cmap, vmin=-500, vmax=500, extent=extent)
cbar = fig.colorbar(cs, label='[$km \\times s^{-1}$]', shrink=0.9)
plt.title('Velocity Contour Map of Galaxy')
plt.xlabel('pixel')
plt.ylabel('pixel')



#
```

Out[12]:
```
Text(0, 0.5, 'pixel')
```



The above is a contour map, identifying the specific temperature transition points across the face of the galaxy. As can be observed, the region that approaches $0\ km \times s^{-1}$ is a thin band across the center of the galaxy.