Data Fitting and Analysis

This is a short exercise to apply the skills taught in the 'Line Profile Measurement' lecture material.

In [1]:
```
from astropy import units as u, constants
from astropy import visualization
import pylab as pl
import numpy as np
pl.style.use('dark_background')
visualization.quantity_support()
```

Out[1]:
```
<astropy.visualization.units.quantity_support.<locals>.MplQuantityConverter at
0x210a9538730>
```

In [2]:
```
from astropy.io import fits
from astropy.table import Table
```

We should have 4 data sets: 1, 3, 20, and 250 integration calibration spectra.

(these represent the *average* over that many "integrations", where each integration is comprised of the spectrograph recording n(pixels)=8192 samples, then taking the Fourier transform of those samples)

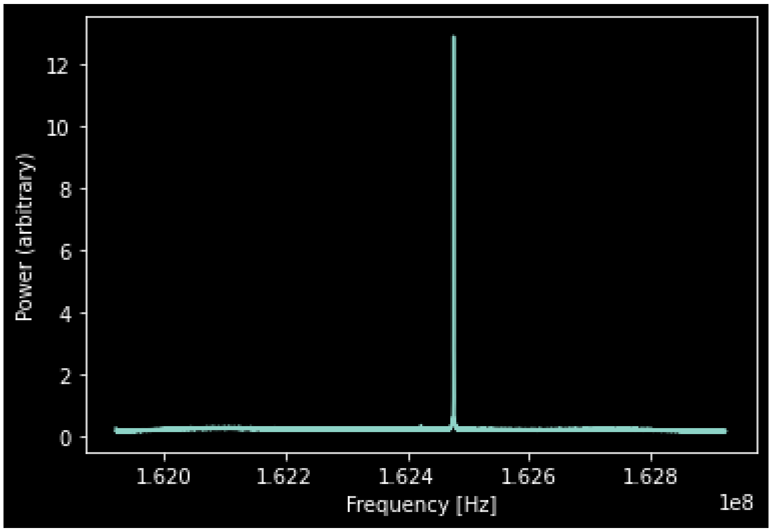
In [3]:
```
ls data
```

```
 Volume in drive C has no label.
 Volume Serial Number is EC8D-ED0B

 Directory of C:\Users\Sydnee O'Donnell\OneDrive\UF\Obs Tech 2\Classwork\data

10/26/2022  05:51 PM    <DIR>          .
10/26/2022  07:57 PM    <DIR>          ..
10/26/2022  05:43 PM           138,240 calibration_spectrum_1integrations.fits
10/26/2022  05:43 PM           138,240 calibration_spectrum_250integrations.fit
s
               2 File(s)        276,480 bytes
               2 Dir(s)  108,046,802,944 bytes free
```

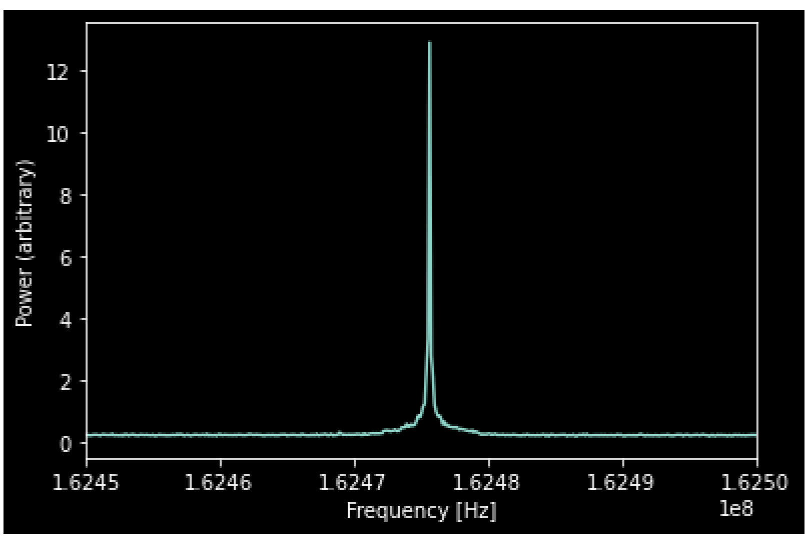
Initial example: Load a spectrum and plot it

In [4]:
```
tbl = Table.read('data/calibration_spectrum_250integrations.fits')
```

In [5]:
```
pl.plot(tbl['freq'].quantity, tbl['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```

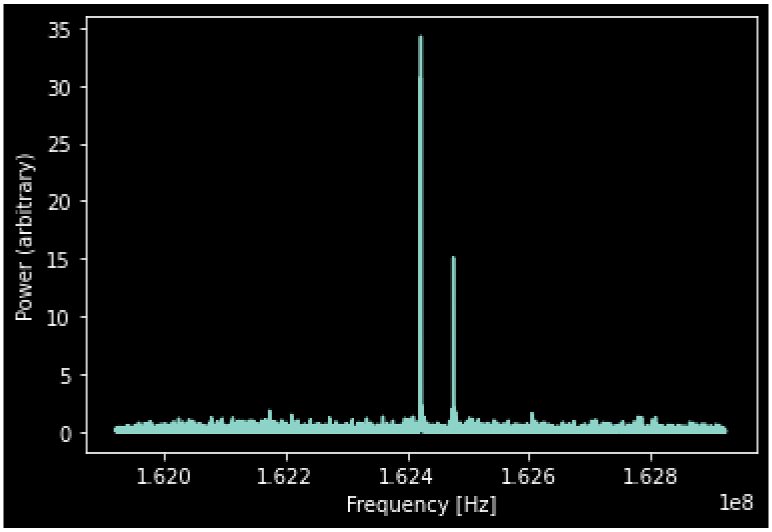


In [6]:
```
pl.plot(tbl['freq'].quantity, tbl['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
pl.xlim(162.45*u.MHz, 162.5*u.MHz);
```



Your work starts here

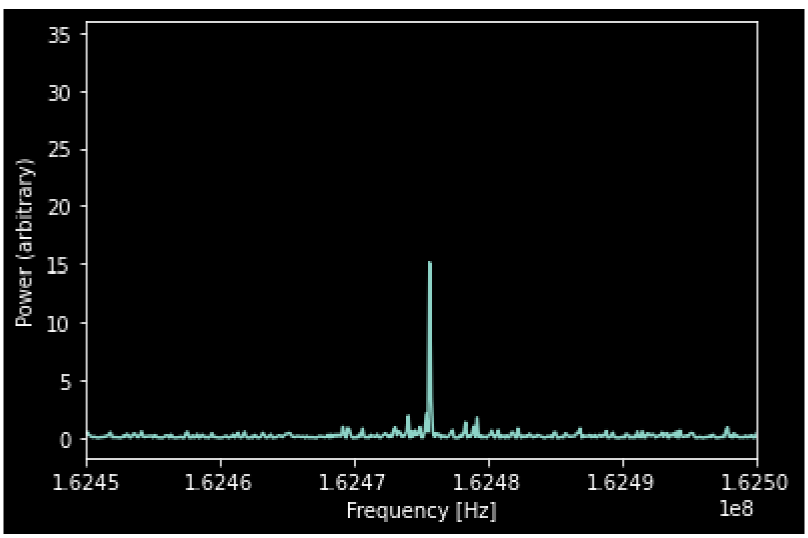
In [7]:
```
# plot the noisiest spectrum (the 1-integration version) vs. frequency
tbl2 = Table.read('data/calibration_spectrum_1integrations.fits')

pl.plot(tbl2['freq'].quantity, tbl2['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```



In [8]:
```
tbl2['freq'].quantity.shape
```

Out[8]:
```
(8192,)
```

In [9]:
```
pl.plot(tbl2['freq'].quantity, tbl2['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
pl.xlim(162.45*u.MHz, 162.5*u.MHz);
```



from the graph above I am going to select my spectrum to be between 16247 and 16248

In [10]:
```
## changing into a numpy array to manipulate
freq = tbl2['freq'].quantity
freq = np.array(freq)
```

In [11]:
```
## func to find value closest to what I'm looking for
def closest_value(input_list, input_value):
    arr = np.asarray(input_list)
    i = (np.abs(arr - input_value)).argmin()
    return arr[i]

num1 = int(162470000)
num2 = int(162480000)

val1 = closest_value(freq,num1)
val2 = closest_value(freq,num2)

print("The closest value to the "+ str(num1)+" is",val1)
print("The closest value to the "+ str(num2)+" is",val2)
```

```
The closest value to the 162470000 is 162470047.18876764
The closest value to the 162480000 is 162480056.9546578
```

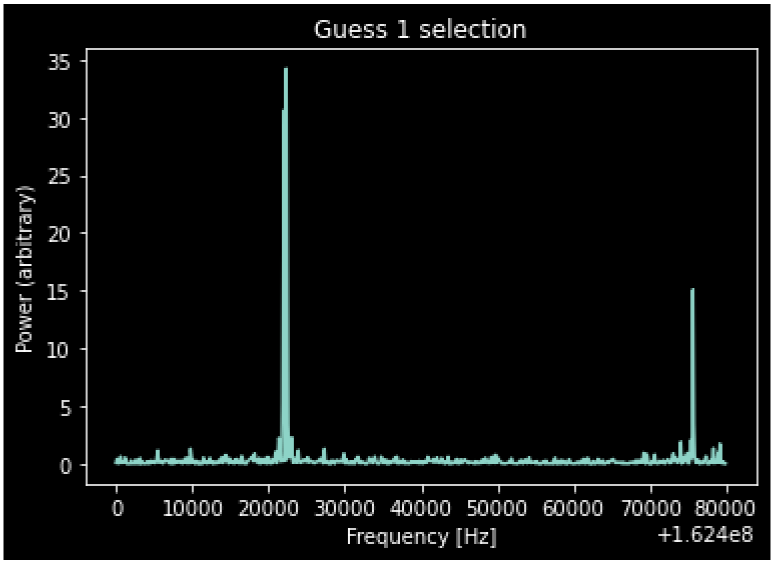
In [12]:
```
np.where(freq == 162399978.82753646)[0]
```

Out[12]:
```
array([3914], dtype=int64)
```

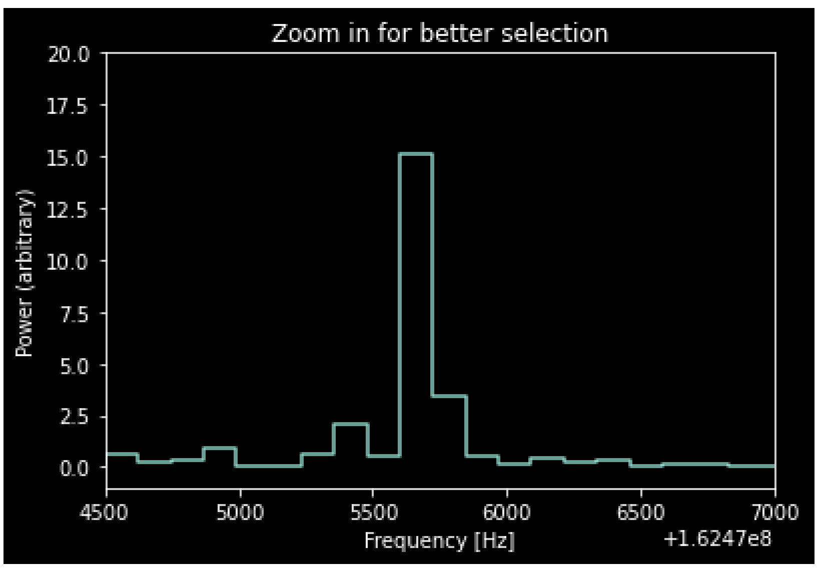
In [13]:
```
np.where(freq == 162480056.9546578)[0]
```

Out[13]:
```
array([4570], dtype=int64)
```

In [14]:
```
# plot a selected subset of the noisy spectrum.
# It should be the subset that includes the line!
selection = freq[3914:4570]
power_sel = tbl2['power'][3914:4570]

pl.plot(selection, power_sel)
pl.title("Guess 1 selection")
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```



In [15]:
```
## zooming in to make a better guess
pl.plot(tbl2['freq'].quantity, tbl2['power'], drawstyle='steps-mid');
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)")
pl.title("Zoom in for better selection")
pl.xlim(162.4745*u.MHz, 162.477*u.MHz)
pl.ylim(-1,20);
```



In [16]:
```
## finding closest values to make slice
num3 = int(162474500)
num4 = int(162477000)

val3 = closest_value(freq,num3)
val4 = closest_value(freq,num4)

print("The closest value to the "+ str(num3)+" is",val3)
print("The closest value to the "+ str(num4)+" is",val4)
```

```
The closest value to the 162474500 is 162474441.72013405
The closest value to the 162477000 is 162477005.19676447
```

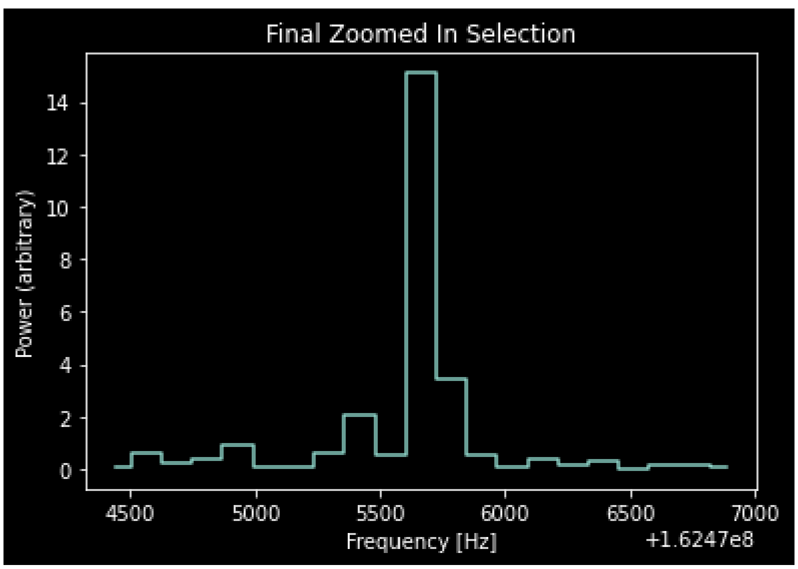
In [17]:
```
np.where(freq == 162474441.72013405)[0]
```

Out[17]:
```
array([4524], dtype=int64)
```

In [18]:
```
np.where(freq == 162477005.19676447)[0]
```

Out[18]:
```
array([4545], dtype=int64)
```

In [19]:
```
## final selection
selection2 = freq[4524:4545]
power_sel2 = tbl2['power'][4524:4545]

pl.plot(selection2, power_sel2, drawstyle='steps-mid')
pl.title("Final Zoomed In Selection")
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```



In [20]:
```
np.max(selection2)
```

Out[20]:
```
162476883.12644872
```

Analyze the 250 integration spectrum

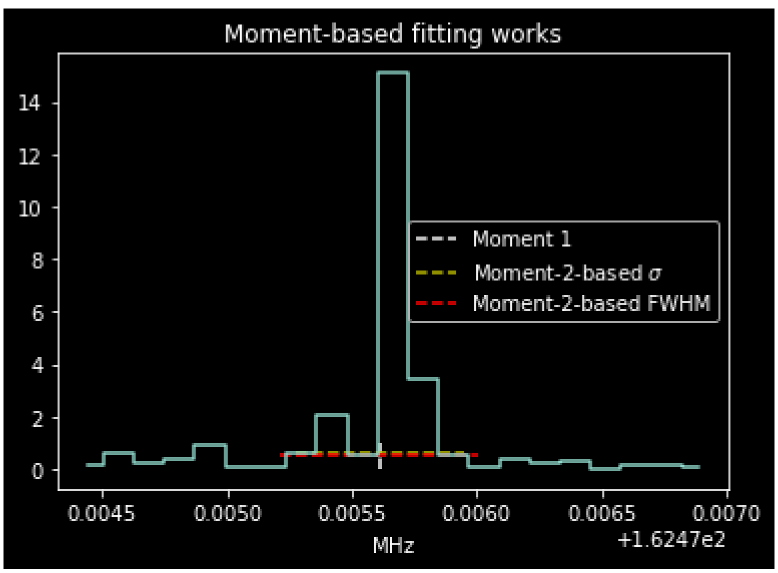
In [21]:
```
frequency_axis = selection2*u.Hz
frequency_axis = frequency_axis.to(u.MHz)
line_profile = power_sel2
```

In [23]:
```
# example moment0 / integrated intensity measurement we might make for an HI line
delta_nu = selection2[1] - selection2[0]
```

In [24]:
```
# moments on the downselected data
moment0 = (line_profile * delta_nu).sum()
moment1 = (frequency_axis * line_profile * delta_nu).sum() / moment0
moment2 = ( (frequency_axis - moment1)**2 * line_profile * delta_nu).sum() / mome
sigma = moment2**0.5
moment0, moment1, moment2, sigma
```

Out[24]:
```
(3177.5211575176213,
 <Quantity 162.47560921 MHz>,
 <Quantity 1.14238143e-07 MHz2>,
 <Quantity 0.00033799 MHz>)
```

In [25]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
#pl.xlim(1420.36,1420.45)
pl.vlines(moment1, 0*u.Jy, 1*u.Jy, color='w', linestyle='--', label='Moment 1')
# the Gaussian width is the half-width at exp(-1/2)
pl.hlines( np.exp(-0.5), moment1-sigma, moment1+sigma, color='y', linestyle='--',
pl.hlines( 0.5, moment1-sigma*2.35/2, moment1+sigma*2.35/2, color='r', linestyle=
pl.legend(loc='best');
pl.title("Moment-based fitting works");
```

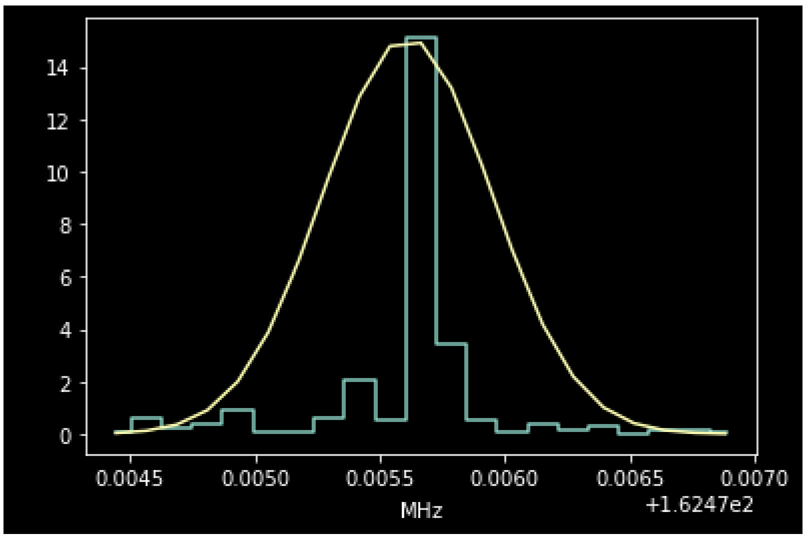


In [26]:
```
from astropy.modeling.models import Gaussian1D

guess_gaussian = Gaussian1D(amplitude=line_profile.max(),
                            mean=moment1,
                            stddev=sigma
                           )
guess_gaussian
```

Out[26]: <Gaussian1D(amplitude=15.093981, mean=162.47560921 MHz, stddev=0.00033799 MHz)>

In [27]:
```
pl.plot(frequency_axis,line_profile,drawstyle='steps-mid')
pl.plot(frequency_axis, guess_gaussian(frequency_axis))
```

Out[27]: [<matplotlib.lines.Line2D at 0x210aad8bbb0>]

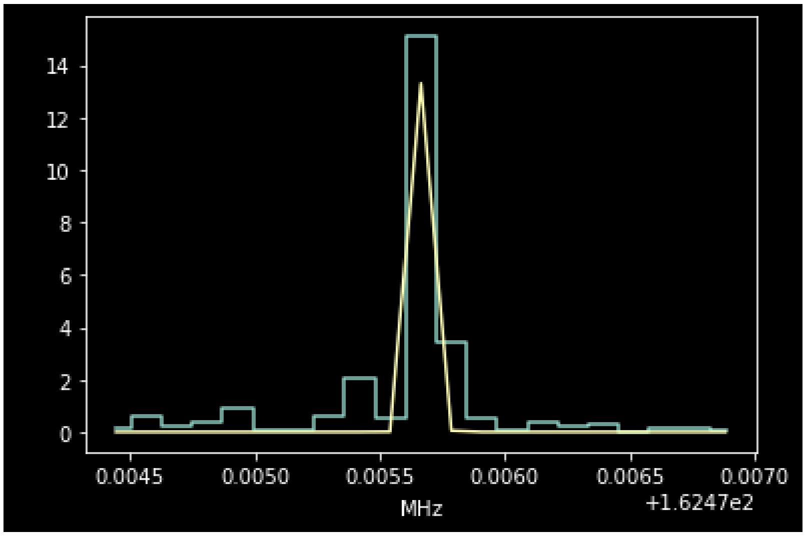


In [28]:
```
from astropy.modeling.fitting import LevMarLSQFitter
fitter = LevMarLSQFitter()
fitted_gaussian = fitter(guess_gaussian, frequency_axis, line_profile)
fitted_gaussian
```

Out[28]: <Gaussian1D(amplitude=13.73222703, mean=162.47567101 MHz, stddev=0.00003394 MH
z)>

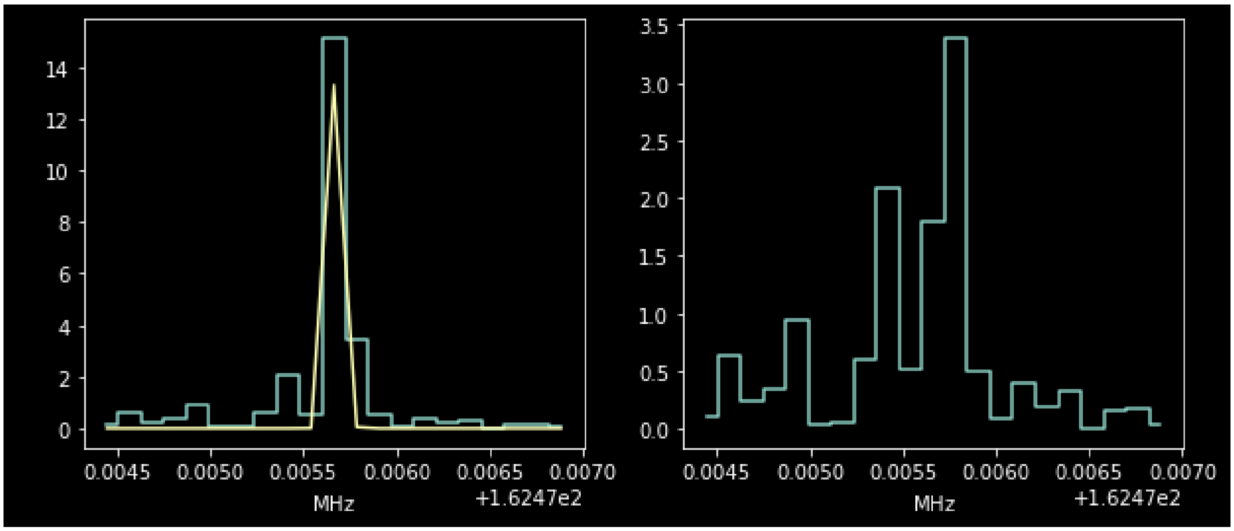
In [29]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
pl.plot(frequency_axis, fitted_gaussian(frequency_axis))
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x210abae8730>]
```



In [31]:
```
model = fitted_gaussian(frequency_axis)
residual = line_profile - model

pl.figure(figsize=(10,4))
pl.subplot(1,2,1).plot(frequency_axis, line_profile,
            drawstyle='steps-mid')
pl.plot(frequency_axis, fitted_gaussian(frequency_axis))

pl.subplot(1,2,2).plot(frequency_axis, residual,
            drawstyle='steps-mid')
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x210abb90250>]
```



In [32]:
```
print(fitter.fit_info['param_cov'])
```

```
[[ 1.04454004e+16  1.37946339e+09 -4.93739575e+10]
 [-6.17522385e+09  8.20474698e+04  2.08451501e+05]
 [-6.57174278e+10  1.70583156e+05  6.98444663e+05]]
```

In [33]:
```
print(fitter.fit_info['param_cov'])

print(f"Peak is {fitted_gaussian.amplitude.value} +/-"
      f" {fitter.fit_info['param_cov'][0,0]**0.5}")

print(f"Centroid is {fitted_gaussian.mean.value} +/-"
      f" {fitter.fit_info['param_cov'][1,1]**0.5}")

print(f"Width is {fitted_gaussian.stddev.value} +/-"
      f" {fitter.fit_info['param_cov'][2,2]**0.5}")
```

```
Peak is 13.732227029646799 +/- 102202741.42389116
```

In [36]:
```
print(f"Centroid is {fitted_gaussian.mean.value:0.5f} +/- {fitter.fit_info['paran
print(f"Width is {fitted_gaussian.stddev.value:0.5f} +/- {fitter.fit_info['param_
print()
print(f"m1 = {moment1:0.5f}")
print(f"sqrt(m2) = {moment2**0.5:0.5f}")
```

```
Centroid is 162.47567 +/- 286.43930
Width is 0.00003 +/- 835.73002

m1 = 162.47561 MHz
sqrt(m2) = 0.00034 MHz
```

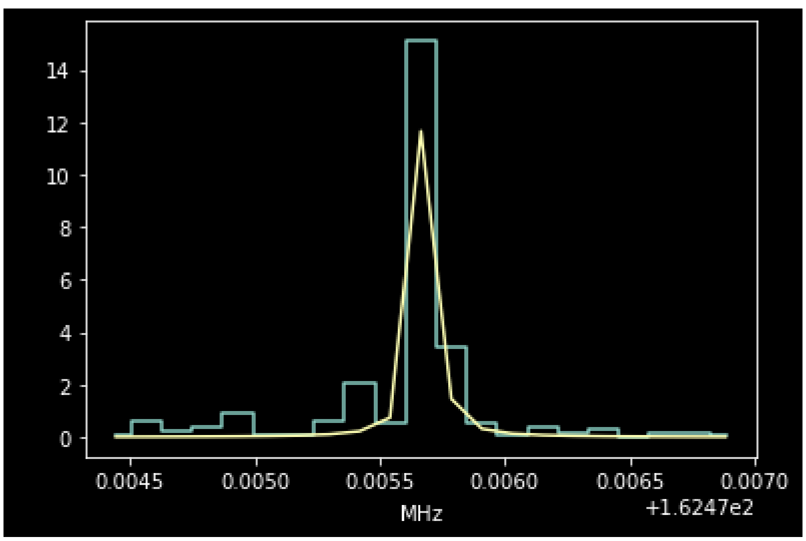
In [37]:
```
# now try fitting a Lorentzian profile (Lorentz1D)
```

In [39]:
```
from astropy.modeling.models import Lorentz1D

guesses = [np.max(selection2), moment1, sigma]

lmfitter = LevMarLSQFitter()
lorenz_fit = lmfitter(Lorentz1D(*guesses), frequency_axis,
                      line_profile)
lorenz_fit
```

Out[39]:
```
<Lorentz1D(amplitude=17.74776501, x_0=162.47568412 MHz, fwhm=0.00006002 MHz)>
```

In [40]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
pl.plot(frequency_axis, lorenz_fit(frequency_axis))
```

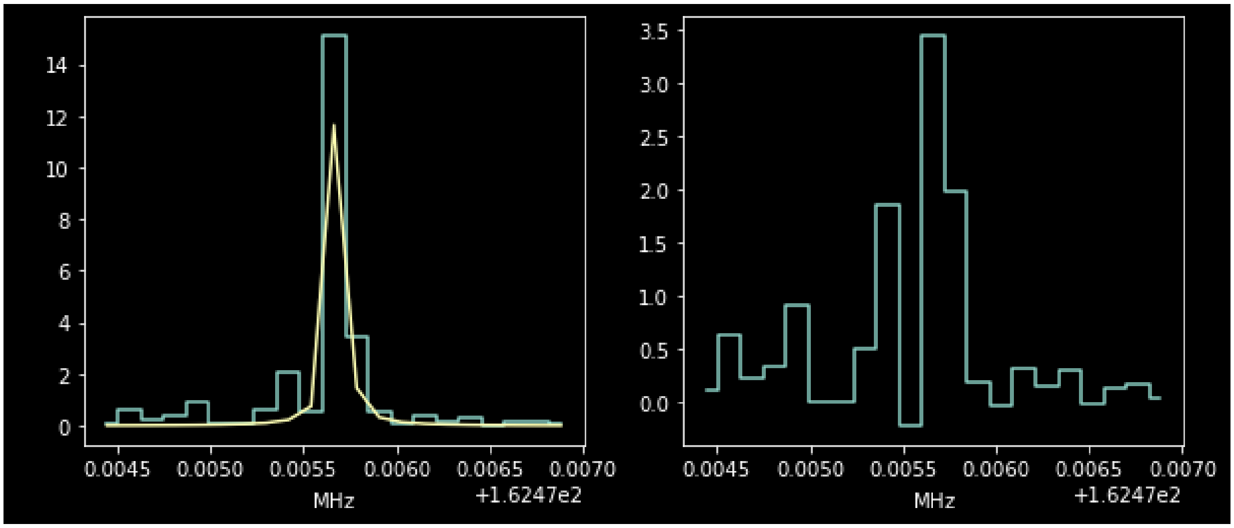
Out[40]:
```
[<matplotlib.lines.Line2D at 0x210abb2e340>]
```



In [41]:
```
lorenz_model = lorenz_fit(frequency_axis)
lor_residual = line_profile - lorenz_model
```

In [42]:
```
pl.figure(figsize=(10,4))
pl.subplot(1,2,1).plot(frequency_axis, line_profile,
            drawstyle='steps-mid')
pl.plot(frequency_axis, lorenz_fit(frequency_axis))

pl.subplot(1,2,2).plot(frequency_axis, lor_residual,
            drawstyle='steps-mid')
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x210abbcf0d0>]
```

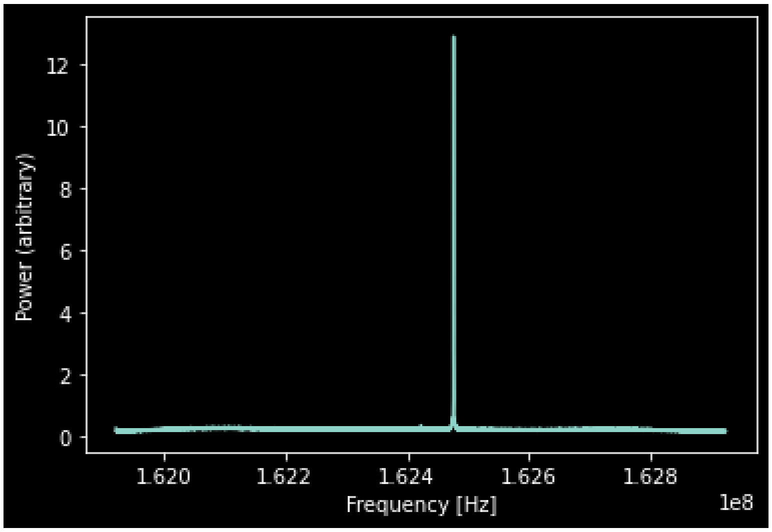


Which fits look best? - gaussian

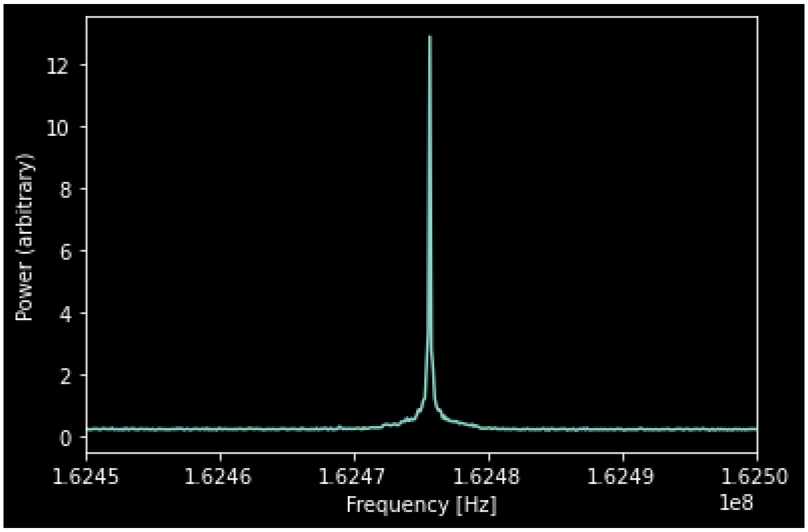
What fitted or measured parameters are "good"? Are there some you are uncomfortable with? - i think the amplitude is okayish, the fwhm looks good, sigma is okay

In [43]:
```
tbl = Table.read('data/calibration_spectrum_250integrations.fits')
```

In [44]:
```
pl.plot(tbl['freq'].quantity, tbl['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```

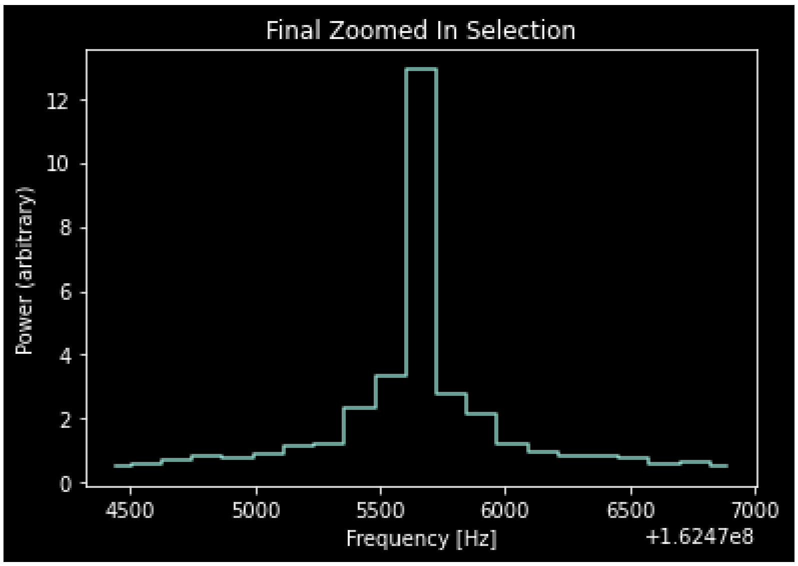


In [45]:
```
pl.plot(tbl['freq'].quantity, tbl['power']);
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
pl.xlim(162.45*u.MHz, 162.5*u.MHz);
```



In [46]:
```
## changing into a numpy array to manipulate
freq = tbl['freq'].quantity
freq = np.array(freq)
```

In [48]:
```
## final selection
selection2 = freq[4524:4545]
power_sel2 = tbl['power'][4524:4545]

pl.plot(selection2, power_sel2, drawstyle='steps-mid')
pl.title("Final Zoomed In Selection")
pl.xlabel("Frequency [Hz]")
pl.ylabel("Power (arbitrary)");
```



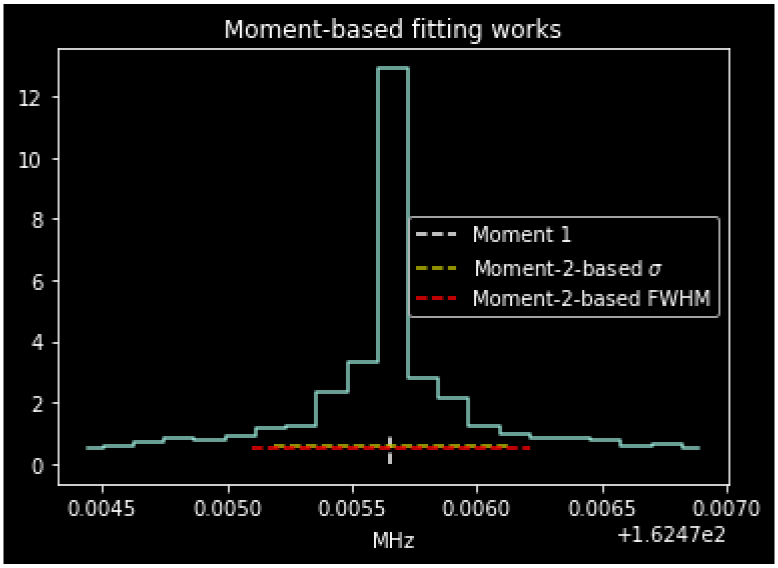
In [49]:
```
frequency_axis = selection2*u.Hz
frequency_axis = frequency_axis.to(u.MHz)
line_profile = power_sel2
```

In [50]:
```
# example moment0 / integrated intensity measurement we might make for an HI line
delta_nu = frequency_axis[1] - frequency_axis[0]
```

In [51]:
```
# moments on the downselected data
moment0 = (line_profile * delta_nu).sum()
moment1 = (frequency_axis * line_profile * delta_nu).sum() / moment0
moment2 = ( (frequency_axis - moment1)**2 * line_profile * delta_nu).sum() / mome
sigma = moment2**0.5
moment0, moment1, moment2, sigma
```

Out[51]:
```
(<Quantity 0.00448292 MHz>,
 <Quantity 162.47565499 MHz>,
 <Quantity 2.2239929e-07 MHz2>,
 <Quantity 0.00047159 MHz>)
```

In [52]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
#pl.xlim(1420.36,1420.45)
pl.vlines(moment1, 0*u.Jy, 1*u.Jy, color='w', linestyle='--', label='Moment 1')
# the Gaussian width is the half-width at exp(-1/2)
pl.hlines( np.exp(-0.5), moment1-sigma, moment1+sigma, color='y', linestyle='--',
pl.hlines( 0.5, moment1-sigma*2.35/2, moment1+sigma*2.35/2, color='r', linestyle=
pl.legend(loc='best');
pl.title("Moment-based fitting works");
```

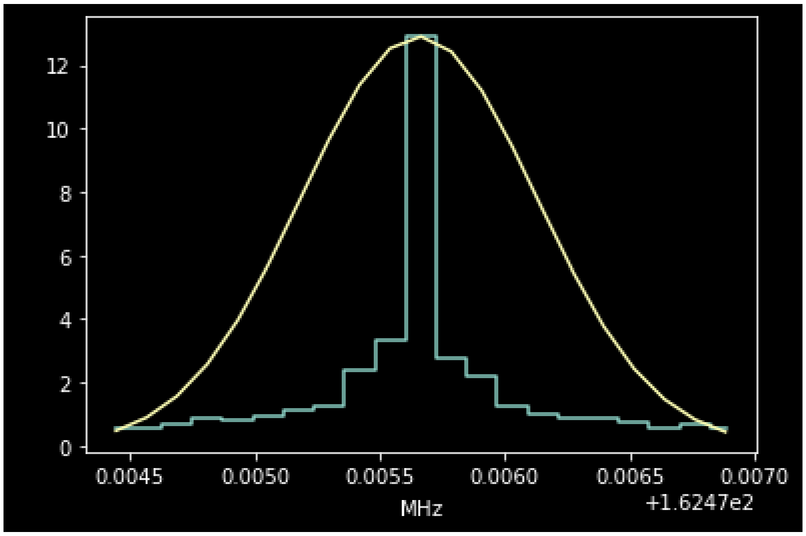


In [53]:
```
guess_gaussian = Gaussian1D(amplitude=line_profile.max(),
                            mean=moment1,
                            stddev=sigma
                           )
guess_gaussian
```

Out[53]:
```
<Gaussian1D(amplitude=12.90504768, mean=162.47565499 MHz, stddev=0.00047159 MH
z)>
```

In [54]:
```
pl.plot(frequency_axis,line_profile,drawstyle='steps-mid')
pl.plot(frequency_axis, guess_gaussian(frequency_axis))
```

Out[54]: [<matplotlib.lines.Line2D at 0x210aceb0f40>]

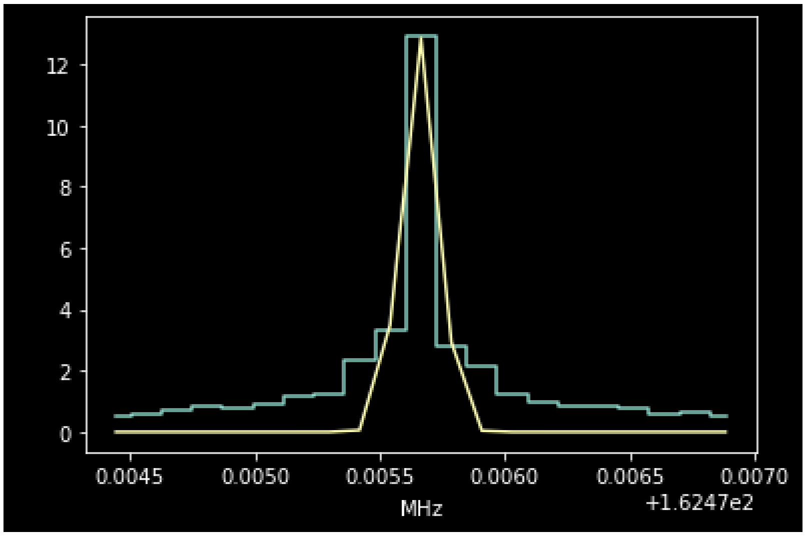


In [55]:
```
fitter = LevMarLSQFitter()
fitted_gaussian = fitter(guess_gaussian, frequency_axis, line_profile)
fitted_gaussian
```

Out[55]: <Gaussian1D(amplitude=12.86895646, mean=162.4756586 MHz, stddev=0.0000732 MHz)>

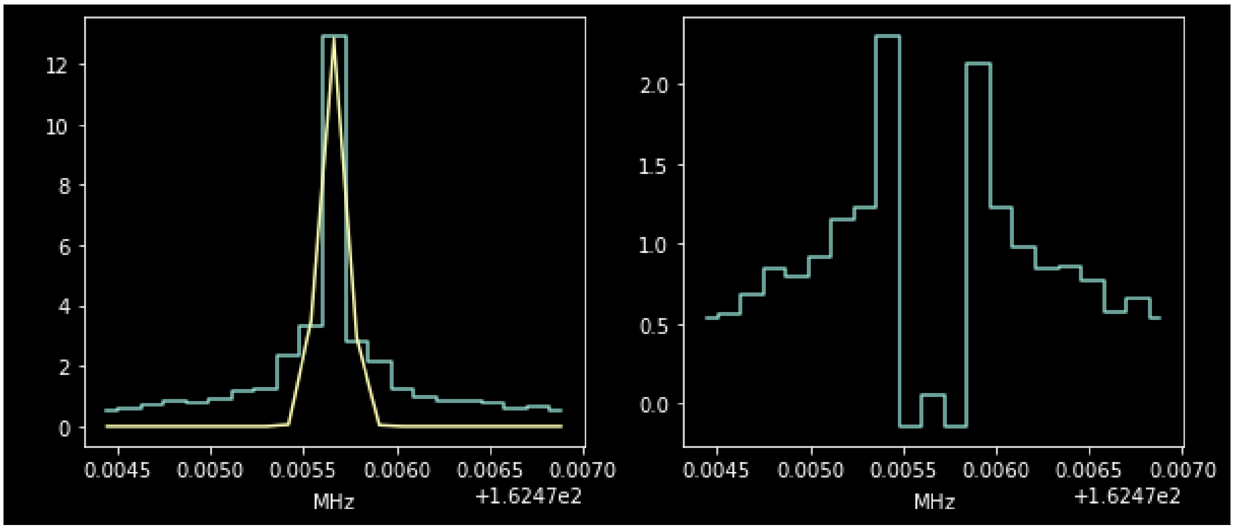
In [56]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
pl.plot(frequency_axis, fitted_gaussian(frequency_axis))
```

Out[56]: [<matplotlib.lines.Line2D at 0x210acf34970>]



In [57]:
```
model = fitted_gaussian(frequency_axis)
residual = line_profile - model

pl.figure(figsize=(10,4))
pl.subplot(1,2,1).plot(frequency_axis, line_profile,
            drawstyle='steps-mid')
pl.plot(frequency_axis, fitted_gaussian(frequency_axis))

pl.subplot(1,2,2).plot(frequency_axis, residual,
            drawstyle='steps-mid')
```

Out[57]:
```
[<matplotlib.lines.Line2D at 0x210acfd4be0>]
```



In [58]:
```
print(fitter.fit_info['param_cov'])

print(f"Peak is {fitted_gaussian.amplitude.value} +/-"
      f" {fitter.fit_info['param_cov'][0,0]**0.5}")

print(f"Centroid is {fitted_gaussian.mean.value} +/-"
      f" {fitter.fit_info['param_cov'][1,1]**0.5}")

print(f"Width is {fitted_gaussian.stddev.value} +/-"
      f" {fitter.fit_info['param_cov'][2,2]**0.5}")
```

```
[[ 1.20394938e+00 -7.67493949e-07 -2.54658228e-06]
 [-7.67493949e-07  1.11800207e-10  6.93862920e-12]
 [-2.54658228e-06  6.93862920e-12  4.60404865e-11]]
Peak is 12.868956456991208 +/- 1.097246269913442
Centroid is 162.47565859862135 +/- 1.0573561681128077e-05
Width is 7.319673934012117e-05 +/- 6.7853140315086296e-06
```

In [59]:
```
print(f"Centroid is {fitted_gaussian.mean.value:0.5f} +/- {fitter.fit_info['param
print(f"Width is {fitted_gaussian.stddev.value:0.5f} +/- {fitter.fit_info['param_
print()
print(f"m1 = {moment1:0.5f}")
print(f"sqrt(m2) = {moment2**0.5:0.5f}")
```

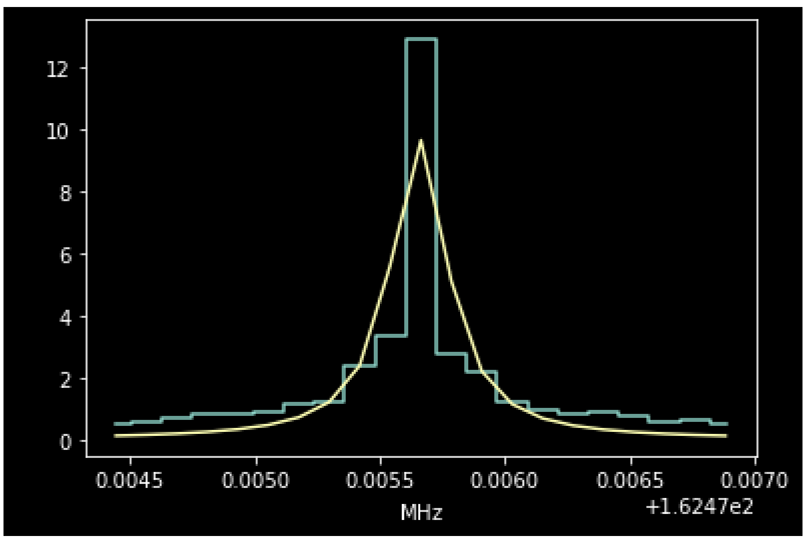
```
Centroid is 162.47566 +/- 0.00001
Width is 0.00007 +/- 0.00001

m1 = 162.47565 MHz
sqrt(m2) = 0.00047 MHz
```

In [60]:
```
guesses = [np.max(selection2), moment1, sigma]

lmfitter = LevMarLSQFitter()
lorenz_fit = lmfitter(Lorentz1D(*guesses), frequency_axis,
                      line_profile)
lorenz_fit
```

Out[60]:
```
<Lorentz1D(amplitude=9.67326523, x_0=162.47565626 MHz, fwhm=0.00027143 MHz)>
```

In [61]:
```
pl.plot(frequency_axis, line_profile, drawstyle='steps-mid')
pl.plot(frequency_axis, lorenz_fit(frequency_axis))
```

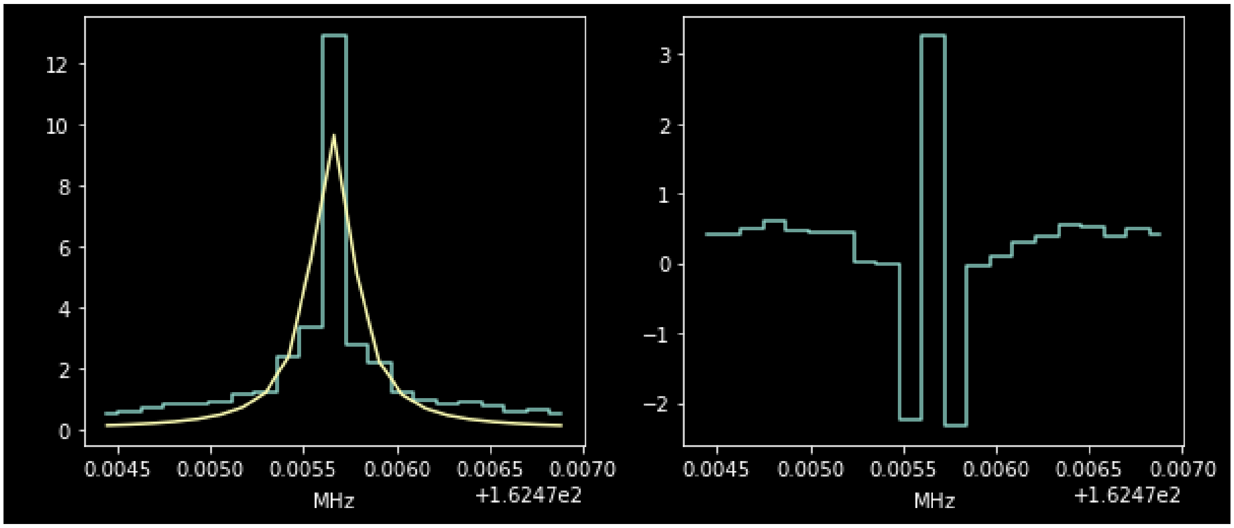
Out[61]:
```
[<matplotlib.lines.Line2D at 0x210ad056eb0>]
```



In [62]:
```
lorenz_model = lorenz_fit(frequency_axis)
lor_residual = line_profile - lorenz_model
```

In [63]:
```
pl.figure(figsize=(10,4))
pl.subplot(1,2,1).plot(frequency_axis, line_profile,
            drawstyle='steps-mid')
pl.plot(frequency_axis, lorenz_fit(frequency_axis))

pl.subplot(1,2,2).plot(frequency_axis, lor_residual,
            drawstyle='steps-mid')
```

Out[63]: [<matplotlib.lines.Line2D at 0x210ad1082b0>]



I believe my gaussian fit looks better

The amplitude needs work, the fwhm is a little wide, sigma seems to be okay

conclusions - the gaussian seems to fit both spectra better and give better parameters overall