```
from astropy.io import fits
import numpy as np
from scipy.ndimage.filters import gaussian_filter
from scipy.optimize import curve_fit
import numpy.ma as ma
import matplotlib.pyplot as plt
import matplotlib

from emission_line_fitting import gauss

def flux(CUBE, sky_ra_min, sky_ra_max, sky_dec_min, sky_dec_max):
    # define parameters from header
    cube = fits.open(CUBE)

    # number of pixels across
    n1 = cube[0].header['naxis1']
    # starting/lowest wavelength observed
    wl = cube[0].header['crval1']
    # change in wavelength per pixel
    dw = cube[0].header['cdelt1']
    # number of pixels vertical
    n2 = cube[0].header['naxis2']
    # number of slices
    n3 = cube[0].header['naxis3']

    # scale -- number of arcseconds per pixel
    scale = float(cube[0].header['sscale'])

    # create list of all measured wavelengths
    wave_list = []
    for j in range(0, n1):
        val = dw * j + wl
        wave_list.append(val)

    # access and get information about primaryHDU -- (slice #, y, x)
    # for axis order, 1,2,3 --> RA, DEC, wavelength
    cube1 = cube[0].data

    # estimate error from a clean sky region
    sky = cube1[sky_ra_min:sky_ra_max, sky_dec_min:sky_dec_max,:]
    err = []

    for j in range(0,n1):
        error = np.std(sky[:,:,j])
        err.append(error)

    # define pixel flux list + calculate total flux for each point in space (for each
RA and dec)
    flux = []
    for j in range(0,n3):
        for k in range(0,n2):
            datapt = cube1[j,k,:]
            flux.append(datapt)

    # separate data into lists for each RA slice
    data = [flux[n2*j:n2*(j+1)] for j in range(0, n3)]

    # data has n3 elements -- each RA slice
    # data[i] has n2 elements -- each DEC slice for 'i'th RA
    # data[i][j] has n1 elements -- flux value for each wavelength at 'i'th RA and 'j'
th DEC

    #flux_data = np.sum(data[:,:,284:295], axis=2)[signal_ra_min:signal_ra_max, signal
_dec_min:signal_dec_max]

    return wave_list, data, err, scale

def uncorrected_kin_maps(CUBE, sky_ra_min, sky_ra_max, sky_dec_min, sky_dec_max, signa
l_ra_min, signal_ra_max, signal_dec_min, signal_dec_max, guess, bounds, filt, channel_
low, channel_high, edges = False):

    params = []
    uncerts = []
    velocity_dispersion = []
    wave = []
    SN = []
    #flux_data = []

    c = 2.998 * 10**5

    wave_list, data, err, scale = flux(CUBE, sky_ra_min, sky_ra_max, sky_dec_min, sky_
dec_max)

    # smooth the data with a gaussian filter
    filtereddata = gaussian_filter(data, sigma = [filt, filt, 0])

    tot_flux = []
    for i in filtereddata[signal_ra_min:signal_ra_max]:
        flux_j = []
        for j in i[signal_dec_min:signal_dec_max]:
            val = np.sum(j[channel_low:channel_high])
            flux_j.append(val)
        tot_flux.append(flux_j)

    # set SN cutoff bound
    SNbound = 5

    # choose axes bounds for plots
    ramin = signal_ra_min
    ramax = signal_ra_max
    deltara = ramax - ramin
    decmin = signal_dec_min
    decmax = signal_dec_max
    deltadec = decmax - decmin

    for j in range(ramin, ramax):
        for k in range(decmin, decmax):

            # curve_fit
            popt, pcov = curve_fit(gauss, wave_list, filtereddata[j][k], p0=guess, sig
ma=err, bounds=(0, bounds))

            #uncertainties in fit values
            unc = np.sqrt(np.diag(pcov))

            # define parameters from popt
            wavelength = popt[0]
            amp = popt[1]
            sig = popt[2]

            vdisp = c * (sig/wavelength)

            signal_to_noise = amp / unc[1]

            # store parameters + uncertainties
            params.append(popt)
            uncerts.append(unc)
            velocity_dispersion.append(vdisp)
```

```
            wave.append(wavelength)
            SN.append(signal_to_noise)
            #flux_data.append(amp)

    SN = np.asarray(SN)
    velocity_dispersion = np.asarray(velocity_dispersion)
    wave = np.asarray(wave)

    SN_mask = ma.masked_where(SN < 5, SN)

    wave_med = ma.median(ma.masked_array(wave, mask=SN_mask.mask))
    redshift = (wave_med/656.3) - 1
    z = (wave - wave_med)/wave_med
    vel = c * (((z + 1)**2 - 1)/((z+1)**2 + 1))

    vel_mask = ma.masked_outside(vel, -150, 150)

    disp_mask = ma.masked_where(velocity_dispersion > 150, velocity_dispersion)

    dispersion_SN_mask = ma.mask_or(SN_mask.mask, disp_mask.mask)
    velocity_SN_mask = ma.mask_or(SN_mask.mask, vel_mask.mask)

    combined_mask = ma.mask_or(dispersion_SN_mask, velocity_SN_mask)

    dispersion = np.asarray([velocity_dispersion[deltadec*j:deltadec*(j+1)] for j in r
ange(0, deltara)])
    dispersion = ma.masked_array(dispersion, mask=combined_mask)

    velocity = np.asarray([vel[deltadec*j:deltadec*(j+1)] for j in range(0, deltara)])
    velocity = ma.masked_array(velocity, mask=combined_mask)

    #flux_data = np.asarray([flux_data[deltadec*j:deltadec*(j+1)] for j in range(0, de
ltara)])

    if edges == True:
        dispersion.mask[0]=True
        dispersion.mask[-1]=True
        np.transpose(dispersion.mask)[0]=True
        np.transpose(dispersion.mask)[-1] = True

        velocity.mask[0]=True
        velocity.mask[-1]=True
        np.transpose(velocity.mask)[0]=True
        np.transpose(velocity.mask)[-1] = True

    #params = np.asarray([params[deltadec*j:deltadec*(j+1)] for j in range(0, deltara)
])


    return dispersion, velocity, scale, params, wave_list, combined_mask, redshift, to
t_flux

def plot_maps(dispersion, velocity, deltara, deltadec, scale, tot_flux):

    velocityscale = max(np.max(velocity), abs(np.min(velocity)))
    cmap_disp = matplotlib.cm.viridis
    cmap_disp.set_bad(color='white')

    cmap_vel = matplotlib.cm.rainbow
    cmap_vel.set_bad(color='white')

    cmap_data = matplotlib.cm.cool
    cmap_vel.set_bad(color='white')

    plt.figure(figsize=(10,6))
    plt.subplot(1,3,1)
    plt.imshow(dispersion, cmap = cmap_disp, origin = 'lower'
               ,extent = (0, deltadec * scale, 0, deltara * scale)
              )
    plt.colorbar()
    plt.title('velocity dispersion')
    plt.xlabel('arcsec') #dec
    plt.ylabel('arcsec') #RA

    plt.subplot(1,3,2)
    plt.imshow(velocity, cmap = cmap_vel, origin = 'lower', vmin = -velocityscale, vma
x = velocityscale
               ,extent = (0, deltadec * scale, 0, deltara * scale)
              )
    plt.colorbar()
    plt.title('velocity')
    plt.xlabel('arcsec') #dec
    plt.ylabel('arcsec') #RA

    plt.subplot(1,3,3)
    plt.imshow(tot_flux, cmap = cmap_data, origin = 'lower'
               ,extent = (0, deltadec * scale, 0, deltara * scale)
              )
    plt.colorbar()
    plt.title('flux')
    plt.xlabel('arcsec') #dec
    plt.ylabel('arcsec') #RA

    plt.tight_layout()
    plt.show()
```