In [64]:
```
import numpy as np
import matplotlib.pyplot as plt

import astropy.table as table
import astropy.io.ascii as ascii
from astropy.nddata import CCDData
from astropy.nddata import Cutout2D
from astropy.coordinates import SkyCoord
import astropy.units as u

import os

import scipy.interpolate
from scipy.integrate import solve_ivp
from scipy.stats import linregress
from scipy.optimize import curve_fit
```

ASTR 310 Fall 2024 --- Final Project

Instructions

- The final project may not be submitted late, but you may upload draft files and

replace them with a final version at any time up until the deadline. It's not as long as it looks, but don't wait until the last minute to start!

- As this project stands in lieu of a final exam, the projects must be your own work and must not be discussed with your peers

prior to submission.

- We will answer clarifying questions, but we will not offer as much assistance as we do with homework assignments. This is a summative

assignment meant to show what you can do on your own after taking the course.

- You are encouraged to consult the lecture and reading slides as well as Python, NumPy/SciPy, and Astropy documentation that you can find online. You can also ask us for clarification on the materials in the lecture and reading slides. Do not ask any other entities (peers, online forums, other faculty, AI or LLMs) to help or do the work for you; violations will be treated as academic integrity infractions.
- Experience suggests that your machine's latex typesetting may look different in this notebook than it does on the PDF document. In case of discrepancies, the PDF version is definitive. Best to do a quick comparison to see whether they are the same.
- If you get stuck on any one part of this project, you may still be able to move forward and complete other parts of the project.

0. Don't panic!

It looks long because I have written out the steps in great detail, but the individual steps are not large.

1. Sampling and the Fourier Transform

This exercise is a more detailed examination of the power spectrum analysis we did for Cepheid variable stars in HW 9.

a.) Initial power spectrum

- Read the data file for one of the stars, X Cygni. Your initial analysis should have showed that it has a pulsation period of 16.4 days.
- Redo the first part of the power spectrum analysis just as for the homework. Select visual band measurements with JD > 2440000, interpolate to a sampling interval of 1 day, compute the power spectrum, and plot it. You don't have to remove the baseline slope. Remember all the appropriate plot labels and so on.
- Zoom in and make another plot showing the power spectrum over a very restricted frequency range: cover a total range of 0.04 cycles/day around the fundamental frequency (the highest peak).

You should see that the "peak" is not a simple spike but is in fact a series of 5 spikes, with the center one being the tallest.

It would be good form to create a function or a class and some methods that compute and plot the power spectrum, since you're going to do that several times.

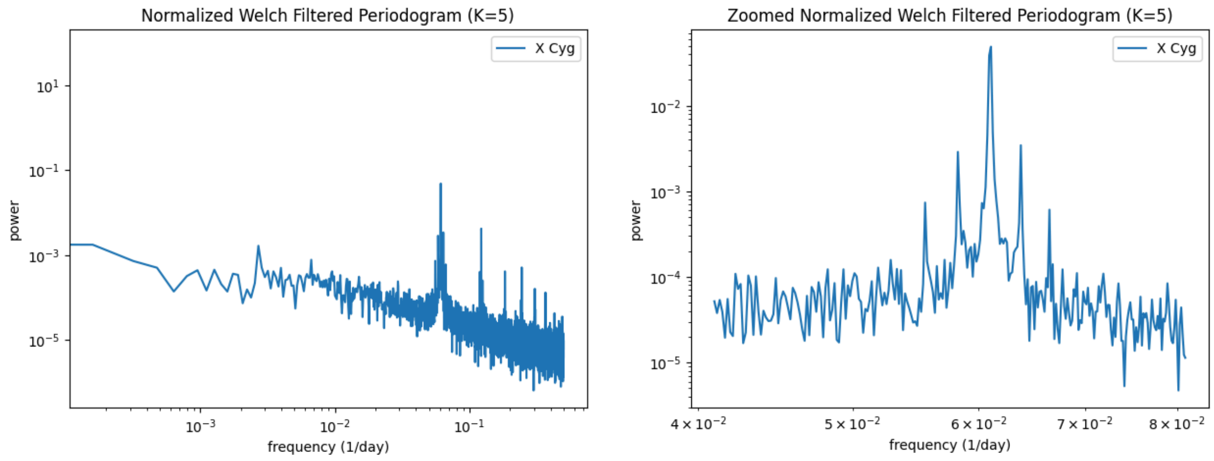
[6 pts]

In [65]:
```python
# read data
t = ascii.read(f"../Homework/HW9/aavso_cepheids/aavso_xcyg.dat")

# select relevant columns
t = t[["JD", "Magnitude"]]

# filter measurements
t = t[t["JD"] > 2440000]

# interpolate with sampling interval of 1 day
interp = scipy.interpolate.interp1d(t["JD"], t["Magnitude"], kind='linear')
t_interp = table.Table()
t_interp["interp_JD"] = np.arange(min(t["JD"]), max(t["JD"]), 1)
t_interp["interp_mag"] = interp(t_interp["interp_JD"])

# compute welch-filtered periodogram
def compute_welch_filtered_periodogram(x_in, y_in):
    N = len(x_in)
    K = 5
    N_seg = 2*N // (K + 1)
    offset = N_seg // 2
    dx = 1

    def my_welch(N_seg):
        welch = []
        for n in range(0, N_seg):
            wn = 1 - ((n - N_seg/2)/(N_seg/2))**2
            welch.append(wn)
        return np.array(welch)


    def normalize(P, W, N_seg):
        return N_seg / np.sum(W**2) * P

    segs_y = []
    segs_x = []

    for i in range(K):
        y = y_in[offset*i:(offset*i)+N_seg]
        segs_y.append(y)
        x = x_in[offset*i:(offset*i)+N_seg]
        segs_x.append(y*my_welch(N_seg))

    spectra = []

    for i in range(K):
        fhat = np.fft.rfft(segs_y[i])
        k = np.fft.rfftfreq(N_seg, dx)

        P = np.abs(fhat)**2 / N_seg**2
        P[1:-1] = 2*P[1:-1]
        spectra.append(P)

    # normalize
    spectra = [normalize(s, my_welch(N_seg), N_seg) for s in spectra]

    # combine
    mean_P = np.mean(spectra, axis=0)
    k = np.fft.rfftfreq(N_seg, dx)

    return k, mean_P

k, P = compute_welch_filtered_periodogram(x_in=t_interp["interp_JD"], y_in=t
```

In [66]:
```
# plot the periodigram
fig, ax = plt.subplots(1,2,figsize=(15,5))

ax[0].loglog(k, P, label="X Cyg")
ax[0].set_xlabel("frequency (1/day)")
ax[0].set_ylabel("power")
ax[0].set_title("Normalized Welch Filtered Periodogram (K=5)")
ax[0].legend()

# calculate the zoomed interval
fund_freq = k[P[1::].argmax()]
```

```
k_min = fund_freq - 0.02
k_max = fund_freq + 0.02

k_zoom = k[(k > k_min) & (k < k_max)]
P_zoom = P[(k > k_min) & (k < k_max)]

# plot the zoomed interval
ax[1].loglog(k_zoom, P_zoom, label="X Cyg")
ax[1].set_xlabel("frequency (1/day)")
ax[1].set_ylabel("power")
ax[1].set_title("Zoomed Normalized Welch Filtered Periodogram (K=5)")
ax[1].legend()
```

Out[66]:
```
<matplotlib.legend.Legend at 0x3202e5c40>
```



Why does the fundamental frequency show up as a series of 5 closely-spaced spikes? The effect is due to the Sun's motion through the celestial sphere. It's very hard to get good optical photometry for objects that are close to the Sun (above the horizon during daytime). Thus, there are certain parts of the year for which we have very little data for a given star. This sampling effect in the timeseries data shows up as a convolution in frequency space.

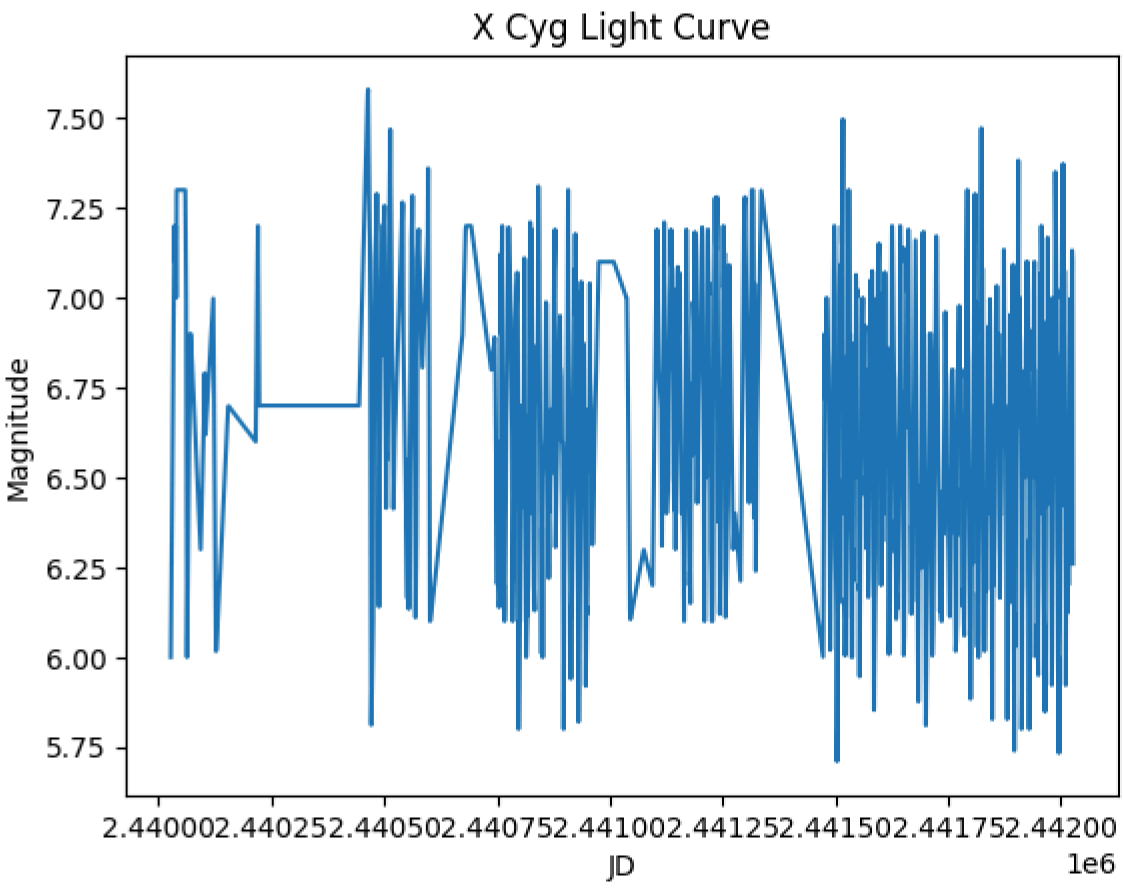
b.) Visualize the sampling effect

To see the sampling effect, it's easiest to plot a subset of the timeseries data. Thus, go back to either the raw data (after the date and band selection steps) or your interpolated version. Plot the light curve, but only for a range of about 2000 consecutive days. It doesn't matter much which days they are as this sampling effect occurs throughout the whole sequence. You should clearly see that there are clumps with lots of data and relatively sparse gaps with little data. The pattern repeats on a yearly cycle. Be sure to label all the axes appropriately.

[6 pts]

In [67]:
```
plt.plot(t_interp["interp_JD"][0:2000], t_interp["interp_mag"][0:2000])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("X Cyg Light Curve")
```

```
Out[67]:  Text(0.5, 1.0, 'X Cyg Light Curve')
```



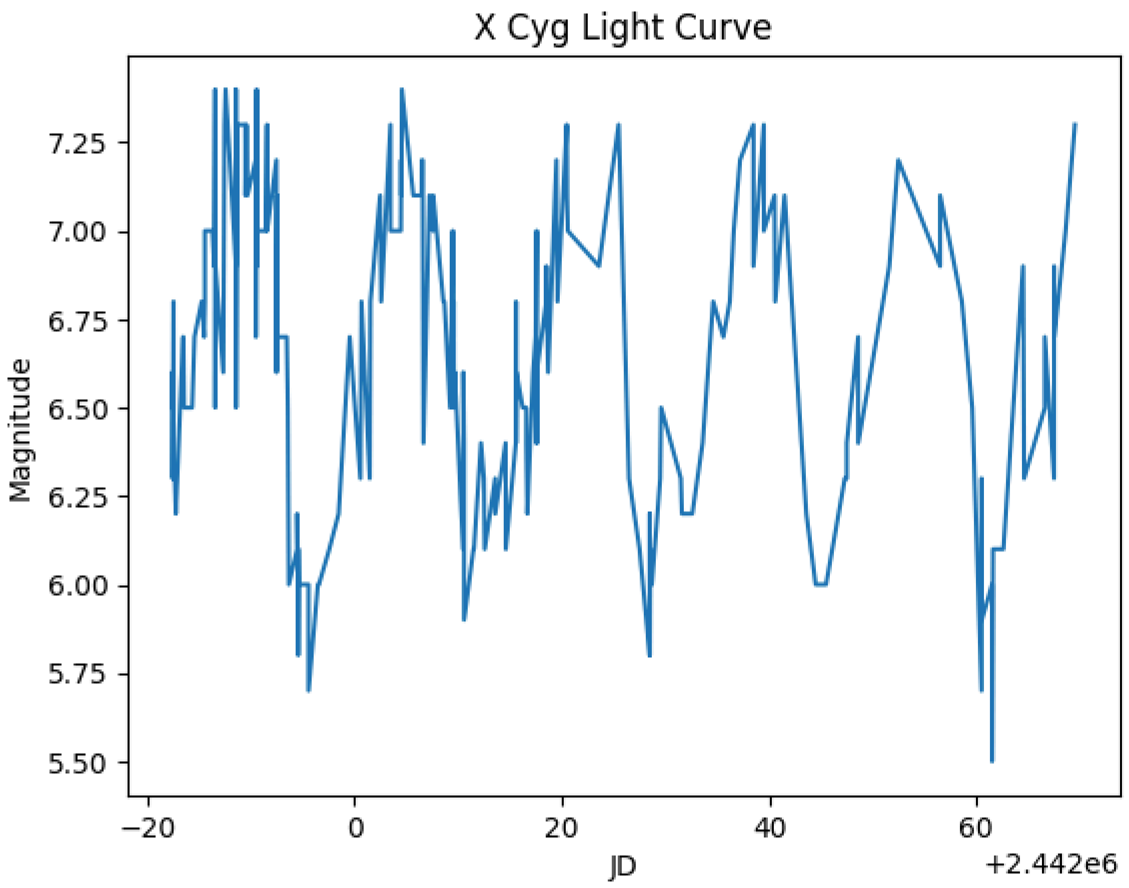
c.) More visualizations

While you're at it, just for fun, make another plot of the timeseries data in which you have zoomed in enough to be able to see something like 5 or 10 of the 16-day oscillations.

[4 pts]

```
In [68]:  plt.plot(t["JD"][1400:1600], t["Magnitude"][1400:1600])
          plt.xlabel("JD")
          plt.ylabel("Magnitude")
          plt.title("X Cyg Light Curve")
```

```
Out[68]:  Text(0.5, 1.0, 'X Cyg Light Curve')
```



d.) Generate a test signal

Now we're going to mimic the yearly sampling pattern to show its effect on the power spectrum. Begin by generating a pure sinusoidal oscillation signal. Let your test signal have the same period as X Cygni and be sampled once per day, just like our interpolated data stream, and let it have the same length as the interpolated data stream. The amplitude of the test signal doesn't matter.

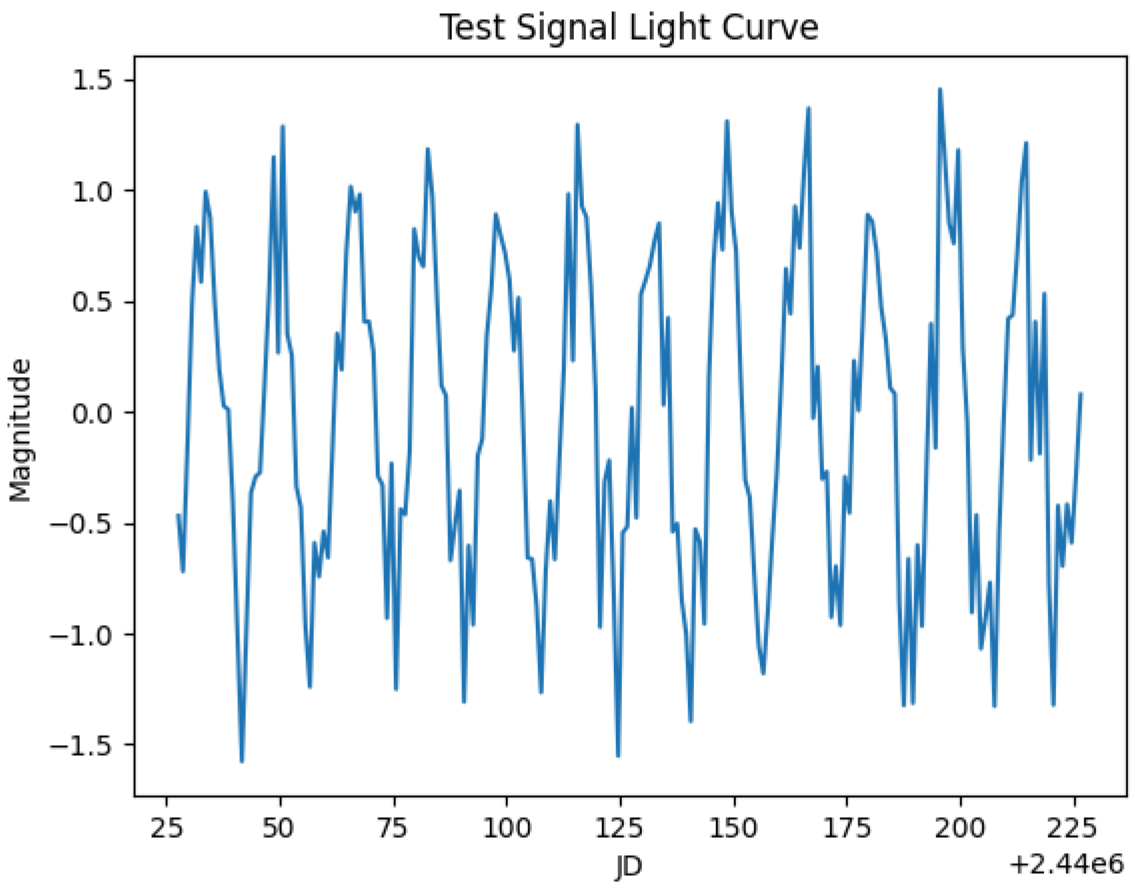
Add some random noise to your test signal. An appropriate noise distribution would be a Gaussian (normal) with a dispersion something like 30% of the test signal amplitude. (Feel free to experiment with the dispersion and see what happens.)

As a sanity check, plot your test signal, zoomed in enough to see the 16-day oscillations.

[6 pts]

In [69]:
```
test_signal = np.sin(2*np.pi*fund_freq*t_interp["interp_JD"]) + np.random.nc

plt.plot(t_interp["interp_JD"][0:200], test_signal[0:200])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("Test Signal Light Curve")
```

Out[69]:
```
Text(0.5, 1.0, 'Test Signal Light Curve')
```



e.) Power spectrum of test signal (before sampling)

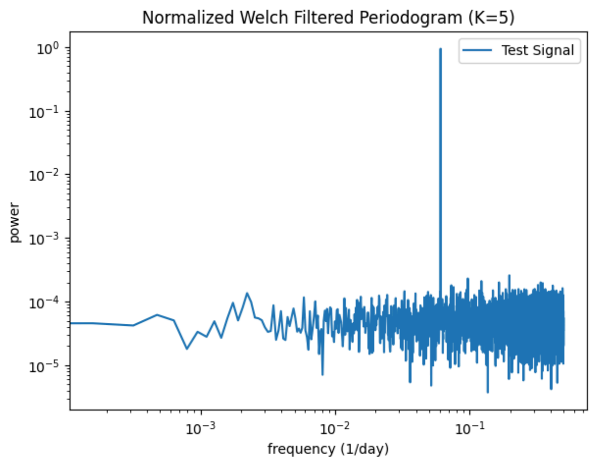
Put this test signal through your power spectrum machinery as before; plot the full power spectrum and a zoom-in version covering a range of 0.04 cycles/day around the fundamental frequency. You should see just a single spike in the power spectrum.

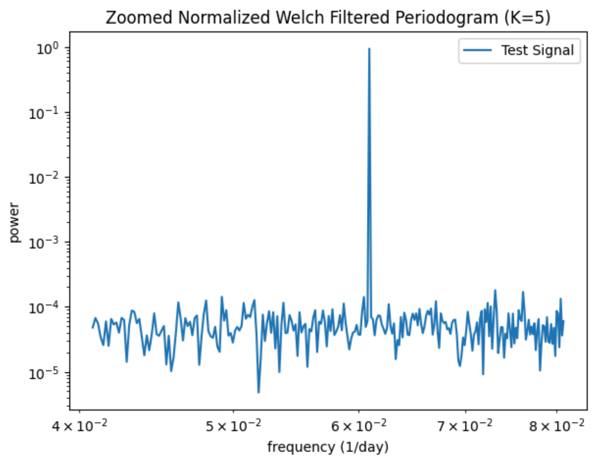
[6 pts]

In [70]:
```python
k_test, P_test = compute_welch_filtered_periodogram(t_interp["interp_JD"], t

# calculate the zoomed interval
fund_freq_test = k_test[P_test[1::].argmax()]
k_min_test = fund_freq_test - 0.02
k_max_test = fund_freq_test + 0.02

k_zoom_test = k_test[(k_test > k_min_test) & (k_test < k_max_test)]
P_zoom_test = P_test[(k_test > k_min_test) & (k_test < k_max_test)]

# plot the periodigram
fig, ax = plt.subplots(1, 2, figsize=(15,5))

ax[0].loglog(k_test, P_test, label="Test Signal")
ax[0].set_xlabel("frequency (1/day)")
ax[0].set_ylabel("power")
ax[0].set_title("Normalized Welch Filtered Periodogram (K=5)")
ax[0].legend()
```

```
ax[1].loglog(k_zoom_test, P_zoom_test, label="Test Signal")
ax[1].set_xlabel("frequency (1/day)")
ax[1].set_ylabel("power")
ax[1].set_title("Zoomed Normalized Welch Filtered Periodogram (K=5)")
ax[1].legend()
```

Out[70]:
```
<matplotlib.legend.Legend at 0x32186c680>
```





f.) Yearly sampling pattern

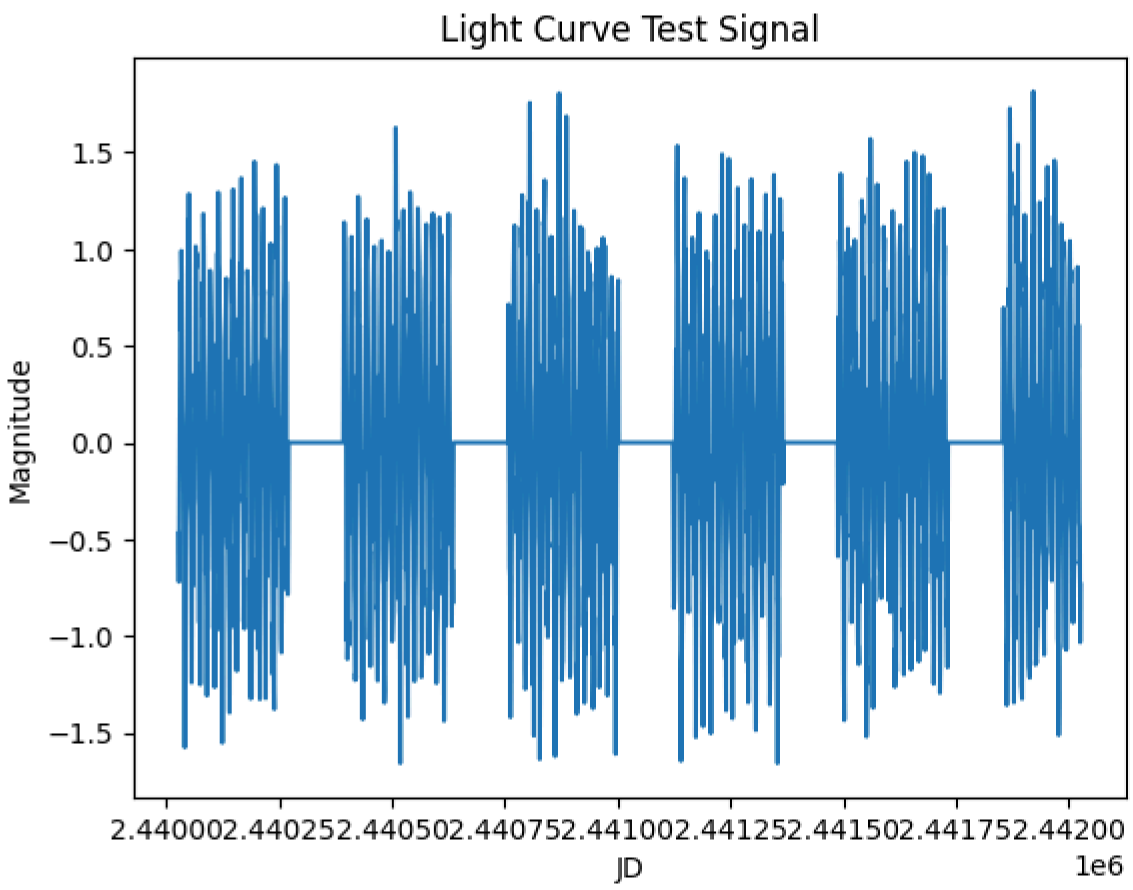
Consider your vector of dates, sampled once per day. For each day, compute what fraction of a year it represents. For example, if day 0 is the start of a year, day 182.625 is half way through the year. Day 547.875 is also half way through a year. Leave your test signal untouched for two-thirds of a year, but set your test signal to 0 for the remaining 1/3 of a year. This process mimics the situation in which you can only observe the source for 8 months out of the year.

Plot your new time series, for a range of about 2000 consecutive days as before. You should clearly see the gaps in which you zeroed out the test signal.

[6 pts]

In [71]:
```
# calculate year fractions
year_frac = (np.array(t_interp["interp_JD"]) - t_interp["interp_JD"][0]) % 3
# set last 1/3rd of the year to 0
test_signal_sampled = test_signal
test_signal_sampled[year_frac > (2/3)] = 0

plt.plot(t_interp["interp_JD"][0:2000], test_signal_sampled[0:2000])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("Light Curve Test Signal")
```

Out[71]:
```
Text(0.5, 1.0, 'Light Curve Test Signal')
```



g.) Power spectrum of test signal (after sampling)

Put your sampled test signal through your power spectrum machinery as before and plot its power spectrum (zoomed in around the fundamental). This time you should see that the central peak is surrounded by some weaker sub-peaks, just as in the case of the real data. There will be more sub-peaks than in the real data; the actual sampling is not as regular or as sharp as our simple model.

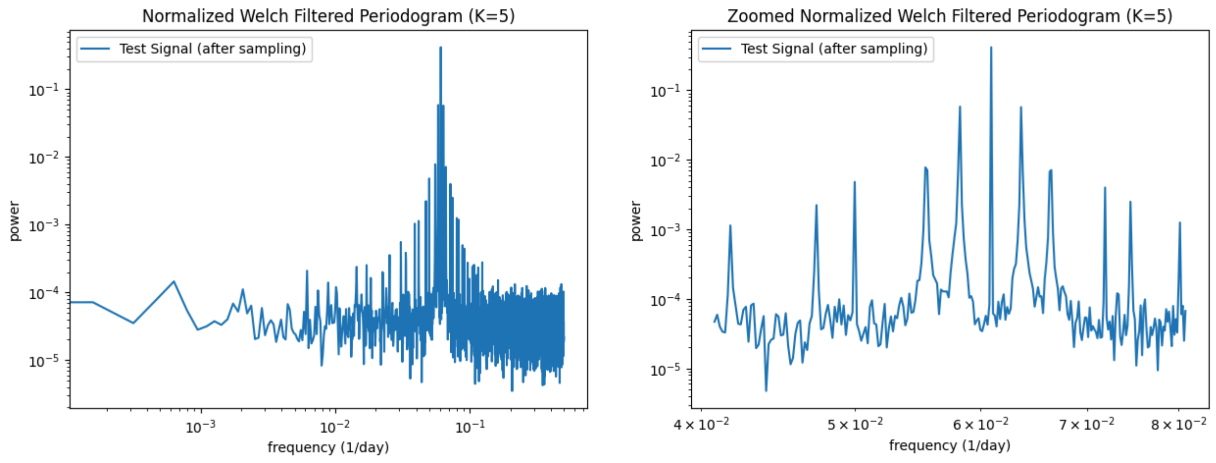
Additional context. The yearly cycle, with good data for some months and then no data for a few months, mimics a beat pattern between two sinusoids of similar frequencies. The beat phenomenon may be familiar to you from tuning musical instruments. Thus, the real power spectrum has some closely spaced spikes. If we call ν_0 the fundamental frequency, the offset sub-peaks will be at $\nu_0 + n\Delta\nu$ where $(\Delta\nu)^{-1}$ = 365.25 days.

[6 pts]

In [72]:
```
k_test, P_test = compute_welch_filtered_periodogram(t_interp["interp_JD"], t

# calculate the zoomed interval
fund_freq_test = k_test[P_test[1::].argmax()]
k_min_test = fund_freq_test - 0.02
k_max_test = fund_freq_test + 0.02

k_zoom_test = k_test[(k_test > k_min_test) & (k_test < k_max_test)]
P_zoom_test = P_test[(k_test > k_min_test) & (k_test < k_max_test)]
```

```
# plot the periodigram
fig, ax = plt.subplots(1, 2, figsize=(15,5))

ax[0].loglog(k_test, P_test, label="Test Signal (after sampling)")
ax[0].set_xlabel("frequency (1/day)")
ax[0].set_ylabel("power")
ax[0].set_title("Normalized Welch Filtered Periodogram (K=5)")
ax[0].legend()

ax[1].loglog(k_zoom_test, P_zoom_test, label="Test Signal (after sampling)")
ax[1].set_xlabel("frequency (1/day)")
ax[1].set_ylabel("power")
ax[1].set_title("Zoomed Normalized Welch Filtered Periodogram (K=5)")
ax[1].legend()
```

```
Out[72]:  <matplotlib.legend.Legend at 0x3216a2ed0>
```



2. Numerical integration

You may recall that in Lecture 26 we worked through the numerical solution of the structure of the isothermal sphere -- a self-gravitating spherical body supported in hydrostatic equilibrium by ideal gas pressure with a constant temperature.

The analogous situation in a plane-parallel geometry is a decent approximation for the vertical structure of galactic disks. Here we'll work through a numerical solution of that simple model. As for the isothermal sphere, we use hydrostatic equilibrium, a mass continuity equation, an ideal gas equation of state, and an assumption of constant temperature. In the case of a galactic disk, the analog of the "temperature" is the (square of the) velocity dispersion of the stars making up the disk.

After scaling to dimensionless variables like we did before, we end up with this differential equation:

$$\frac{d^2\rho}{dz^2} - \frac{1}{\rho}\left(\frac{d\rho}{dz}\right)^2 + 2\rho^2 = 0.$$

- Find a numerical solution for the conditions $\rho(z_0) = 1.0$ and $\left.\frac{d\rho}{dz}\right|_{z_0} = 0.0$. Integrate for $z = 0$ to 5 and plot the solution.
- Mathematically, this one is a little simpler than the sphere and it actually has a nice analytic solution: $\rho(z) = \rho_0 \,\mathrm{sech}^2(z)$.

Plot the analytic solution on top of your numerical solution to check that everything worked correctly.

- Also plot the ratio of the analytic solution to your numerical solution; the ratio makes it easier to see discrepancies at large z.

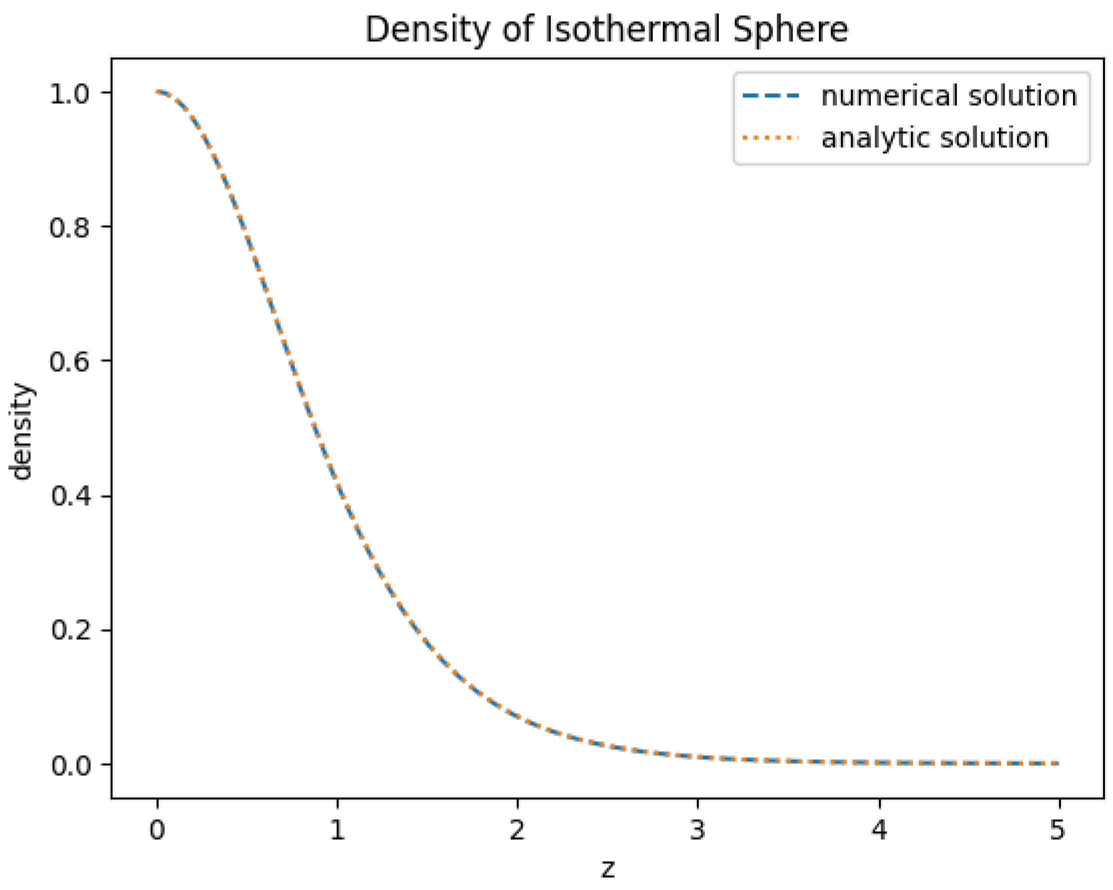
Label everything appropriately.

The point of this problem is to motivate the sech^2 fitting function that we're going to use below.

[10 pts]

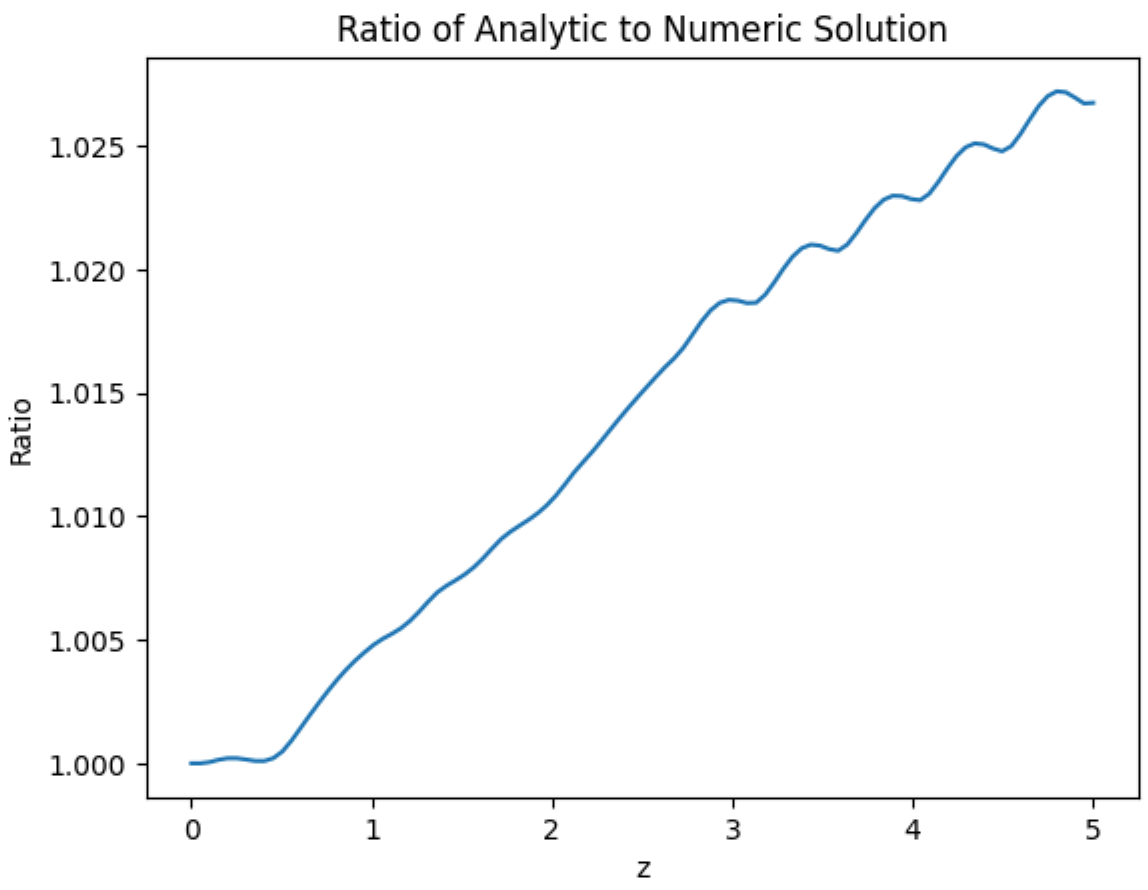
In [73]:
```
# define the equation
def equation(z, p):
    # dp1/dz = p2
    # dp2/dz = 1/p1 * p2^2 - 2*p1^2
    # p1_initial = 1
    # p2_initial = 0

    p, dp_dz = p
    dp2_dz = 1/p * dp_dz**2 - 2 * p**2

    return [dp_dz, dp2_dz]

# define the bounds and initial values
z_range = (0, 5)
t_eval = np.linspace(0,5,100)
y1_initial = 1
y2_initial = 0

# solve the problem
result = solve_ivp(equation, z_range, [y1_initial, y2_initial], t_eval=t_eva

# plot the results
plt.plot(result.t, result.y[0], label='numerical solution', ls='dashed')
# plot the analytical solution
plt.plot(t_eval, (1/np.cosh(t_eval))**2, label="analytic solution", ls='dott

plt.xlabel("z")
plt.ylabel("density")
plt.legend()
plt.title("Density of Isothermal Sphere")
```

Out[73]:
```
Text(0.5, 1.0, 'Density of Isothermal Sphere')
```



In [74]:
```
# calculate the ratios
ratio = (1/np.cosh(t_eval))**2 / result.y[0]

# plot the ratio
plt.plot(t_eval, ratio)
plt.title("Ratio of Analytic to Numeric Solution")
plt.xlabel("z")
plt.ylabel("Ratio")
```

Out[74]:
```
Text(0, 0.5, 'Ratio')
```



3. Structure of a spiral galaxy: vertical and radial profiles

a.) Read the image `NGC4565.phot.1.fits` and its corresponding mask `NGC4565.1.final_mask.fits`. These are image products of the Spitzer Survey of Stellar Structure in Galaxies (S4G; Sheth et al. 2010, Munoz-Mateos et al. 2013, and Querejeta et al. 2015.)

Obtain the pixel scale (arcseconds per pixel) from image's world coordinate system in the CD matrix. The raw values are degrees per pixel. We'll use that below. The header parameters PXSCAL1 and PXSCAL2 look attractive but they are not what we want here.

Ignore all the warnings about SIP distortions.

[6 pts]

In [75]:
```
# load the data
img = CCDData.read("NGC4565.phot.1.fits")
mask = CCDData.read("NGC4565.1.final_mask.fits")
# get the pixel scale
pix_scale = img.wcs.wcs.cd[1][1]
```

```
INFO:
                Inconsistent SIP distortion information is present in the FI
TS header and the WCS object:
                SIP coefficients were detected, but CTYPE is missing a "-SI
P" suffix.
                astropy.wcs is using the SIP distortion coefficients,
                therefore the coordinates calculated here might be incorrec
t.

                If you do not want to apply the SIP distortion coefficients,
                please remove the SIP coefficients from the FITS header or t
he
                WCS object.  As an example, if the image is already distorti
on-corrected
                (e.g., drizzled) then distortion components should not apply
and the SIP
                coefficients should be removed.

                While the SIP distortion coefficients are being applied her
e, if that was indeed the intent,
                for consistency please append "-SIP" to the CTYPE in the FIT
S header or the WCS object.

                 [astropy.wcs.wcs]
INFO:
        Inconsistent SIP distortion information is present in the current WC
S:
        SIP coefficients were detected, but CTYPE is missing "-SIP" suffix,
        therefore the current WCS is internally inconsistent.

        Because relax has been set to True, the resulting output WCS will ha
ve
        "-SIP" appended to CTYPE in order to make the header internally cons
istent.

        However, this may produce incorrect astrometry in the output WCS, if
        in fact the current WCS is already distortion-corrected.

        Therefore, if current WCS is already distortion-corrected (eg, drizz
led)
        then SIP distortion components should not apply. In that case, for a
WCS
        that is already distortion-corrected, please remove the SIP coeffici
ents
        from the header.

         [astropy.wcs.wcs]
INFO:
                Inconsistent SIP distortion information is present in the FI
TS header and the WCS object:
                SIP coefficients were detected, but CTYPE is missing a "-SI
P" suffix.
                astropy.wcs is using the SIP distortion coefficients,
                therefore the coordinates calculated here might be incorrec
t.
```

```
                If you do not want to apply the SIP distortion coefficients,
                please remove the SIP coefficients from the FITS header or t
he
                WCS object.  As an example, if the image is already distorti
on-corrected
                (e.g., drizzled) then distortion components should not apply
and the SIP
                coefficients should be removed.

                While the SIP distortion coefficients are being applied her
e, if that was indeed the intent,
                for consistency please append "-SIP" to the CTYPE in the FIT
S header or the WCS object.

                 [astropy.wcs.wcs]
INFO:
        Inconsistent SIP distortion information is present in the current WC
S:
        SIP coefficients were detected, but CTYPE is missing "-SIP" suffix,
        therefore the current WCS is internally inconsistent.

        Because relax has been set to True, the resulting output WCS will ha
ve
        "-SIP" appended to CTYPE in order to make the header internally cons
istent.

        However, this may produce incorrect astrometry in the output WCS, if
        in fact the current WCS is already distortion-corrected.

        Therefore, if current WCS is already distortion-corrected (eg, drizz
led)
        then SIP distortion components should not apply. In that case, for a
WCS
        that is already distortion-corrected, please remove the SIP coeffici
ents
        from the header.

         [astropy.wcs.wcs]
```

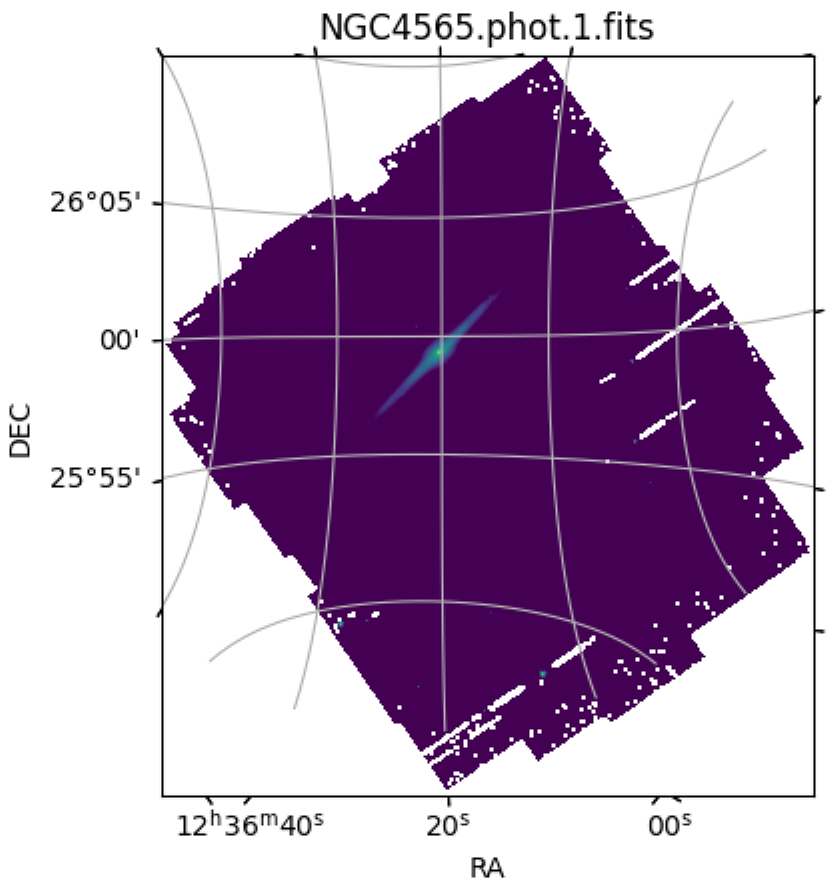
b.) Display the image so that you can see the structure of the galaxy. I found that log scaling worked well. This is an edge-on spiral galaxy. You can clearly see the galaxy's disk, which is long and thin because of the edge-on orientation. You can also see the central bulge and a nuclear point source associated with an active galactic nucleus.

Display the mask as well, with a colorbar. The mask marks pixels that are contaminated by foreground stars and background galaxies. This information will be useful to us later when we are studying the faint outer structures of NGC 4565. Notice that the mask = 0 for pixels that we can use in our analysis and the mask is > 0 for pixels that are contaminated.

[8 pts]

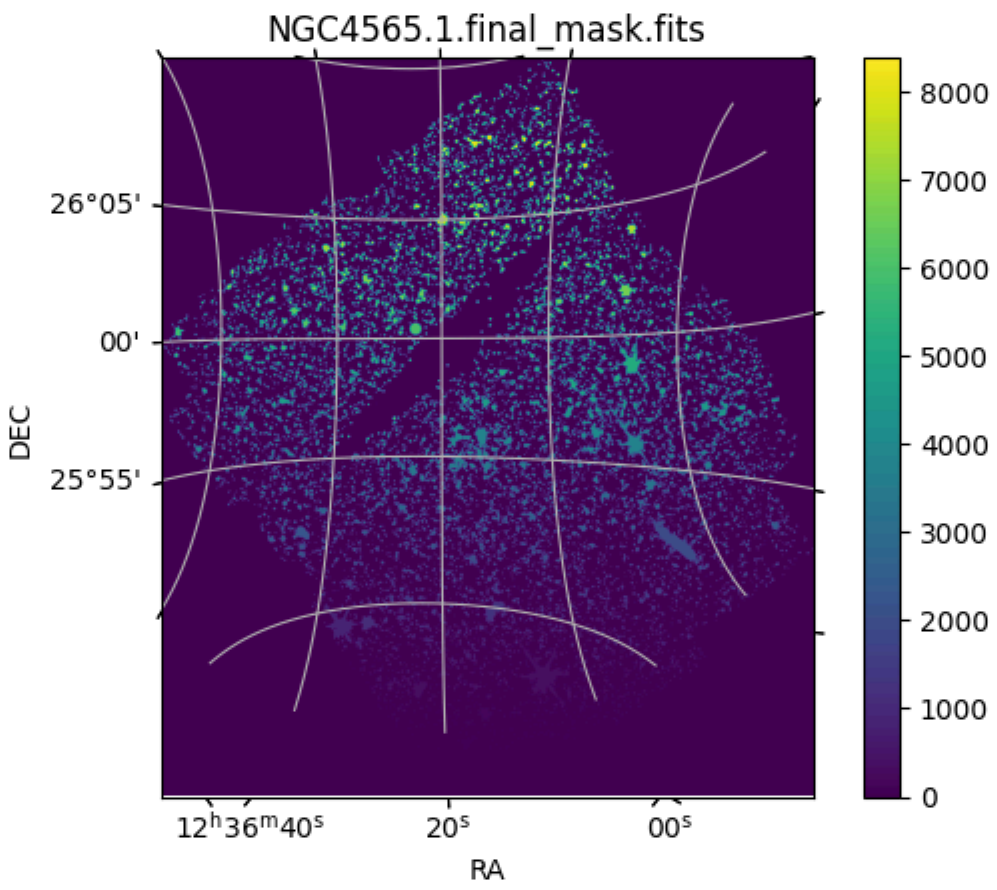
In [76]:
```python
# display the stretched image
plt.subplot(projection=img.wcs)
plt.imshow(np.log10(img), vmin=0, vmax=2)
plt.xlabel("RA")
plt.ylabel("DEC")
plt.title("NGC4565.phot.1.fits")
plt.grid()
```

```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/419315511.p
y:3: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(img), vmin=0, vmax=2)
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
```



In [77]:
```
# display the mask
plt.subplot(projection=img.wcs)
plt.imshow(mask)
plt.xlabel("RA")
plt.ylabel("DEC")
plt.title("NGC4565.1.final_mask.fits")
plt.colorbar()
plt.grid()
```

```
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
/opt/anaconda3/lib/python3.12/site-packages/astropy/wcs/wcsapi/fitswcs.py:34
6: UserWarning: 'WCS.all_world2pix' failed to converge to the requested accu
racy.
After 20 iterations, the solution is diverging at least for one input point.
  warnings.warn(str(e))
```



c.) Rotate the image (and the mask) so that the galaxy's disk is horizontal. It is straightforward to use `scipy.ndimage.rotate` to accomplish the rotation, though I found that I had to fill the NaNs with zeros first. It is OK to experiment with rotation angles unil you get a result that looks good (i.e. just eyeball estimate when the disk is horizontal).

Display the rotated images to verify that you got what you wanted.

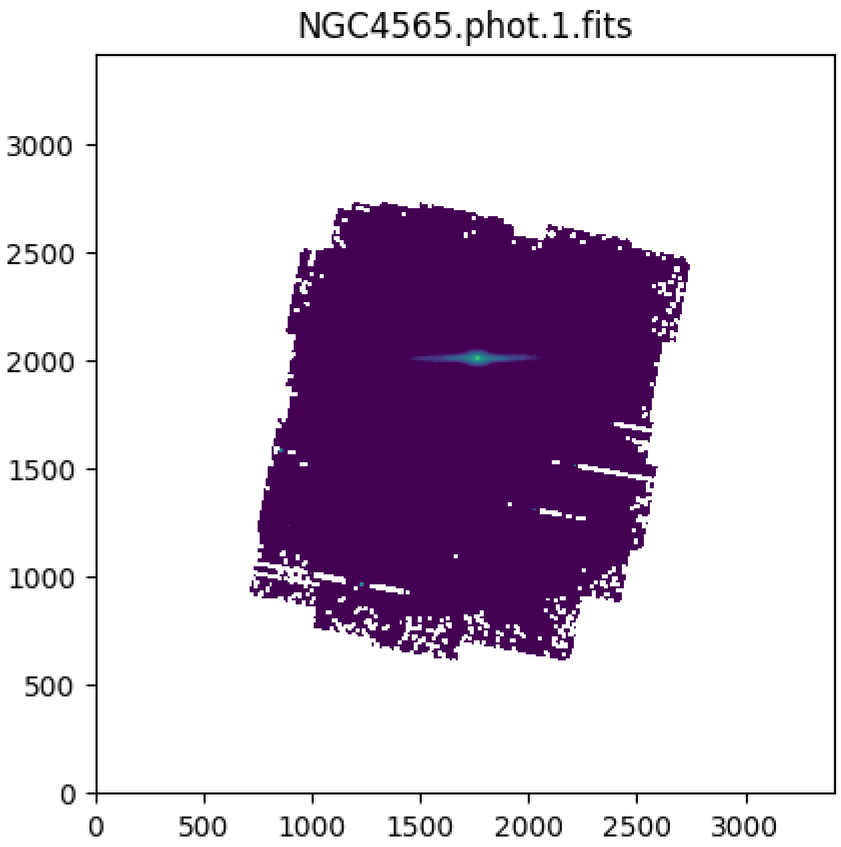
From this point forward, work with the rotated versions of the image and the mask.

[6 pts]

In [78]:
```
# rotate the image
img.data = np.nan_to_num(img.data)
img_rotated = scipy.ndimage.rotate(img, 45)
mask_rotated = scipy.ndimage.rotate(mask, 45)

# display the image
plt.imshow(np.log10(img_rotated), vmin=0, vmax=2, origin='lower')
plt.title("NGC4565.phot.1.fits")
```

```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/1175597123.p
y:7: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(img_rotated), vmin=0, vmax=2, origin='lower')
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/1175597123.p
y:7: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(img_rotated), vmin=0, vmax=2, origin='lower')
```

Out[78]:
```
Text(0.5, 1.0, 'NGC4565.phot.1.fits')
```

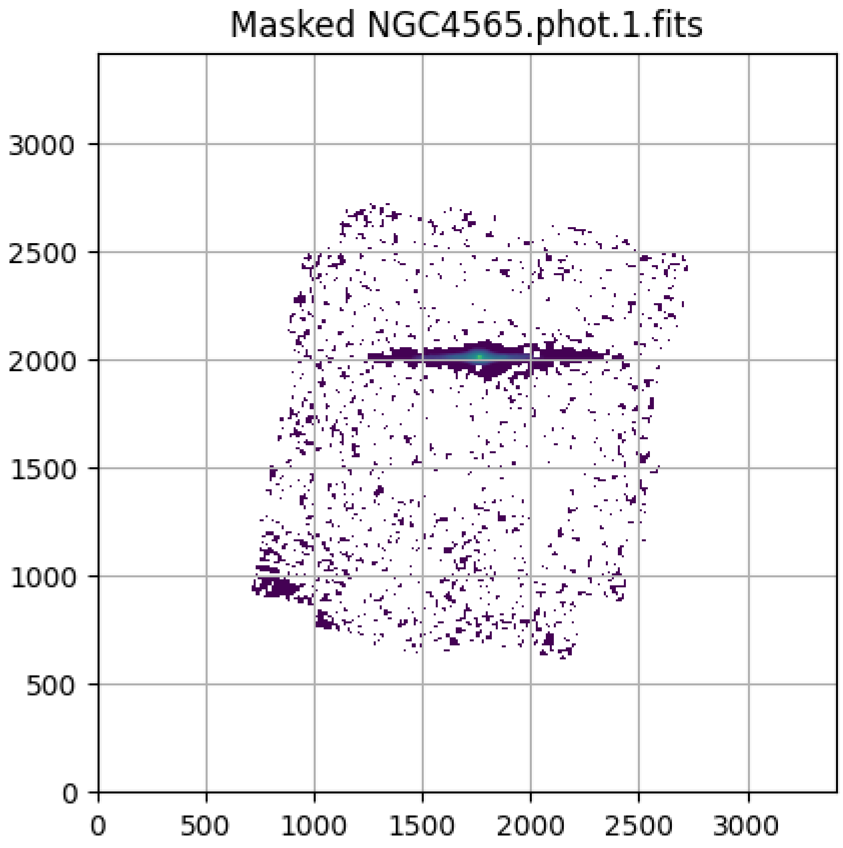


d.) Apply the mask to the image. Specifically: pixels with mask values > some threshold should have their image values set to zero or NaN. That way, when we integrate over various regions of the image, the masked pixels will not contribute to the sums. You can experiment with threshold values so that you exclude the most seriously contaminated pixels without suffering from weird interpolation edge effects. (Plot the masked image to see what's happening.)

[4 pts]

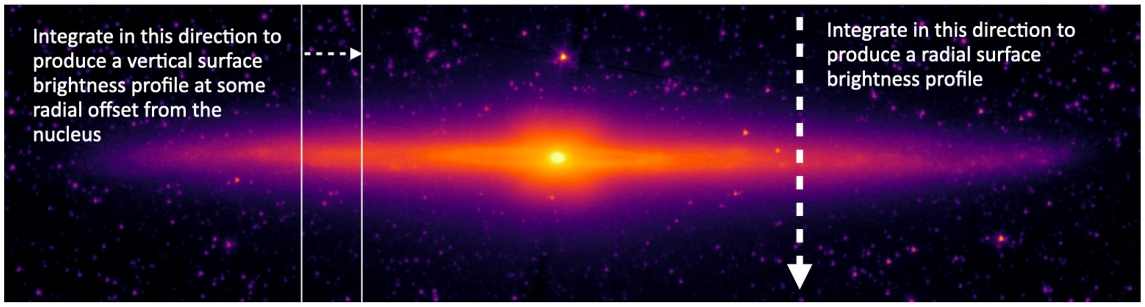
In [79]:
```python
# apply mask
img2 = img_rotated
threshold = 100
img2[mask_rotated > threshold] = 0

# display the image
plt.imshow(np.log10(img2), vmin=0, vmax=2, origin='lower')
plt.title("Masked NGC4565.phot.1.fits")
plt.grid()
```

```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/3277225886.p
y:7: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(img2), vmin=0, vmax=2, origin='lower')
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/3277225886.p
y:7: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(img2), vmin=0, vmax=2, origin='lower')
```



e.) Create the radial surface brightness profile

- Extract a subimage from your masked image. You'll want a region that covers the full width of the image but only about 400 pixels in the vertical direction, centered on the galaxy. Use that subimage for further analysis.
- Integrate the subset along the axis perpendicular to the disk, to create a 1D vector describing the integrated surface brightness of the galaxy as a function of distance along the disk. This is our radial surface brightness profile. You'll also need a 1D vector describing the distance (in pixels) along the disk.


Integrate in this direction to produce a vertical surface brightness profile at some radial offset from the nucleus
Integrate in this direction to produce a radial surface brightness profile

Note the cartoon shows the unmasked image because it's prettier, but you should do your analysis on the masked version. If the cartoon above doesn't show up, check the PDF version.

- Plot the radial surface brightness profile vs distance in pixels. Use a log scale for the surface brightness. (Usually we do log10 but this is one case where it might be helpful to use the natural log, given the scaling relation below.)

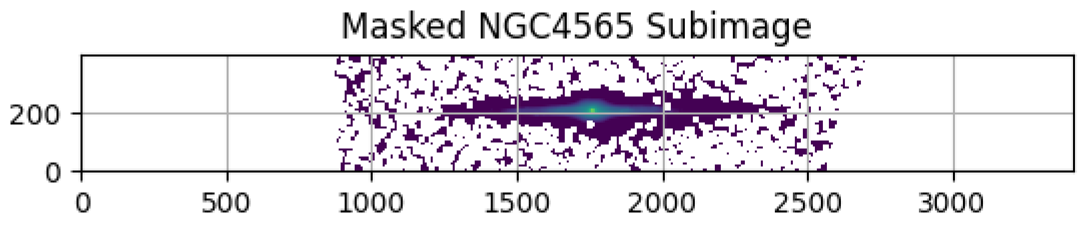
Checkpoint: Your radial surface brightness profile should have zero or near-zero values at its extreme ends, where there's no data. Set the plot y-limits to ignore those values and highlight the interesting structures in the central parts of the galaxy. Stepping inwards, there's a flat "sky" region where the profile is dominated by extragalactic background light. Then there's an approximately straight line region where the profile is dominated by the exponential disk, and a sharp peak in the center due to the bulge and the active nucleus. The whole galaxy is roughly 1200 pixels in diameter.

Spiral galaxy disks typically have exponential surface brightness profiles. Specifically, they follow $I(r) = I_0\, e^{-r/H_r}$, with a radial scale length parameter H_r that describes the distance over which the surface brightness drops by a factor of e. This form shows up as straight lines in a log-linear plot. (Verify for yourself: $\ln(I/I_0) = -r/H_r$, which makes a straight line if you plot $\ln I$ vs r.)

[8 pts]

In [80]:
```
# create the subimage
subimage = img2[1800:2200]

# display the subimage
plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
plt.title("Masked NGC4565 Subimage")
plt.grid()
```

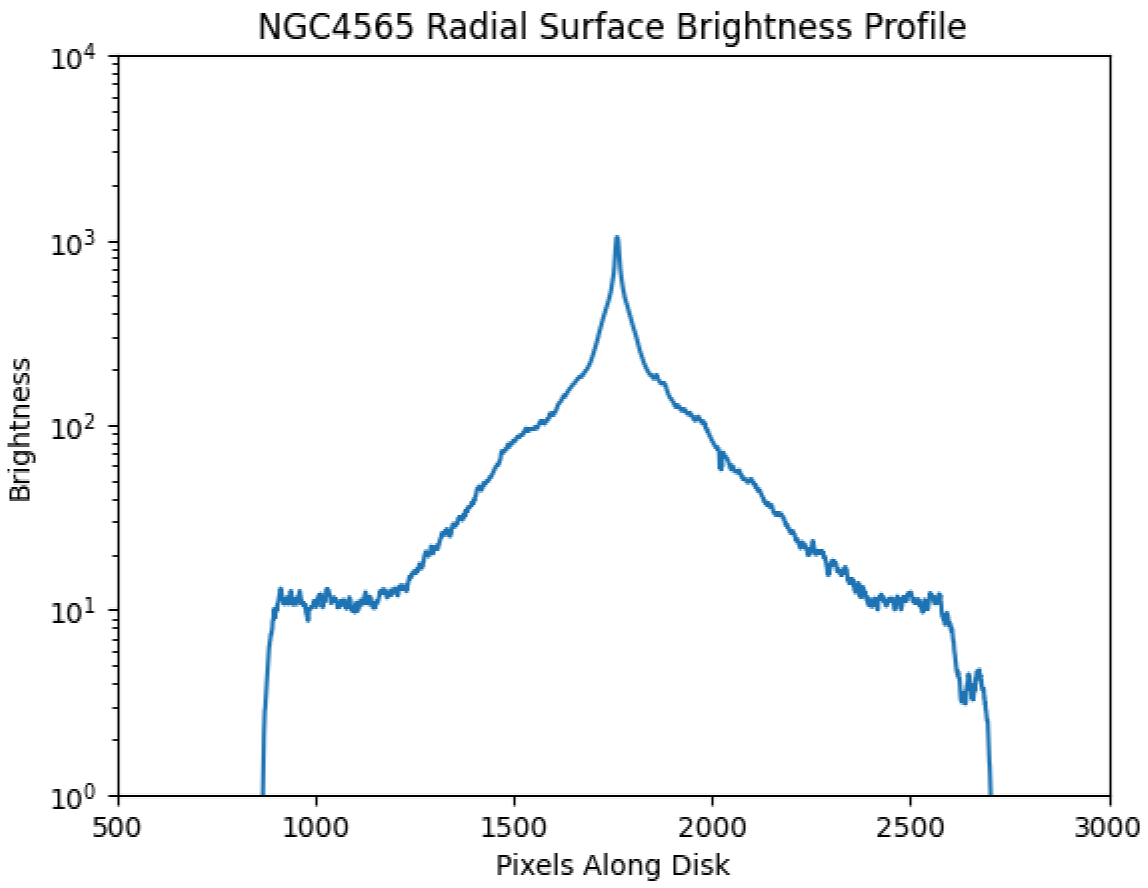
```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/1926833120.p
y:5: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/1926833120.p
y:5: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
```



In [81]:
```
# calculate the profile vector
profile = np.sum(subimage, axis=0)
# create the distance vector
distance = np.arange(0, subimage.shape[1], 1)
```

```
# plot the surface brightness profile
plt.plot(distance, profile)
plt.title("NGC4565 Radial Surface Brightness Profile")
plt.xlabel("Pixels Along Disk")
plt.ylabel("Brightness")
plt.yscale('log')
plt.ylim(1, 1e4)
plt.xlim(500, 3000)
```

```
Out[81]: (500.0, 3000.0)
```

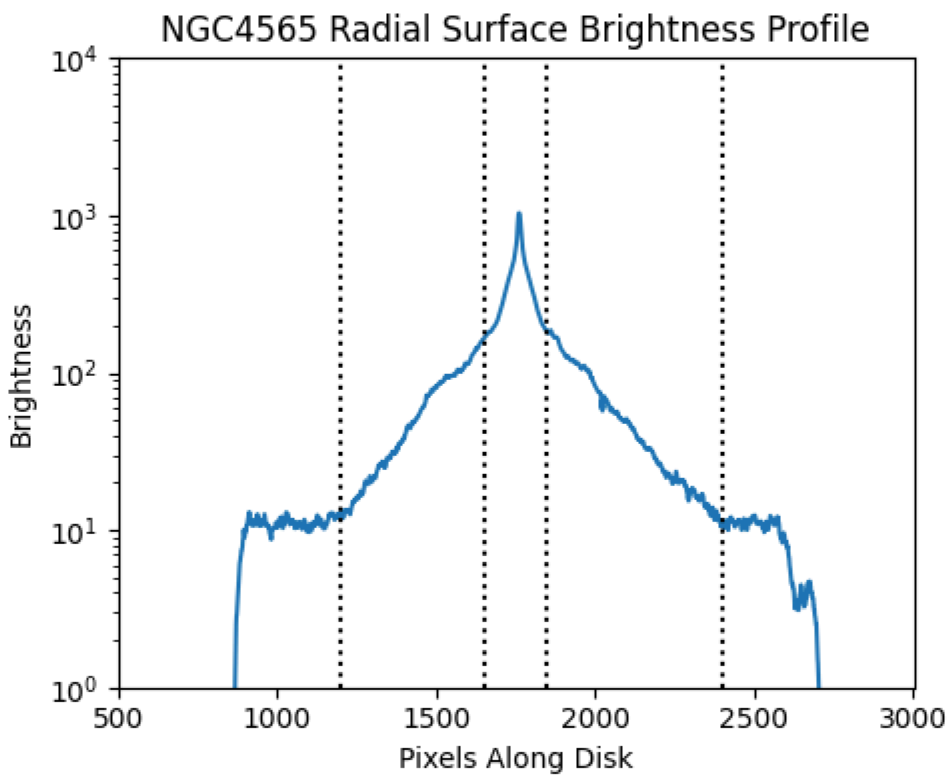


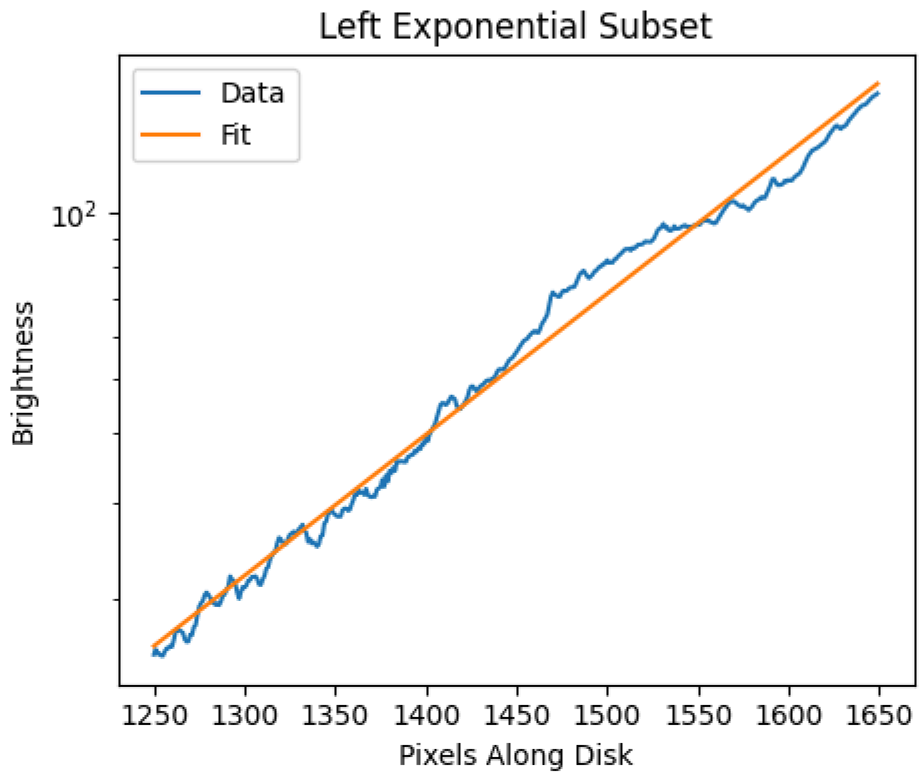
f.) Fit the radial surface brightness profile and measure the radial scale length. Specifically:

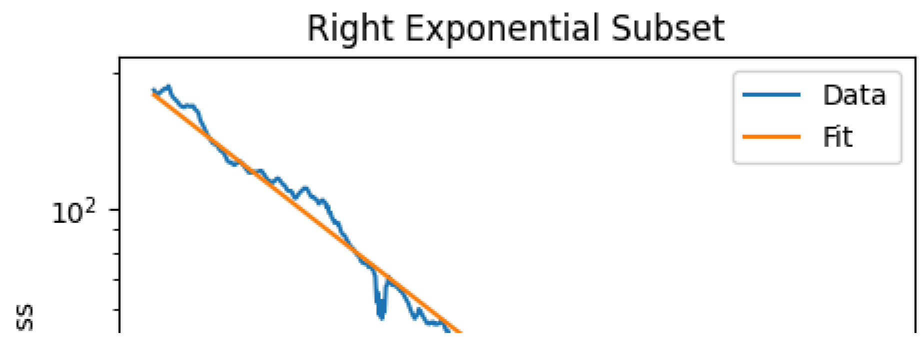
- Extract a subset of the surface brightness profile, covering just the exponential disk region on either the right or the left side of the nucleus.
- Fit the data for this subset with an exponential function as described above, or fit the log of the surface brightness with a straight line.
- Plot your best fit on top of the measured surface brightness profile to verify that it looks right.
- Do a similar fit for the other side of the galaxy.

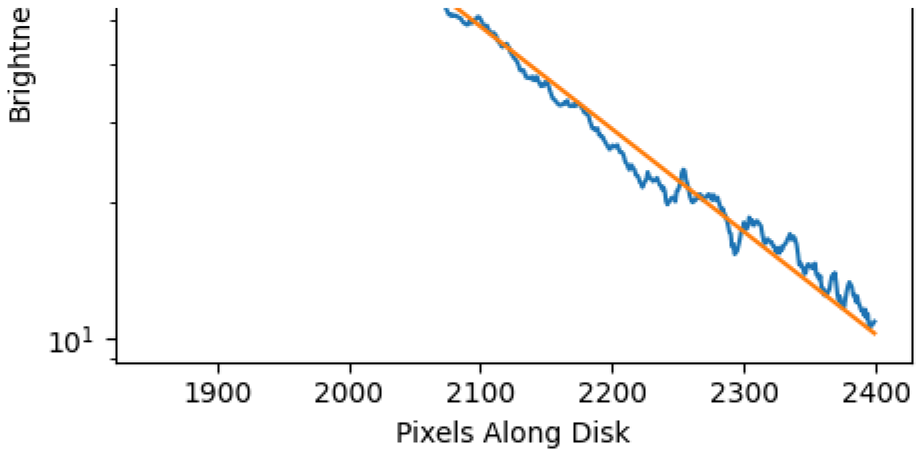
[10 pts]

In [82]:
```python
# plot the surface brightness profile
fig, ax = plt.subplots(3, 1, figsize=(5,12))
ax[0].plot(distance, profile)
ax[0].set_title("NGC4565 Radial Surface Brightness Profile")
ax[0].set_xlabel("Pixels Along Disk")
ax[0].set_ylabel("Brightness")
ax[0].set_yscale('log')
ax[0].set_ylim(1, 1e4)
ax[0].set_xlim(500, 3000)
# find the exponential regions
ax[0].axvline(1200, color='k', ls='dotted')
ax[0].axvline(1650, color='k', ls='dotted')
ax[0].axvline(1850, color='k', ls='dotted')
ax[0].axvline(2400, color='k', ls='dotted')

# cut out the exponential regions
profile_exp_left = profile[1250:1650]
distance_exp_left = distance[1250:1650]
profile_exp_right = profile[1850:2400]
distance_exp_right = distance[1850:2400]

# plot the left exponential subset
ax[1].plot(distance_exp_left, profile_exp_left, label='Data')
ax[1].set_title("Left Exponential Subset")
ax[1].set_xlabel("Pixels Along Disk")
ax[1].set_ylabel("Brightness")
ax[1].set_yscale('log')

# find a fit
r_left = linregress(distance_exp_left, np.log(profile_exp_left))
y_fit_left = np.exp(r_left.slope * distance_exp_left + r_left.intercept)
ax[1].plot(distance_exp_left, y_fit_left, label="Fit")
ax[1].legend()

# plot the right exponential subset
ax[2].plot(distance_exp_right, profile_exp_right, label='Data')
ax[2].set_title("Right Exponential Subset")
ax[2].set_xlabel("Pixels Along Disk")
ax[2].set_ylabel("Brightness")
ax[2].set_yscale('log')

# find a fit
r_right = linregress(distance_exp_right, np.log(profile_exp_right))
y_fit_right = np.exp(r_right.slope * distance_exp_right + r_right.intercept)
ax[2].plot(distance_exp_right, y_fit_right, label="Fit")
ax[2].legend()

plt.tight_layout()
```


NGC4565 Radial Surface Brightness Profile
Brightness
10^4
10^3
10^2
10^1
10^0
500
1000
1500
2000
2500
3000
Pixels Along Disk


Left Exponential Subset
Data
Fit
Brightness
10^2
1250
1300
1350
1400
1450
1500
1550
1600
1650
Pixels Along Disk


Right Exponential Subset
Data
Fit
10^2
ss



g.) Report your radial scale lengths

Print values for H_r in pixels, arcseconds (see the pixel scale info above), and kpc. For the conversion to kpc, you may assume a distance of 11.7 Mpc. (The distance to this galaxy still has a substantial uncertainty.) Radial scale lengths are typically in the range of a few to 10 kpc for normal-sized spiral galaxies.

[6 pts]

```
In [83]: # log(brightness) = -r * 1/H_r
         # slope = 1/H_r

         scale_length_left = 1 / r_left.slope

         print(f"The galaxy scale length is {scale_length_left} pixels")
         print(f"The galaxy scale length is {scale_length_left*pix_scale*3600} arcsec
         print(f"The galaxy scale length is {scale_length_left*pix_scale*np.pi/180*11
```

```
The galaxy scale length is 170.44752193550147 pixels
The galaxy scale length is 127.83564145162589 arcseconds
The galaxy scale length is 7.2512467453718985 kpc
```

h.) Create the vertical surface brightness profile

This time we will integrate along the direction parallel to the disk, so we get surface brightness as a function of distance above and below the midplane. But don't integrate the whole galaxy; using your radial surface brightness plot or your displayed image as a guide, choose a restricted portion offset from the nucleus as suggested in the cartoon above.

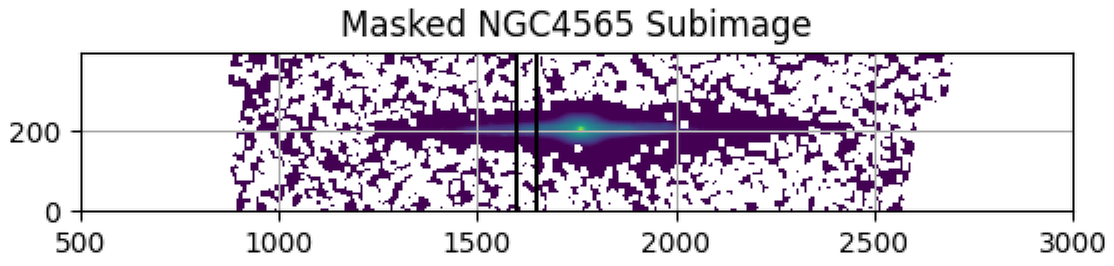
- Plot your rotated image again and draw some lines on it to indicate the region you're using for your vertical profile. I like `plt.axvline`.
- Plot the vertical surface brightness profile vs distance in pixels. As before, use a log scale for the surface brightness.

[10 pts]

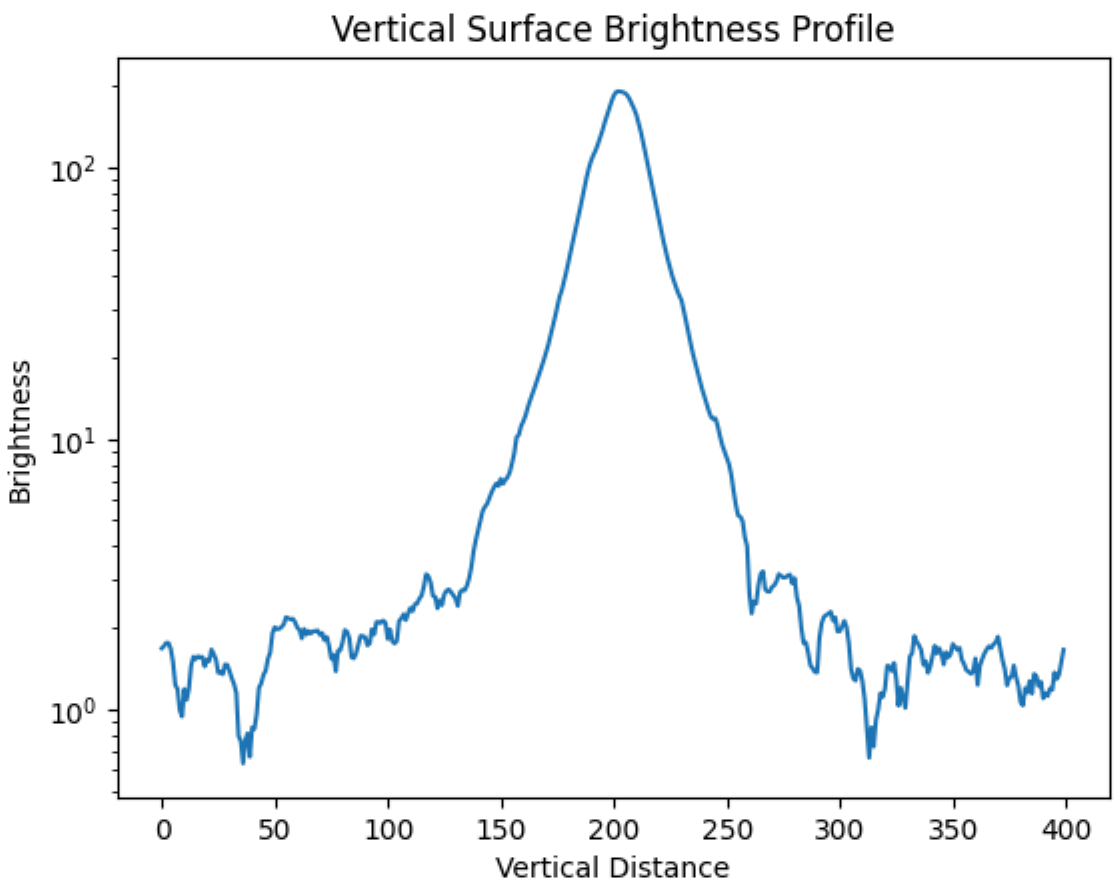
```
In [84]: # display the subimage
         plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
         plt.title("Masked NGC4565 Subimage")
         plt.grid()
         plt.axvline(1600, color='k')
         plt.axvline(1650, color='k')
         plt.xlim(500, 3000)
```

```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/3593361052.p
y:2: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/3593361052.p
y:2: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
```

```
Out[84]: (500.0, 3000.0)
```



```
In [85]: # calculate the profile vector
         profile_vert = np.sum(subimage[:,1600:1650], axis=1)
         # create the distance vector
         distance_vert = np.arange(0, 400, 1)

         plt.plot(distance_vert, profile_vert)
         plt.xlabel("Vertical Distance")
         plt.ylabel("Brightness")
         plt.title("Vertical Surface Brightness Profile")
         plt.yscale('log')
```



i.) Fit the vertical surface brightness profile

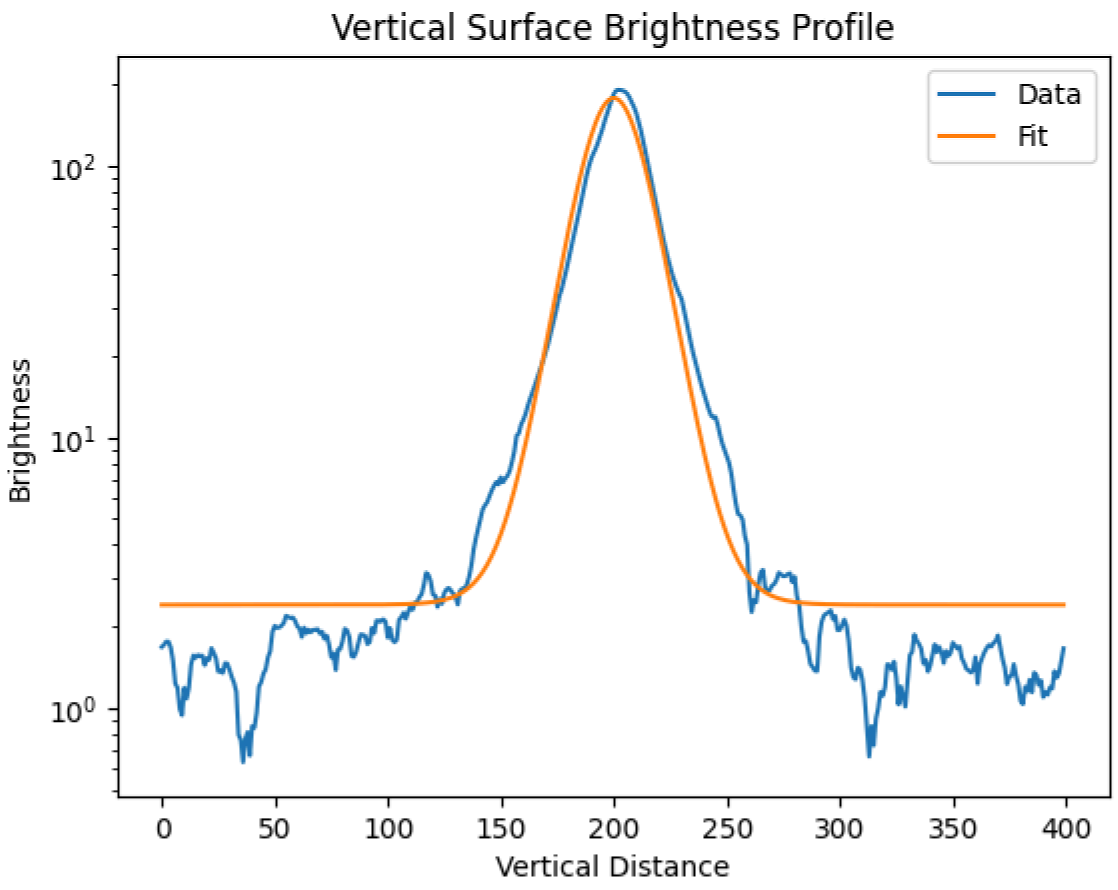
- Fit with a sech^2 function: $I(z) = I_0\ \mathrm{sech}^2((z - z_0)/H_z) + \mathcal{C}$, where $\mathcal{C}$ is a constant offset describing the background sky level.
- Plot your best-fit sech^2 function on top of your measured surface brightness profile to ensure the fit has done what you wanted.
- Report the vertical scale height H_z in pixels, arcseconds, and kpc.

Typical values for the vertical scale heights are less than 1 kpc. You may notice that the actual profile has stronger "wings" than the sech^2 -- that might be evidence for a thick disk or a galactic halo.

[12 pts]

In [86]:
```
# perform the curve fit
def func(z, I0, H_z, C):
    return (I0 * (1/np.cosh((z-200)/H_z))**2) + C

p0, cov = curve_fit(func, distance_vert, profile_vert, [1,1,1])

# plot the data
plt.plot(distance_vert, profile_vert, label="Data")
plt.xlabel("Vertical Distance")
plt.ylabel("Brightness")
plt.title("Vertical Surface Brightness Profile")
plt.yscale('log')

# plot the fit
plt.plot(distance_vert, func(distance_vert, *p0), label="Fit")
plt.legend()
```

Out[86]:
```
<matplotlib.legend.Legend at 0x321f3faa0>
```



In [87]:
```
scale_length_left = p0[1]

print(f"The galaxy scale height is {scale_length_left} pixels")
print(f"The galaxy scale height is {scale_length_left*pix_scale*3600} arcsec
print(f"The galaxy scale height is {scale_length_left*pix_scale*np.pi/180*11
```

```
The galaxy scale height is 17.030793806151177 pixels
The galaxy scale height is 12.773095354613362 arcseconds
The galaxy scale height is 0.7245308512298865 kpc
```

OPTIONAL! Extra credit.

Explore one or more of these extensions.

For #1 (Fourier transform): Show that you get an even better match to the observed power spectrum if you generate a test signal that is sampled at the same times as the real data stream, and then interpolate. This procedure reproduces the fact that the yearly pattern is somewhat irregular. (Sometimes people are motivated to do pre-dawn observing and sometimes they aren't.)

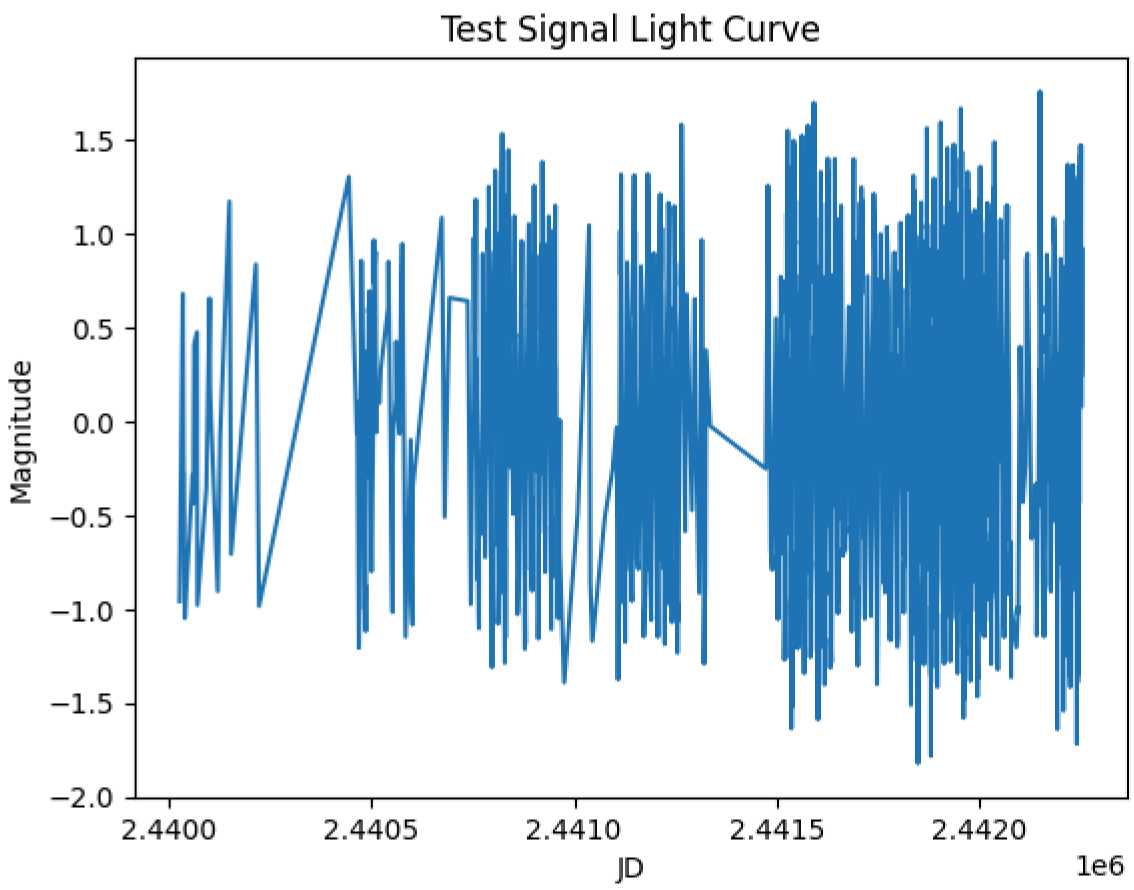
For #3 (the disk structure project):

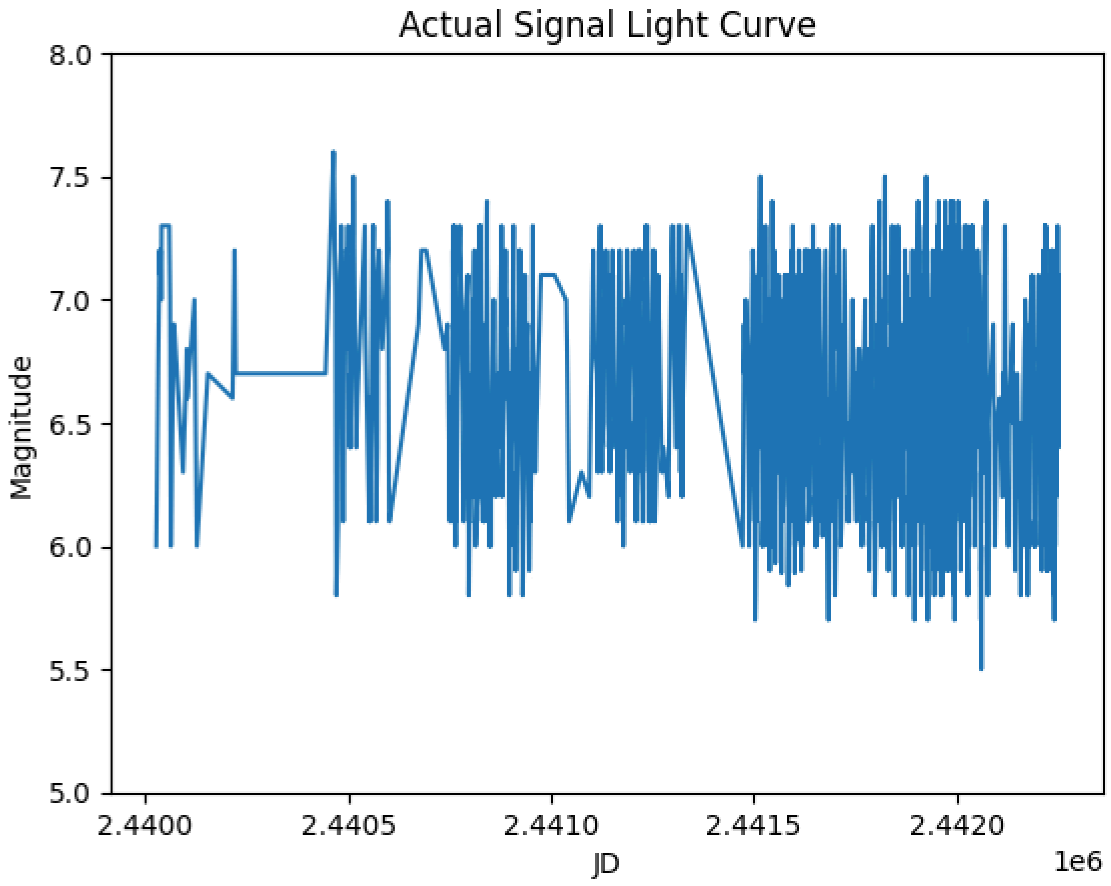
- Does the vertical scale height depend on the profile's radial offset from the nucleus? In general, disks should flare, meaning that the scale height should get larger the farther out you move in radius. Measure the vertical scale height at several locations across the disk and plot height vs radius.
- Does it make a difference whether you mask the pixels first and then rotate the image, or rotate first and then mask?
- Do you get different answers from the two possible methods for the radial surface brightness fit (fitting an exponential function vs fitting a straight line to the log of the surface brightness)?

[up to 12 pts]

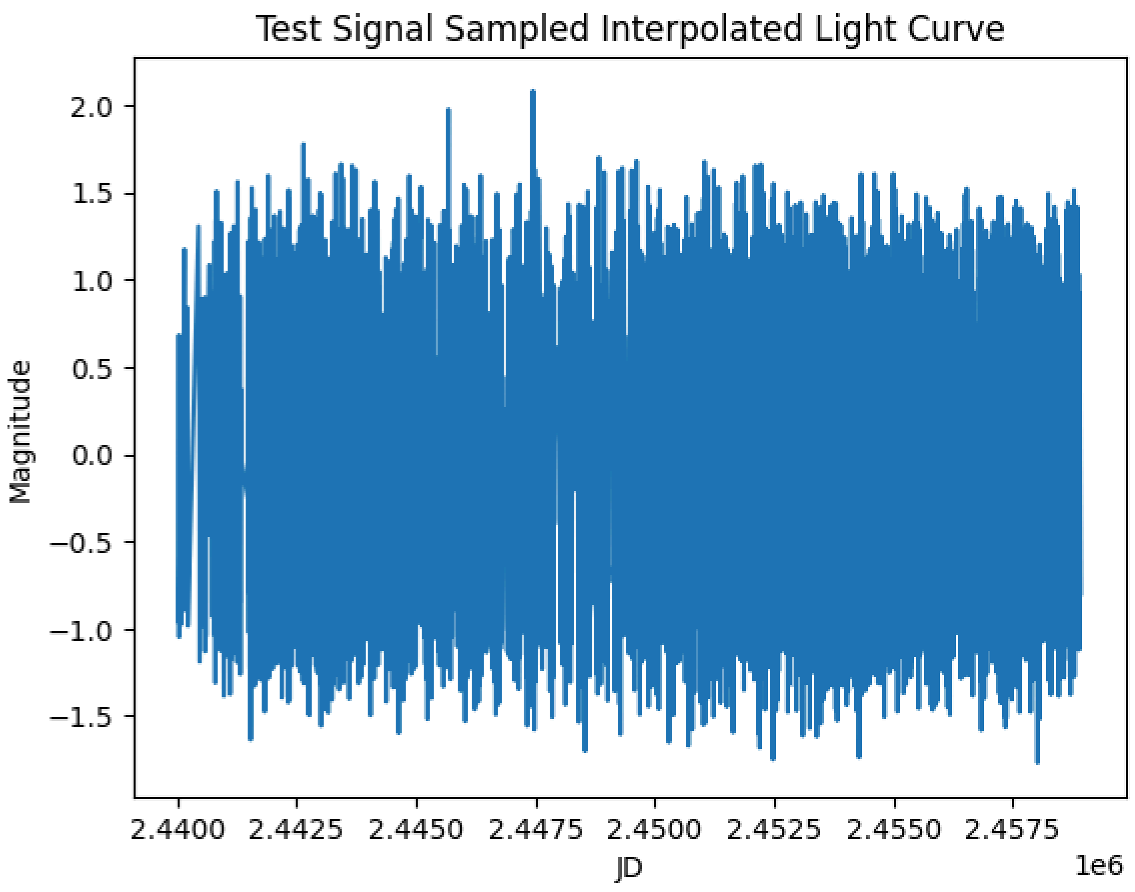
Extra Credit For #1

In [88]:
```
# create the test signal
test_signal_real_sampled = np.sin(2*np.pi*fund_freq*t["JD"]) + np.random.nor

# plot the test signal
plt.plot(t["JD"][0:2000], test_signal_real_sampled[0:2000])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("Test Signal Light Curve")
plt.show()

# plot the real signal
plt.plot(t["JD"][0:2000], t["Magnitude"][0:2000])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("Actual Signal Light Curve")
plt.ylim(5,8)
plt.show()
```


Test Signal Light Curve
Magnitude
JD
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2.4400
2.4405
2.4410
2.4415
2.4420


Actual Signal Light Curve
Magnitude
JD
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
2.4400
2.4405
2.4410
2.4415
2.4420

In [89]:
```
# interpolate with sampling interval of 1 day
interp2 = scipy.interpolate.interp1d(t["JD"], test_signal_real_sampled, kinc
t_interp2 = table.Table()
t_interp2["interp_JD"] = np.arange(min(t["JD"]), max(t["JD"]), 1)
t_interp2["interp_mag"] = interp2(t_interp2["interp_JD"])

# plot the correctly sampled and then interpolated test signal
plt.plot(t_interp2["interp_JD"], t_interp2["interp_mag"])
plt.xlabel("JD")
plt.ylabel("Magnitude")
plt.title("Test Signal Sampled Interpolated Light Curve")
plt.show()
```

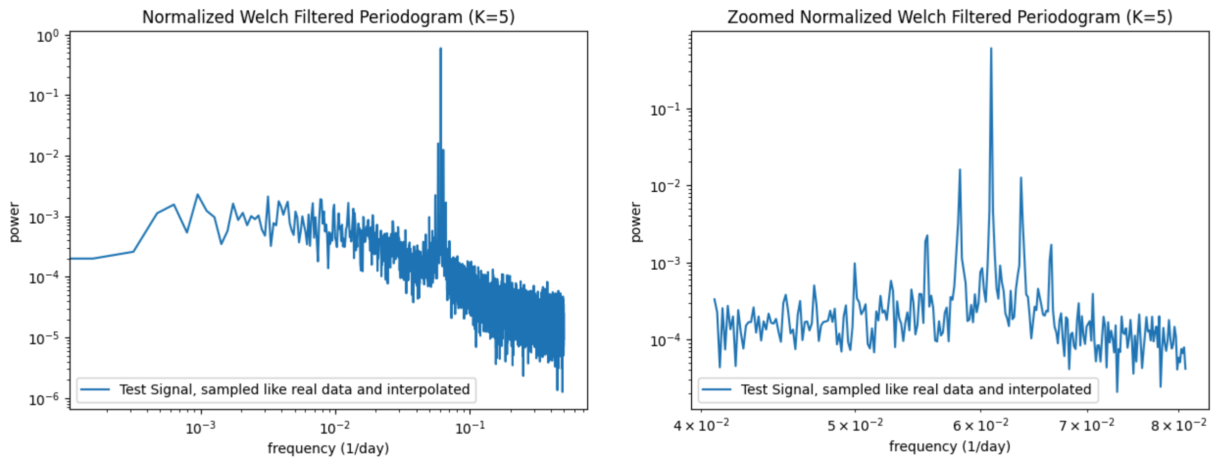


In [90]:
```
k_test2, P_test2 = compute_welch_filtered_periodogram(t_interp2["interp_JD"]

# calculate the zoomed interval
fund_freq_test2 = k_test2[P_test2[1::].argmax()]
k_min_test2 = fund_freq_test2 - 0.02
k_max_test2 = fund_freq_test2 + 0.02

k_zoom_test2 = k_test2[(k_test2 > k_min_test2) & (k_test2 < k_max_test2)]
P_zoom_test2 = P_test2[(k_test2 > k_min_test2) & (k_test2 < k_max_test2)]

# plot the periodigram
fig, ax = plt.subplots(1, 2, figsize=(15,5))

ax[0].loglog(k_test2, P_test2, label="Test Signal, sampled like real data ar
ax[0].set_xlabel("frequency (1/day)")
```

```
ax[0].set_ylabel("power")
ax[0].set_title("Normalized Welch Filtered Periodogram (K=5)")
ax[0].legend()

ax[1].loglog(k_zoom_test2, P_zoom_test2, label="Test Signal, sampled like re
ax[1].set_xlabel("frequency (1/day)")
ax[1].set_ylabel("power")
ax[1].set_title("Zoomed Normalized Welch Filtered Periodogram (K=5)")
ax[1].legend()
```

Out[90]:

```
<matplotlib.legend.Legend at 0x321eef950>
```



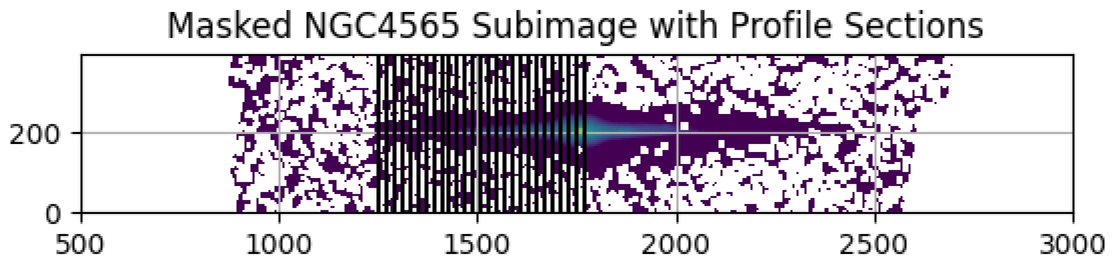
Extra Credit For #3

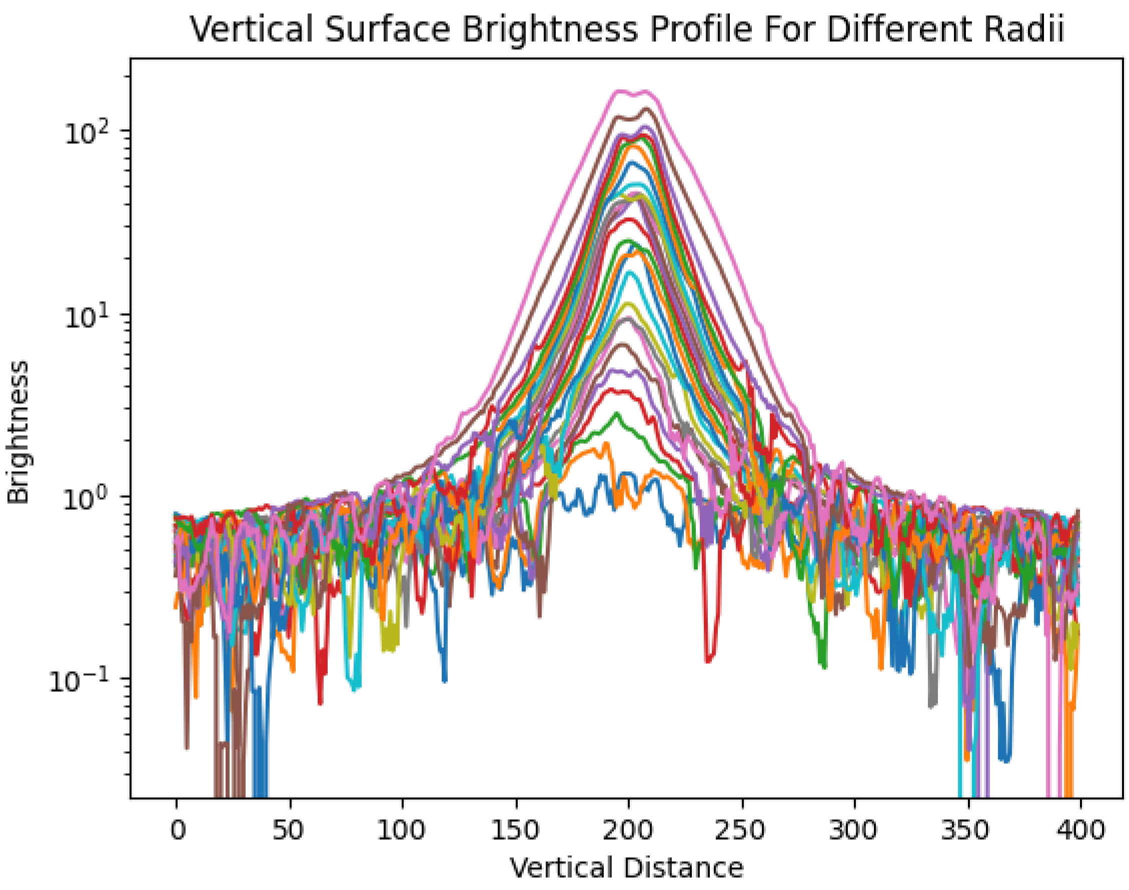
In [91]:

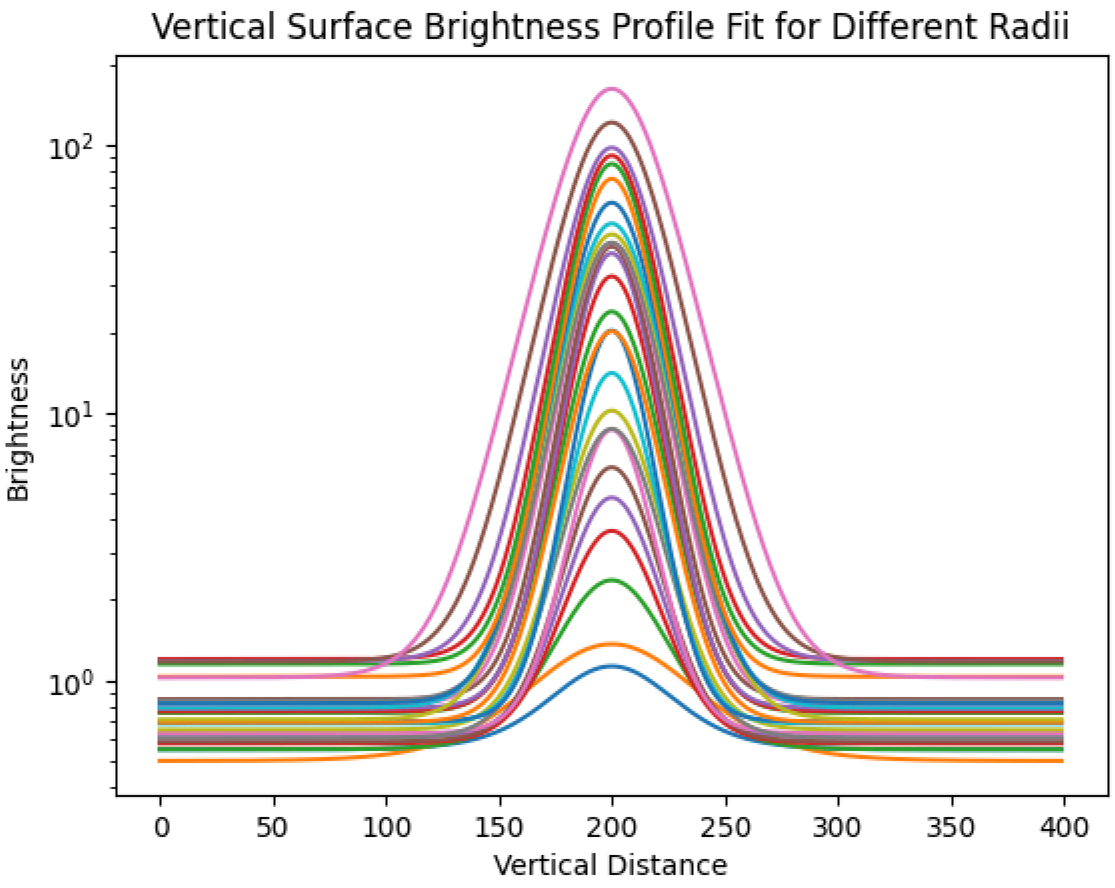
```
# display the subimage
plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
plt.title("Masked NGC4565 Subimage with Profile Sections")
plt.grid()
plt.xlim(500, 3000)

for i in range(27):
    plt.axvline(1250 + 20*i, color='k')

plt.show()

# calculate the profile vectors
profiles = []
for i in range(27):
    profile_vert = np.sum(subimage[:,1200+20*i:1220+20*i], axis=1)
    profiles.append((profile_vert, i))

# create the distance vector
distance_vert = np.arange(0, 400, 1)

# display the data
for profile, i in profiles:
    plt.plot(distance_vert, profile, label=i)
plt.xlabel("Vertical Distance")
plt.ylabel("Brightness")
plt.title("Vertical Surface Brightness Profile For Different Radii")
plt.yscale('log')
```

```python
plt.show()

# calculate curve fits
heights = []
for profile, i in profiles:
    p0, cov = curve_fit(func, distance_vert, profile, [1,1,1])
    heights.append(p0[1])

    # plot the fit
    plt.plot(distance_vert, func(distance_vert, *p0), label="Fit")
    plt.xlabel("Vertical Distance")
    plt.ylabel("Brightness")
    plt.title("Vertical Surface Brightness Profile Fit for Different Radii")
    plt.yscale('log')
```

```
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/357597815.p
y:2: RuntimeWarning: divide by zero encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
/var/folders/41/_gkgvhb94wd4156zplzr4cg00000gn/T/ipykernel_1225/357597815.p
y:2: RuntimeWarning: invalid value encountered in log10
  plt.imshow(np.log10(subimage), vmin=0, vmax=2, origin='lower')
```

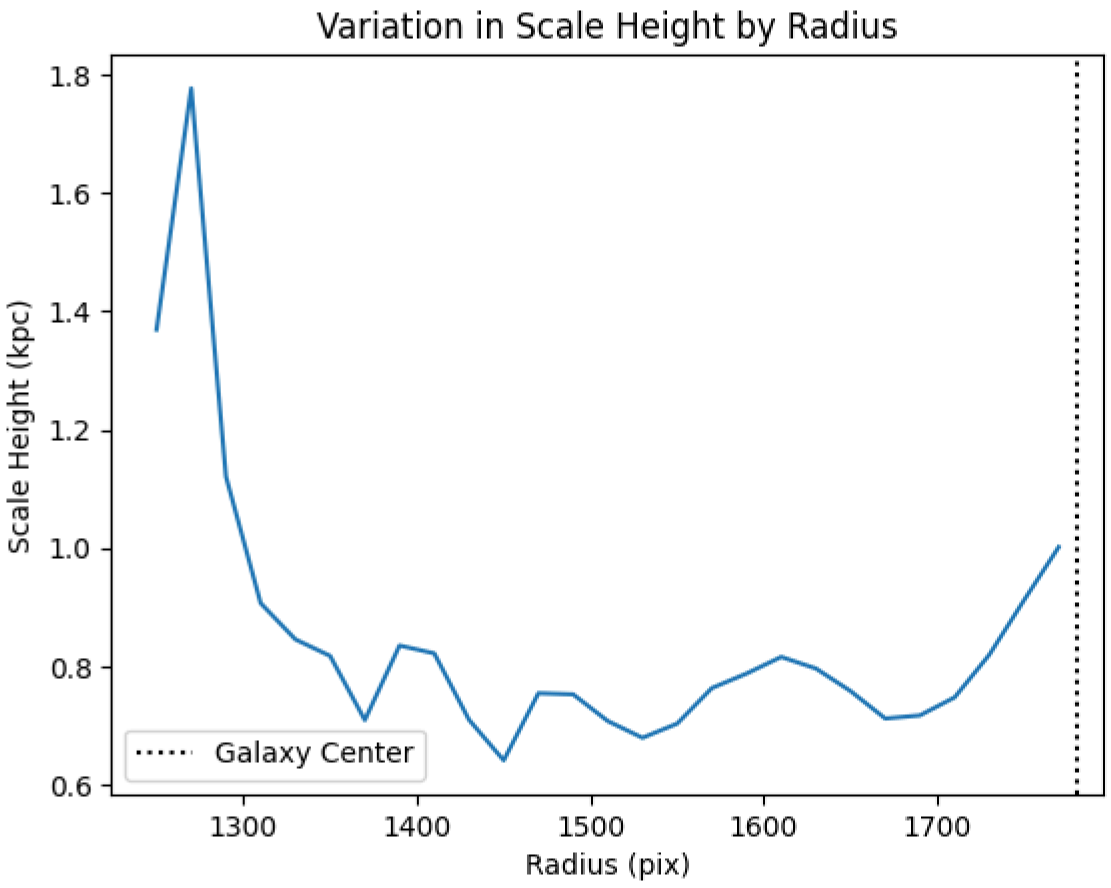



Vertical Surface Brightness Profile For Different Radii
Brightness
Vertical Distance
10^2
10^1
10^0
10^{-1}
0
50
100
150
200
250
300
350
400


Vertical Surface Brightness Profile Fit for Different Radii
Brightness
Vertical Distance
10^2
10^1
10^0
0
50
100
150
200
250
300
350
400

In [92]:
```
heights = np.array(heights)*pix_scale*np.pi/180*11700
radii = np.arange(1250, 1790, 20)
plt.plot(radii, heights)
plt.xlabel("Radius (pix)")
plt.ylabel("Scale Height (kpc)")
plt.axvline(1780, label='Galaxy Center', color='k', ls='dotted')
plt.legend()
plt.title("Variation in Scale Height by Radius")
```

Out[92]:
```
Text(0.5, 1.0, 'Variation in Scale Height by Radius')
```



In [ ]: