Millimeter Cartography: A Step by Step Guide to Creating Simulated Millimeter Images

Abstract

Simulated maps of the sky can be helpful in observational astronomy in order to predict what one would observe with a given telescope. In our case, we develop simulated maps for the TolTEC camera on the Large Millimeter Telescope (LMT). Source positions are identified using Hubble Space Telescope imaging for five extragalactic fields, adopting the photometric catalogs from the 3D-HST Survey. In order to make the simulated maps, we convolve the 1.1mm point spread function (PSF) for the TolTEC camera with the predicted mm flux densities when extrapolating the 3D-HST spectral energy distributions at the location of each source. We then add Gaussian noise to the image, mimicking the anticipated depth of the Ultra Deep Galaxy Survey.

This tutorial is a step by step guide to creating simulated TolTEC images. With these simulated maps, we can predict how the sky should look at mm wavelengths when observing these well-known extragalactic fields. Source confusion is the main bottleneck of longer wavelength observations, where different sources are so close together that their the extended light profiles blend together such that they cannot be distinguished individually. Simulated maps such as those created through this tutorial are therefore invaluable for testing the depth to which we can reliably extract flux densities with real observations.

3D-HST Survey website: https://archive.stsci.edu/prepds/3d-hst/

Ultra Deep Galaxy Survey: http://toltec.astro.umass.edu/science_ultra_deep_survey.php

Introduction

Dust-obscured galaxies, known as submillimeter galaxies, are best detected at very long wavelengths. These type of galaxies are known for having high star formation rates, where the vast majority of light is obscured by dust and re-radiated in the far-infrared (which includes mm wavelengths). However, longer wavelength observations suffer from poorer resolution given that the theoretical diffraction limit of a telescope scales with telescope size and the wavelength of

observation. Most observations to date at these wavelengths are relatively shallow, typically having low signal to noise ratio (SNR). The combination of poor resolution and relatively shallow depths exacerbates issues related to source confusion (see description above in abstract).

The Large Millimeter Telescope's TolTEC camera is a fairly new instrument, replacing the previous camera AzTEC, that will allow us to image huge areas of the sky and trace out the formation of structures ranging from the sizes of comets to the largest objects in the Universe. This camera will allow us to explore the universe at the millimeter and submillimeter wavelengths, which is useful for observing through the efeects of interstellar dust and to study galaxy structure and evolution.

We will use the LMT's TolTEC camera point spread function (PSF) and the Ultra-Deep Galaxy Survey to simulate how large portions of the sky would look like if we were to use TolTEC for future observations. These simulations are valuable as we are able to refine our simulated maps of a chosen extragalactic field until we get a desirable result and SNR. This would ultimately help us obtain more accurate data in the future as we are able to test the capabilities of our chosen instruments (in this case the LMT and TolTEC) before we begin taking data from real observations.

Data and Methods

In this notebook, we will work with FITS (Flexible Image Transport System) files that are created by code that we will write in Python. FITS files are used in astronomy to read data from images, as each pixel contains a number of counts that can be converted to a flux density given the absolute calibration stored in the 'header' of the FITS file. This conversion factor to go from counts to an actual flux density is often called the image 'zeropoint'." FITS files in astronomy typically show a 2D image in x and y coordinates (though they are also a useful way to store data tables in a structure format), and it contains information within the header that is useful, including the zeropoint as described above, but also values such as the exposure time, filter used, location of the point on the sky in the world coordinate system (WCS), etc.

After writing our functions in the Python language, we will use the source positions as defined from the 3D-HST flux catalogs. Each flux density estimate is then convolved with the Point Spread Function (PSF) at the location of each source. For this tutorial, we are using extrapolated 1.1mm flux densities together with the 1.1mm TolTEC PSF on the Large Millimeter Telescope.

Data from the 3D-HST photometric catalog is extrapolated to millimeter wavelengths using the best-fit Prospector model for each source. The survey includes five different extragalactic fields (AEGIS, COSMOS, GOODS-NORTH, GOODS-SOUTH, and UDS). While TolTEC observes simultaneously at three wavelengths (1.1mm, 1.4mm, and 2.0mm), we focus herein only on 1.1mm

observations. The coordinates of each sources in right ascension and declination are adopted in order to build our map and set the locations and source density of objects on the sky.

In this tutorial, we will write four different functions that will help us to create a simulated map of a desired field. The first one is the "makeField" function, which will create an empty square field into which we will fit our data. This function will be called in within the "makeSignalMap" and "makeNoiseMap" functions to create the fields for both the Signal and Noise Maps. The "makeSignalMap" function will take a desired survey field catalog and will ultimately convolve it using the TolTEC's 1.1mm beam PSF data. This function will also return a FITS file of the Signal Map. The "makeNoiseMap" function will construct a random simulated noise map in a similar way as the previous function, by taking a random Gaussian noise map and convolving it with our PSF data to return a FITS file of the now constructed noise map. The final function is the "makeSignalNoiseMap" function, which will take the FITS files for both the desired survey field map and the noise map, and add them together to generate the final simulated map. This function will return the final map as a FITS file. We will show the returned FITS files of the Signal Maps, the Noise Map, and the Final Maps later in the notebook.

With the simulated maps, we can measure the true values of the millimeter flux as seen using the TolTEC camera of the different stars and galaxies in the given catalog, as well as measure their flux as seen using the LMT once the simulated noise is added to it. It is valuable to compare the true flux injected into the images with that which we recover in order to verify the depth down to which photometry can be measured robustly.

Getting Started

> **Before we begin**

Make sure to save a copy of this notebook by going to "File" and selecting "Save a copy in Drive." DO NOT EDIT THE ORIGINAL NOTEBOOK, START RUNNING THE TUTORIAL IN YOUR SAVED COPY.

> **Importing the packages**

For this code, we will be using the packages in the code cell below. Make sure to run the code cell to import all the packages.

```
import numpy as np
from astropy import wcs
import matplotlib.pyplot as plt
from astropy.io import ascii
```

```
from scipy.signal import fftconvolve
from scipy.ndimage import shift
from astropy.io import fits
from scipy.stats import norm
```

Getting our Data

The gdown package will allow us to download to Google Colab the necessary files without having to upload them every single time that the notebook is opened, as long as the file is stored in Google Drive. The necessary data are already in Google Drive so you just need to run the code cells below to download them to the notebook. You'll still need to run the following cell whenever you open the notebook to download the data.

```
pip install gdown
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.11/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from gdo
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.11/dist-
```

Run the following code cell to download the 3D-HST catalog. You'll notice it will appear in your files if you click the "Files" folder on the left. It will appear as "3dhst_mmflux.cat" if you look at your files.

```
!gdown https://drive.google.com/uc?id=1XpEnOxSC0FJT-tEL17am9XNFJc8aEgdR
```

```
Downloading...
From: https://drive.google.com/uc?id=1XpEnOxSC0FJT-tEL17am9XNFJc8aEgdR
To: /content/3dhst_mmflux.cat
100% 6.42M/6.42M [00:00<00:00, 141MB/s]
```

Run the following code cell to download the TolTEC Point Spread Function (PSF). For this notebook, we will use the TolTEC 1.1 mm PSF. It will appear as "toltec_1100_PSF.fits" on your files to the left.

```
!gdown https://drive.google.com/uc?id=1Hyi3XXkS6axpG5KCD-yNr2qUJto4va6r
```

```
Downloading...
From: https://drive.google.com/uc?id=1Hyi3XXkS6axpG5KCD-yNr2qUJto4va6r
To: /content/toltec_1100_PSF.fits
100% 985k/985k [00:00<00:00, 68.6MB/s]
```

Since we are using TolTEC's 1.1mm band, we define our Full Width at Half-Maximum (FWHM) and our sigma to correspond with the chosen wavelenght. The corresponding TolTEC wavelength and FWHM values obtained here: http://toltec.astro.umass.edu/about.php. The sigma (optical depth) values can be obtained here: http://toltec.astro.umass.edu/science_ultra_deep_survey.php.

We then set a value for the field size, assuming it covers a rectangular area. In this case, we will do a coverage of 0.2 square degrees.

We will use the following parameters when creating the simulated maps:

```
band = 1.1 # in millimeters  TolTEC bands = [1.1, 1.4, 2.0]
factor = 1.19 # TolTEC factors = [1.19, XX, XX]
FWHM = 5 # in arcseconds
sigma = 0.025 #Also called optical depth (rms), with units: mJy/bm
fieldSize = 0.2 # area in degree^2
```

Reading the Data

We will use the ascii package to read our catalog data. You'll notice that nothing happens when you run the cell below, but by uncommenting the second line, wou will see that our data appears as a table!

Note: The following catalog contains data from five different fields, so we will have to separate them.

```
data = ascii.read('3dhst_mmflux.cat', guess=False, format='basic')
#data
```

Separating our Data Into Separate Fields

After reading the data, it's time to separate the 3D-HST flux catalog into their corresponding fields. We do this by separating our data into five lists based on the field's right ascension and declination, as each field covers different parts of the sky, they don't overlap so we can separate our data in this way. You can uncomment the last line in the following cells to see the top part of our new data!

```
AEGIS = data[data["ra"] > 200]
#AEGIS
```

```
COSMOS_y = data[data["ra"] < 180]
COSMOS = COSMOS_y[COSMOS_y["ra"] > 140]
#COSMOS


GOODSN_y = data[data["ra"] < 200]
GOODSN = GOODSN_y[GOODSN_y["ra"] > 170]
#GOODSN


GOODSS = data[data["dec"] < -10]
#GOODSS


UDS_x = data[data["dec"] < 0]
UDS = UDS_x[UDS_x["dec"] > -10]
#UDS


### SANITY CHECK ###

#To ensure that all sources were assigned a field, we check the length original data to the
#length of the sum of all fields. If the sum of all fields is the same length as the origina
#data then every source has an assigned field!

#len() returns the number of items in an object, which in this case is an array.

print(len(data))
print(len(AEGIS)+len(COSMOS)+len(GOODSN)+len(GOODSS)+len(UDS))
```

```
63413
63413
```

Creating the Maps

After separating the 3D-HST catalog data into the five separate extragalactic fields, we can now start to create our simulated maps. For this step we will create four functions that we will use to create our Signal Map, our Noise Map, and our Final Combined Map.

> **Field Function**

First, we make a function that creates an empty 2D array with a given area and resolution. We will call this function when we create both our Signal Map and our Noise Map. You will notice that nothing happens when running the code cell, this is normal as we will call the function at a later time.

```
def makeField(area, res):
    '''
    This function takes an argument for area in squared degrees and converts it to a field c
    is a square. It also takes an argument for the resolution of each pixel in arcseconds ar
    Then, this function takes both the field dimension in degrees and divides it by the resc
    get a number of pixels for the field dimensions. Then, taking the number of pixels in bc
    then creates an empty 2D array with dimensions of the same number of pixels of length.
    '''
    # area = square degrees of desired field
    # res = resolution of each pixel in arcseconds
    fieldDim = np.sqrt(area) # dimensions for a square field
    degrees = (res/60)/60 # convert arcseconds to degrees
    pixelNumber = fieldDim/degrees # number of pixels dimension
    fieldArray = np.zeros((int(pixelNumber),int(pixelNumber))) #empty array with the dimensi
    return fieldArray #returns the empty array
```

Signal Map

Now that we have a function that will generate a blank field, we can start to generate our maps! Let's start with writing a function that will return a Signal Map for the given extragalactic field. This function that we will write will take the data of the desired extragalactic field and convolve it with the TolTEC PSF, which will ultimately mimic the expected data from the Ultra-Deep Galaxy Survey.

The Signal Map will show us how a specific part of the sky would look like assuming there is zero noise. This is helpful to have as some of the dimmest stars are lost in the noise from the telescope, the Earth's atmosphere, and the far-infrared background noise in outer-space.

In this function, we will input the data from the Ultra Deep Galaxy survey and the PSF data. The array of the PSF will be scaled to have the same array length as the array length of the survey field before being convolved and then normalized.

> A convolution is when two functions are combined to form a third function and it takes the form of the following integral (where f and g are two functions)

$$[f * g](t) \equiv \int_0^t f(\tau)\, g(t-\tau)\, d\tau,$$

This new function obtained by convolving both the PSF and the survey data is what will help us produce an image that will resemble real maps. Since in this simulation, all detected objects are unresolved (small, point-like objects), this convolution is necessary to be able to detect our simulated sources as it spreads the light from each object (while conserving their flux values) as it would when collected by the telescope, thus making it look like a real observed image.

```
def makeSignalMap(inputFlux, path_to_PSF, band, FWHM, fieldSize, fieldName):
    TolTEC_PSF_resized = fits.open(path_to_PSF)[0] #Opens the PSF. Since on the same directo

    newArray = makeField(fieldSize,1.0) #Makes an empty array using the makeField function.

    # Create WCS object
    w = wcs.WCS(naxis=2) #Creates a 2D World Coordinate System (WCS)
    w.wcs.ctype = ["RA---ARC", "DEC--ARC"] # zenithal/azimuthal equidistant

    # Gridding from WCS to Cartesian
    pixelCoords = w.all_world2pix(np.array(inputFlux['ra']), np.array(inputFlux['dec']), 1)
    xCoords = pixelCoords[0] #x-coordinates of the WCS
    yCoords = pixelCoords[1] #y-coordinates of the WCS

    # Broadcast to array the size of desired field
    oldXRange = max(xCoords) - min(xCoords) #Gives the range (length) of the x-axis
    oldYRange = max(yCoords) - min(yCoords) #Gives the range (length) of the y-axis
    '''
    The code grabs the x-coordinates and substracts its minumum value. It also grabs the ler
    and substracts 1 from it. The function then grabs these two new values, multiplies them
    the axis. This yields the new coordinate system for the x-axis. The same procedure is fo
    '''
    xCoordsNew = ((xCoords - min(xCoords))*(len(newArray)-1))/oldXRange #Sets up a new x-coo
    yCoordsNew = ((yCoords - min(yCoords))*(len(newArray)-1))/oldYRange #Sets up a new y-coo

    # Rounds coordinates to nearest integer and converts dtype to integer
    xInt = np.rint(xCoordsNew).astype(int)
    yInt = np.rint(yCoordsNew).astype(int)

    # Array of x, y, and flux densities from passbands
    for i in range(len(xInt)):
        newArray[yInt[i]][xInt[i]] =  newArray[yInt[i]][xInt[i]] + np.array(inputFlux['flux'
        #This loop fills the new empty array with the values of the now rounded new x and y
    signalArray = newArray #Defines this now filled new array to be the Signal Array
    dg_TolTEC=fftconvolve(signalArray,TolTEC_PSF_resized.data,mode='same') # [2:,0:]
        #This convolves the Signal Array and the data of the PSF into a single array with di
    dg_TolTEC_scaled = dg_TolTEC*(np.max(signalArray)/(np.max(dg_TolTEC)))
        #This scales this new convolved array by multiplying it by the value of the ratio be
        #the maximum value of the convolved array.
    signalImage = fits.PrimaryHDU(dg_TolTEC_scaled) #Creates a Header Data Unit for the scal
    signalImage.writeto('toltec_'+str(int(band*1000))+'_noNoise_'+str(int(fieldSize))+'sd_'+

    return signalArray, dg_TolTEC_scaled #When this function is called, it will return value
                                         #of the Scaled, convolved array formed by convolving
```

Noise Map

Now, we create the Noise Map by following a similar procedure as the previous functions.

We create our Noise Map by defining the noise as a random normal (Gaussian) distribution, with parameters for the noise distribution set by the FWHM of the detector's PSF and the anticipated depth of the image as defined by "sigma". For example, the UDGS has an expected depth of 0.025 mJy/beam at 1.1mm and thus this sets the value of sigma. Since this noise is simulated, artifacts resulting from cosmic rays or other sources of noise are not accounted for.

```
def makeNoiseMap(path_to_PSF, band, FWHM, fieldSize, sigma, factor):
    TolTEC_PSF_resized = fits.open(path_to_PSF)[0] #Opens the PSF. Since on the same directo
    SIZE = fieldSize # degree^2
    PXRES = 1.0 #Sets the pixel resolution in arcseconds

    blankMap = makeField(SIZE,PXRES) #Creates a new blank map using the makeField function,

    # Create WCS object (leave just in case need to output ds9)
    w = wcs.WCS(naxis=2) #Creates a 2D World Coordinate System (WCS). Same as in previous fu
    w.wcs.ctype = ["RA---ARC", "DEC--ARC"] # zenithal/azimuthal equidistant


    detectorFWHM = FWHM #Full width Half max (FWHM) of the detector in arcseconds. Value giv
    beamArea = (np.pi*(detectorFWHM)**2)/(4*np.log(2)) #Gives the area of the beam by knowir
    PX_PER_BEAM = beamArea*PXRES #Gives the number of pixels per beam by multiplying the bea
    noise = (np.random.normal(0,(sigma/np.sqrt(PX_PER_BEAM))*factor,len(blankMap.flatten()))




    noiseReshaped = np.reshape(noise, (-1, len(blankMap))) #Reshapes the noise into a 2D arr
    blankField_noise = blankMap + noiseReshaped #Takes the blank map and adds noise to it

    dg_blankField = fftconvolve(blankField_noise,TolTEC_PSF_resized.data,mode='same')
    #This convolves the Noise Field Array and the data of the PSF into a single array with c

    # Save file as a FITS file
    hdu = fits.PrimaryHDU(dg_blankField) #Creates a Header Data Unit for the convolved array
    hdu.writeto('toltec_'+str(int(band*1000))+'_n_rms'+str(sigma).replace('.', 'p')+'_'+str(

    return blankField_noise, TolTEC_PSF_resized, dg_blankField
        #This returns the noise map, the PSF, and the convolved noise-PSF array
```

Final Map

Finally, we combine the Signal and Noise Maps in the following function to get our Final Map.

This Final Map is how we should expect a specific portion of the sky to look like using TolTEC's 1.1mm band. By having these simulated maps, we can predict how future observations using the LMT might look like if we have a flux catalog of the desired field to be observed, and we also have some predicted data to compare real observations to.

```
def makeSignalNoiseMap(SignalMap, NoiseMap, fieldName):
    '''
    This function takes the signal map and the noise map and sums them together to create th
    and converts it into a FITS file and saves this map in the given destination. In this ca
    that I have all the data for this notebook.
    '''
    finalMap = SignalMap + NoiseMap #adds the signal and noise to create a simulated map
    corners=np.where(SignalMap==0)
    finalMap[corners]=0
    finalMap_FITS = fits.PrimaryHDU(finalMap) #converts the simulated map into a fits file
    finalMap_FITS.writeto('toltec_'+str(int(band*1000))+'_rms'+str(sigma).replace('.', 'p')+

    return finalMap #returns the final simulated map
```

Generating our Maps

Signal Maps

Having all of our functions defined, we can call our Signal Map function to create a Signal Map for all five of our fields, we call the function by giving it the desired field map, the TolTEC PSF, and the respective parameters corresponding to TolTEC's 1.1mm band. After running the code cell below, you will notice that five maps will appear to the left side on "Files."

```
'''
sigarray = Filled Signal Array Map
dg = Scaled, convolved array formed by convolving the Signal Array and the given PSF
'''
sigarray_AEGIS, dg_AEGIS = makeSignalMap(AEGIS, 'toltec_1100_PSF.fits', band, FWHM, fieldSiz
sigarray_COSMOS, dg_COSMOS = makeSignalMap(COSMOS, 'toltec_1100_PSF.fits', band, FWHM, field
sigarray_GOODSN, dg_GOODSN = makeSignalMap(GOODSN, 'toltec_1100_PSF.fits', band, FWHM, field
sigarray_GOODSS, dg_GOODSS = makeSignalMap(GOODSS, 'toltec_1100_PSF.fits', band, FWHM, field
sigarray_UDS, dg_UDS = makeSignalMap(UDS, 'toltec_1100_PSF.fits', band, FWHM, fieldSize, 'UD
```

Noise Map

Only one noise map needs to be generated for all of our signal maps. We will generate our map by calling our Noise Map function. Since we're only generating noise, we don't have to give the function a field map, only the TolTEC PSF and it's corresponding parameters.

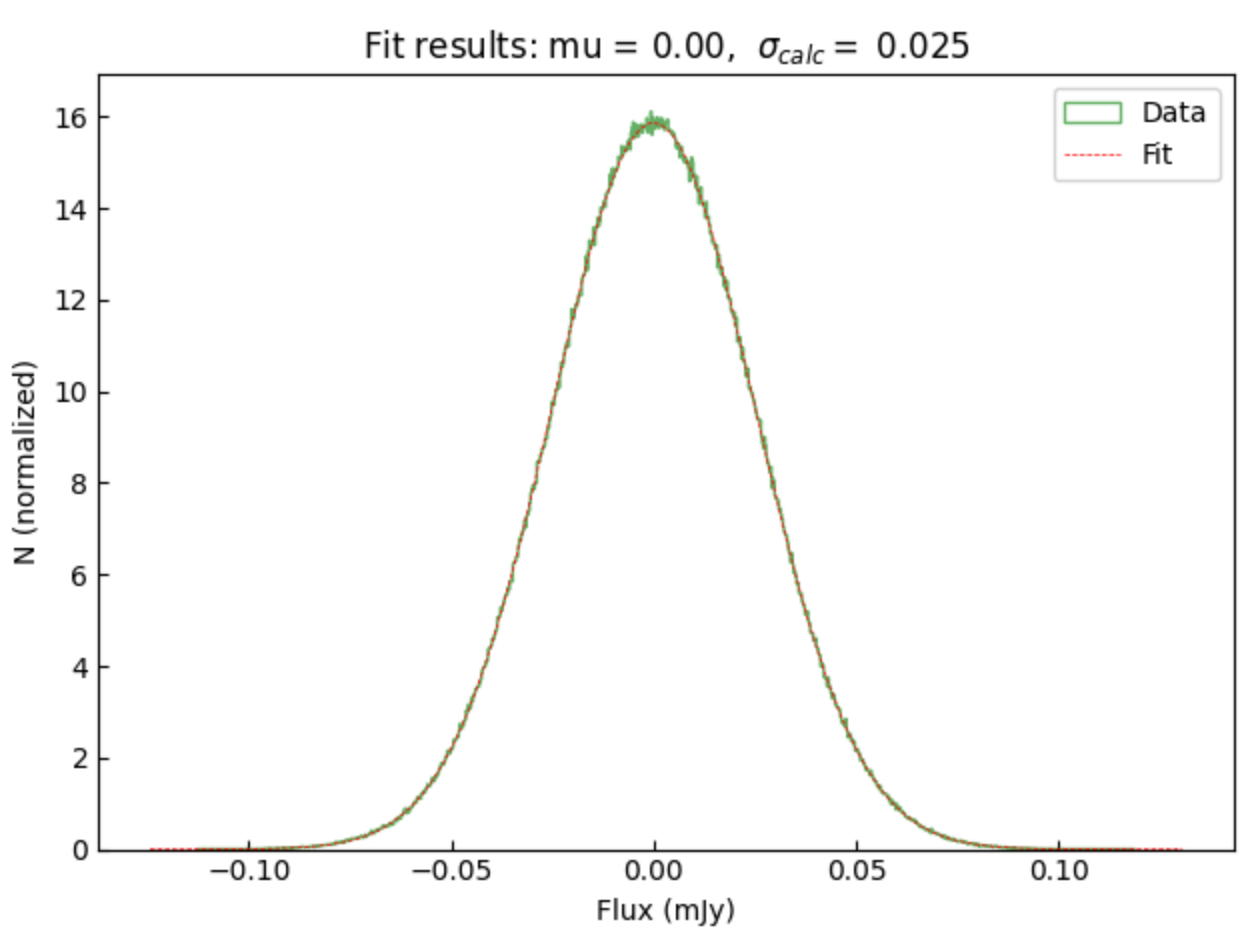
```
'''
rawNoise = Noise Map
psf = Original PSF (unmodified by function)
```

```
noiseMap = Convolved Noise-PSF array
'''
rawNoise, psf, noiseMap = makeNoiseMap('toltec_1100_PSF.fits', band, FWHM, fieldSize, sigma,
```

Noise Map Histogram

We can see how our Noise was distributed by plotting a flattened (One Dimensional Array) version of it below. Since our Noise was simulated using a normal (Gaussian) distribution, our Noise resembles an almost perfect Gaussian curve.

```
plt.figure(3)
noise_data = noiseMap.flatten() #Turns the Noise Map into a 1D array

# Fit a normal distribution to the data:
mu, std = norm.fit(noise_data) #Gets a value for mu and the standard deviation from the 1D N

# Plot the 1D Noise Map into a histogram.
plt.hist(noise_data, bins=900, density=True, alpha=0.6, color='g', histtype='step', label='D

# Plot the Probability Density Function (PDF).
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100) #Prints 100 evenly-spaced values in between the x-min and x
p = norm.pdf(x, mu, std) #Returns the Probability Density Function (PDF) at each of the x-va
plt.plot(x, p, 'r--', linewidth=0.5, label='Fit')
title = "Fit results: mu = %.2f,  $\sigma_{calc} = $ %.3f" % (mu, std)
plt.title(title)
plt.xlabel("Flux (mJy)"); plt.ylabel("N (normalized)")
plt.tick_params(direction="in")
plt.legend()
plt.tight_layout()
plt.show()
```



Final Maps

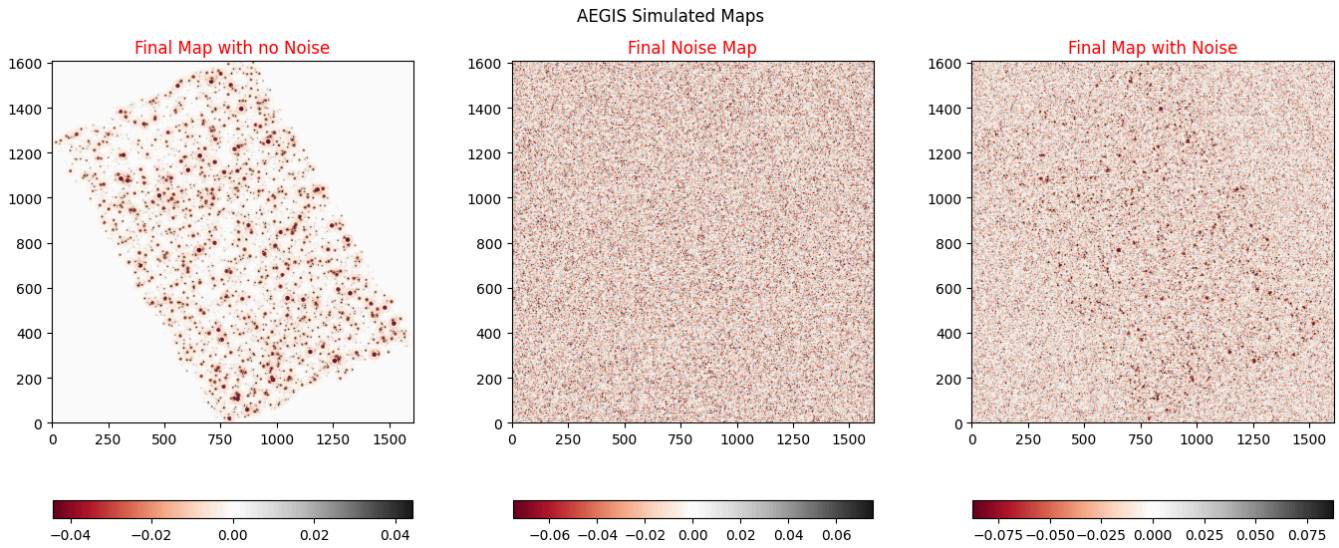
Now that we have all of our Signal Maps and the Noise Maps, we will call our Final Map function in order to add the Noise Map to all five of our Signal Maps. All these new maps will get saved in the notebook and will appear to the left.

```
'''
final = Final Map created by combining the Scaled, convolved array formed by convolving the
'''
final_AEGIS = makeSignalNoiseMap(dg_AEGIS, noiseMap, 'AEGIS')
final_COSMOS = makeSignalNoiseMap(dg_COSMOS, noiseMap, 'COSMOS')
final_GOODSN = makeSignalNoiseMap(dg_GOODSN, noiseMap, 'GOODSN')
final_GOODSS = makeSignalNoiseMap(dg_GOODSS, noiseMap, 'GOODSS')
final_UDS = makeSignalNoiseMap(dg_UDS, noiseMap, 'UDS')
```

Plotting the AEGIS Maps

After obtaining all our simulated maps, we can check them by plotting them! We will do this by comparing the Signal Map to the Noise Map and to the Final Simulated Map. We will show our results for the AEGIS Field.

```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('AEGIS Simulated Maps')

#dg = Scaled, convulged array formed by convolving the Signal Array and the given PSF
#Saved as FITS file as toltec_1100noNoise_0sd_AEGIS_Signal.fits

opt_S_A = dict(vmin=-3*np.std(dg_AEGIS), vmax=3*np.std(dg_AEGIS), cmap='RdGy')
#opt_S_A = dict(vmin=-0.012, vmax=0.2, cmap='plasma')

signal_AEGIS = ax1.imshow(dg_AEGIS, origin='lower', **opt_S_A)
ax1.set_title('Final Map with no Noise',color='r',fontsize=12)


#noiseMap = Convolved Noise-PSF array
#Saved as FITS file as toltec_1100_n_rms0p025_0sd_Noise.fits

opt_N = dict(vmin=-3*np.std(noiseMap), vmax=3*np.std(noiseMap), cmap='RdGy')
#opt_N = dict(vmin=-0.001, vmax=0.2, cmap='plasma')

noise = ax2.imshow(noiseMap, origin='lower', **opt_N)
ax2.set_title('Final Noise Map',color='r',fontsize=12)


#final = Final Map created by combining the scaled, convolved array formed by convolving the
#Saved as FITS file as toltec_1100_rms0p025_0sd_AEGIS_Final.fits

opt_F_A = dict(vmin=-3*np.std(final_AEGIS), vmax=3*np.std(final_AEGIS), cmap='RdGy')
#opt_F_A = dict(vmin=-0.001, vmax=0.2, cmap='plasma')

final_A = ax3.imshow(final_AEGIS, origin='lower', **opt_F_A)
ax3.set_title('Final Map with Noise',color='r',fontsize=12)

plt.colorbar(signal_AEGIS, ax=ax1, location='bottom', shrink=0.9)
plt.colorbar(noise, ax=ax2, location='bottom', shrink=0.9)
plt.colorbar(final_A, ax=ax3, location='bottom', shrink=0.9)
plt.tight_layout()
plt.show()
```



Comparing Each Map

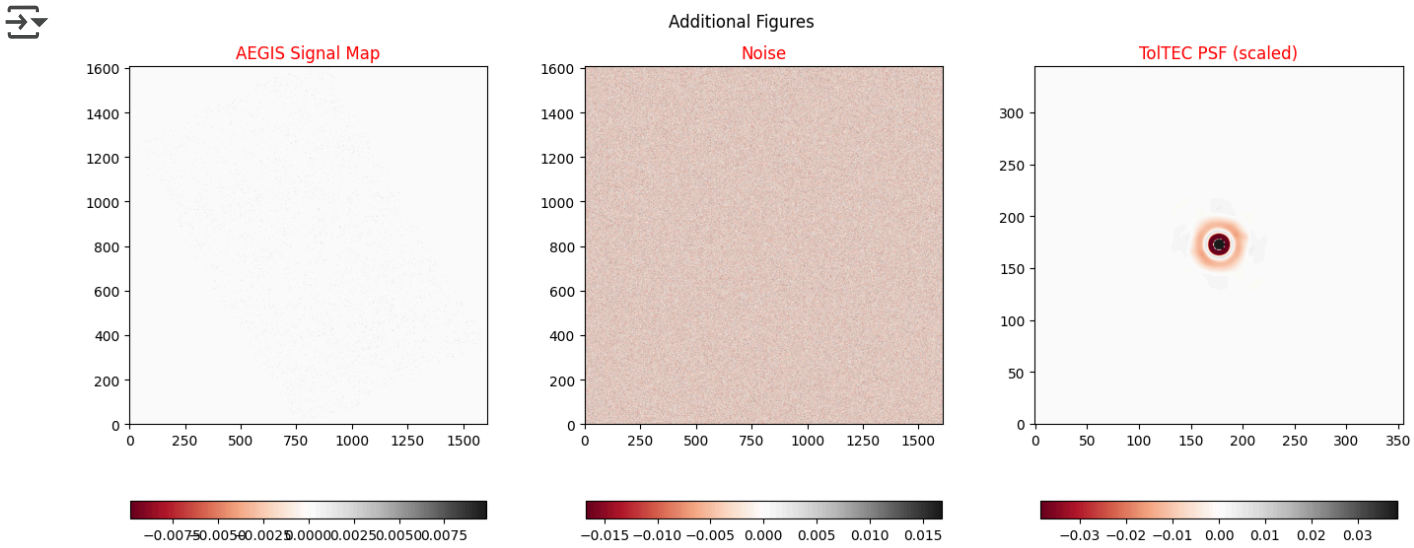
The Signal Map is what we will be observing in an ideal world with no noise, sadly for us this is not the case in the real world. This map shows the signal of each source after applying TolTEC's PSF

Checking our Original Data Before Convolved to a PSF

The following figures show the original Signal from Aegis and our original Noise before both of the maps being convolved with the TolTEC 1.1mm band PSF, as well as the Point Spread Function of the TolTEC camera.

```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('Additional Figures')

#sigarray = Filled Signal Array Map
```

```
opt_s_A = dict(vmin=-3*np.std(sigarray_AEGIS), vmax=3*np.std(sigarray_AEGIS), cmap='RdGy')

sky_AEGIS = ax1.imshow(sigarray_AEGIS, origin='lower', **opt_s_A)
ax1.set_title('AEGIS Signal Map',color='r',fontsize=12)


#rawNoise = Noise Map

opt_n = dict(vmin=-3*np.std(rawNoise), vmax=3*np.std(rawNoise), cmap='RdGy')

r_noise = ax2.imshow(rawNoise, origin='lower', **opt_n)
ax2.set_title('Noise',color='r',fontsize=12)


#psf = Original PSF (unmodified by function)

opt_P = dict(vmin=-3*np.std(psf.data), vmax=3*np.std(psf.data), cmap='RdGy')

psf_1100 = ax3.imshow(psf.data, origin='lower', **opt_P)
ax3.set_title('TolTEC PSF (scaled)',color='r',fontsize=12)

plt.colorbar(sky_AEGIS, ax=ax1, location='bottom', shrink=0.9)
plt.colorbar(r_noise, ax=ax2, location='bottom', shrink=0.9)
plt.colorbar(psf_1100, ax=ax3, location='bottom', shrink=0.9)
plt.tight_layout()
plt.show()
```



As you can see, when not convolving the signal data with the PSF data, the point sources are very dim and unresolved. This is because the original sources are compact single pixels, meaning that they are concentrated and unresolved. Convolving them with a PSF makes it so that the light of each source is spread out, with each source conserving it's flux value but appearing brighter. Using a PSF is an inevitable part of observations (comes from the optical setup of a telescope) and we need to convolve the point sources with a PSF in order to produce an image that will resemble real telescope observations.

Due to the poor resolution of the LMT and due to it being a land based telescope, observed galaxies appear as unresolved point sources as they are below the resolution limit of both the LMT and TolTEC. This is helpful when creating mock observation images as we can take the total flux of an unresolved object and make it a point source at the resolution of the TolTEC camera without having to worry if it's a galaxy or something else. This only applies to unresolved sources though.

Discussion / Future Applications

To summarize, we create a signal map by convoluting our source data with the PSF of LMT/TolTEC at 1.1mm, along with creating a noise map defined by random noise from a normal distribution. These two maps are then added to create our simulated map for the Aegis extragalactic field map, showing how this portion of the Ultra-Deep Galaxy Survey would potentially look given the anticipated performance of LMT/TolTEC.

Future applications of the code written in this notebook could be to simulate other extragalactic surveys or to simulate the same surveys with a different PSF, as to see how a specific telescope's PSF will make the same data to change.

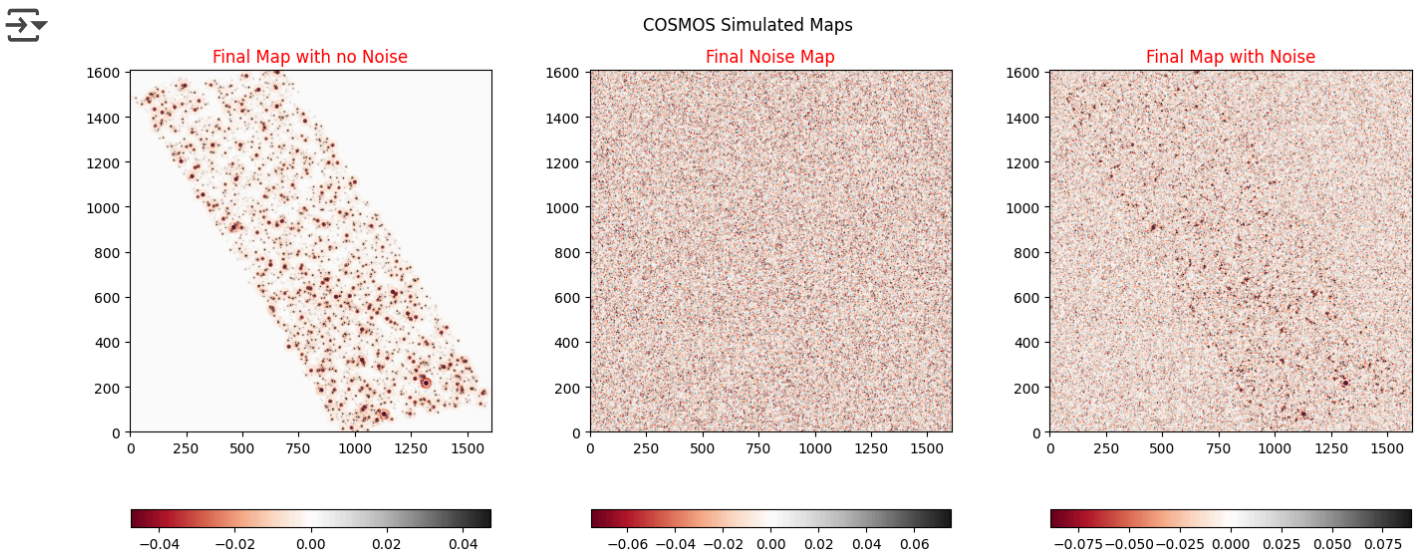
Ultimately, these simulations will be of help when gathering real data of submillimeter galaxies as we would be able to compare predictions to observations.

> **(Optional) Plotting the rest of our Maps**

We can check the maps for the other four fields by running the following four code cells!

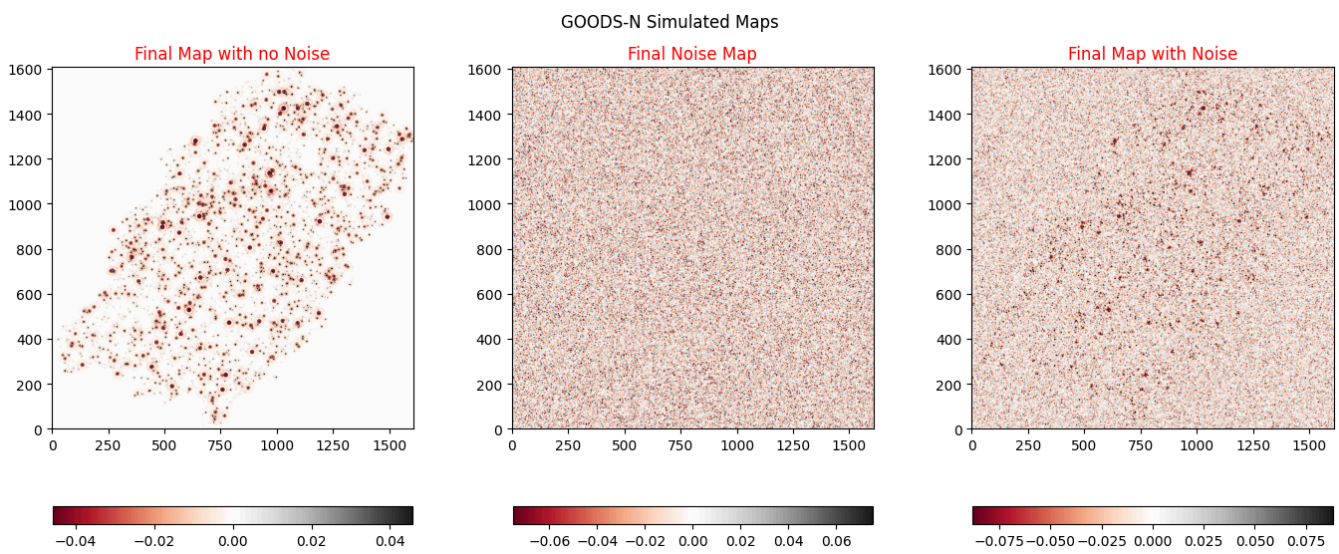
```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('COSMOS Simulated Maps')

#dg = Scaled, convulged array formed by convolving the Signal Array and the given PSF
#Saved as FITS file as toltec_1100noNoise_0sd_COSMOS_Signal.fits

opt_S_C = dict(vmin=-3*np.std(dg_COSMOS), vmax=3*np.std(dg_COSMOS), cmap='RdGy')

signal_COSMOS = ax1.imshow(dg_COSMOS, origin='lower', **opt_S_C)
ax1.set_title('Final Map with no Noise',color='r',fontsize=12)


#noiseMap = Convolved Noise-PSF array
#Saved as FITS file as toltec_1100_n_rms0p025_0sd_Noise.fits

opt_N = dict(vmin=-3*np.std(noiseMap), vmax=3*np.std(noiseMap), cmap='RdGy')

noise = ax2.imshow(noiseMap, origin='lower', **opt_N)
ax2.set_title('Final Noise Map',color='r',fontsize=12)


#final = Final Map created by combining the scaled, convolved array formed by convolving the
#Saved as FITS file as toltec_1100_rms0p025_0sd_COSMOS_Final.fits

opt_F_C = dict(vmin=-3*np.std(final_COSMOS), vmax=3*np.std(final_COSMOS), cmap='RdGy')

final_C = ax3.imshow(final_COSMOS, origin='lower', **opt_F_C)
```

```
ax3.set_title('Final Map with Noise',color='r',fontsize=12)

plt.colorbar(signal_COSMOS, ax=ax1, location='bottom', shrink=0.9)
plt.colorbar(noise, ax=ax2, location='bottom', shrink=0.9)
plt.colorbar(final_C, ax=ax3, location='bottom', shrink=0.9)
plt.tight_layout()
plt.show()
```




```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('GOODS-N Simulated Maps')

#dg = Scaled, convulged array formed by convolving the Signal Array and the given PSF
#Saved as FITS file as toltec_1100noNoise_0sd_GOODSN_Signal.fits

opt_S_GN = dict(vmin=-3*np.std(dg_GOODSN), vmax=3*np.std(dg_GOODSN), cmap='RdGy')

signal_GOODSN = ax1.imshow(dg_GOODSN, origin='lower', **opt_S_GN)
ax1.set_title('Final Map with no Noise',color='r',fontsize=12)
```

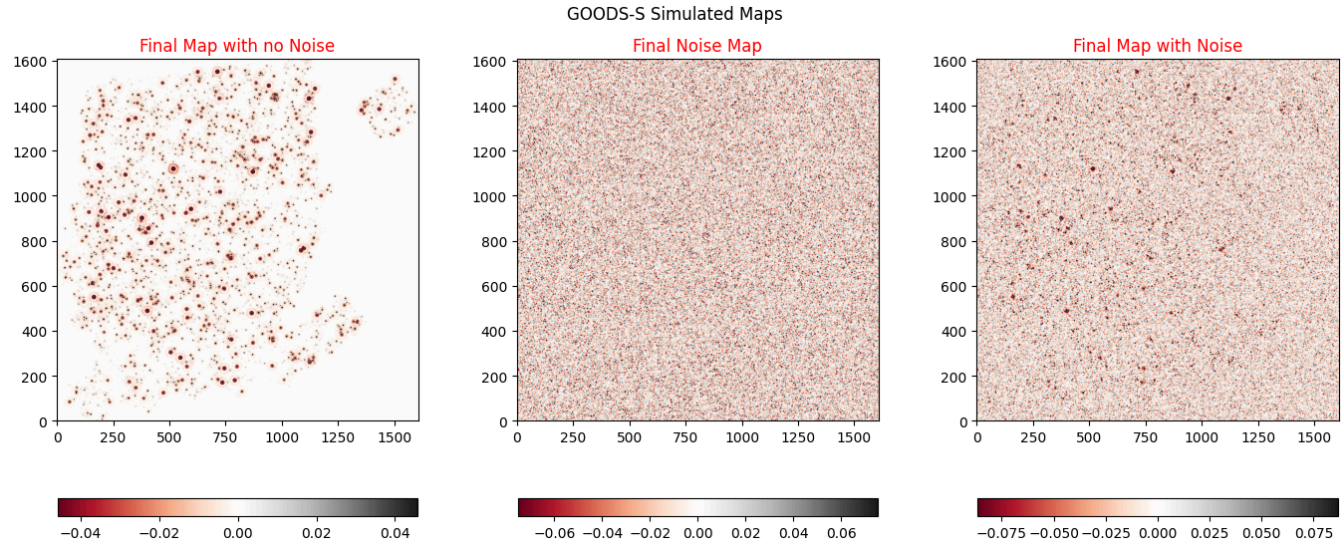
```
#noiseMap = Convolved Noise-PSF array
#Saved as FITS file as toltec_1100_n_rms0p025_0sd_Noise.fits

opt_N = dict(vmin=-3*np.std(noiseMap), vmax=3*np.std(noiseMap), cmap='RdGy')

noise = ax2.imshow(noiseMap, origin='lower', **opt_N)
ax2.set_title('Final Noise Map',color='r',fontsize=12)


#final = Final Map created by combining the scaled, convolved array formed by convolving the
#Saved as FITS file as toltec_1100_rms0p025_0sd_GOODSN_Final.fits

opt_F_GN = dict(vmin=-3*np.std(final_GOODSN), vmax=3*np.std(final_GOODSN), cmap='RdGy')

final_GN = ax3.imshow(final_GOODSN, origin='lower', **opt_F_GN)
ax3.set_title('Final Map with Noise',color='r',fontsize=12)

plt.colorbar(signal_GOODSN, ax=ax1, location='bottom', shrink=0.9)
plt.colorbar(noise, ax=ax2, location='bottom', shrink=0.9)
plt.colorbar(final_GN, ax=ax3, location='bottom', shrink=0.9)
plt.tight_layout()
plt.show()
```




```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('GOODS-S Simulated Maps')

#dg = Scaled, convulged array formed by convolving the Signal Array and the given PSF
#Saved as FITS file as toltec_1100noNoise_0sd_GOODSS_Signal.fits

opt_S_GS = dict(vmin=-3*np.std(dg_GOODSS), vmax=3*np.std(dg_GOODSS), cmap='RdGy')

signal_GOODSS = ax1.imshow(dg_GOODSS, origin='lower', **opt_S_GS)
ax1.set_title('Final Map with no Noise',color='r',fontsize=12)


#noiseMap = Convolved Noise-PSF array
#Saved as FITS file as toltec_1100_n_rms0p025_0sd_Noise.fits

opt_N = dict(vmin=-3*np.std(noiseMap), vmax=3*np.std(noiseMap), cmap='RdGy')

noise = ax2.imshow(noiseMap, origin='lower', **opt_N)
ax2.set_title('Final Noise Map',color='r',fontsize=12)
```

```
#final = Final Map created by combining the scaled, convolved array formed by convolving the
#Saved as FITS file as toltec_1100_rms0p025_0sd_GOODSS_Final.fits

opt_F_GS = dict(vmin=-3*np.std(final_GOODSS), vmax=3*np.std(final_GOODSS), cmap='RdGy')

final_GS = ax3.imshow(final_GOODSS, origin='lower', **opt_F_GS)
ax3.set_title('Final Map with Noise',color='r',fontsize=12)

plt.colorbar(signal_GOODSN, ax=ax1, location='bottom', shrink=0.9)
plt.colorbar(noise, ax=ax2, location='bottom', shrink=0.9)
plt.colorbar(final_GS, ax=ax3, location='bottom', shrink=0.9)
plt.tight_layout()
plt.show()
```

```
<ipython-input-25-8d9ae80f0287>:30: UserWarning: Adding colorbar to a different Figure <
  plt.colorbar(signal_GOODSN, ax=ax1, location='bottom', shrink=0.9)
```




```
fig,((ax1),(ax2),(ax3)) = plt.subplots(1, 3, figsize=(14,6))
fig.suptitle('UDS Simulated Maps')
```

```
#dg = Scaled, convulged array formed by convolving the Signal Array and the given PSF
#Saved as FITS file as toltec_1100noNoise_0sd_UDS_Signal.fits

opt_S_U = dict(vmin=-3*np.std(dg_UDS), vmax=3*np.std(dg_UDS), cmap='RdGy')

signal_UDS = ax1.imshow(dg_UDS, origin='lower', **opt_S_U)
ax1.set_title('Final Map with no Noise',color='r',fontsize=12)


#noiseMap = Convolved Noise-PSF array
#Saved as FITS file as toltec_1100_n_rms0p025_0sd_Noise.fits
```