Using Exoplanet Transits to Meausure Exoplanet Radius, Mass, and Density

Jack Colvin, Along with my group, Larrence Xing, Kevin Wu, Annabelle Yarborough and Olivia Lucal

1. Introduction

Since the discovery of the first transiting exoplanet in 1999, exoplanet transits have been one of the main ways to detect exoplanets, along with measuring radial velocity, direct imaging, gravitational microlensing, and astrometry. Out of these methods, the transit method has been by far the most prevalent and widely used, with 4278 planets discovered using this method since 1999, including over 3000 planets discovered as a part of NASA's Kepler and K2 missions. We decided to use the transit method both because of its prevalence and because it is a method that is relatively measurable by SEO (at least in theory). We specifically attempted to measure the transits of four different planets, with varying successes: TOI-4138b, TOI-2260b, TOI-1836b, and TOI-2591b (which was an archival transit observation which my group used to ensure that our method for measuring transits was correct so long as the transit depth was sufficient). Numerical details about all of these planets, whether or not we were able to obtain a transit for each, and the lowestlevel of noise we were able to acheive are outlined below:

Property	TIC 257060897 b	TOI 2260 b	TOI-1836b	TOI-2591b
Star V-Band Luminosity	11.8	10.5	9.8	10.5
Expected Transit Depth (ppt)	7.7	.3	.24	20.1
Expected Mid Transit Date (J2000)	10809.8572	10813.9010	10815.7316	?
Best Acheived RMS noise (ppt)	20	7	5.4	4.5
Transit Extracted?	N	N	N	Y

Thus, while we were unable to discern a transit, there are many things that we can still take away from this Lab. First of all, we were able to considerably reduce our observed RMS noise, which we did by increaing our exposure time. We found that we could still get around 50 data points within our transit while using a longer exposure (120s vs 30s), which greatly increased our SNR and thus allowed us to acheive a much lower level of noise. We also, however, realized that there was a limit to the amount this noise could be reduced (~5 ppt) which meant that many of our transits were limited by the fact that they were simply far too faint to be detected by SEO, as their depth was much less than what could be acheived even with the best possible SNR one could expect with SEO. Finally, we were able to use an archival measurement to verify that our analysis techniques work properly, and to conduct basic analysis on a real transit.

2. Methods and Data Collection

In python, we used DAOStarFinder to locate our star along with multiple comparison stars within the Image, as in each observation, the brightest star within our cropped frame was also our target star, making it very easy to pinpoint by sorting based on DAO's measure of estimated flux. We chose to do this instead of using RA and DEC as our star shifted considerably between frames, and found it easiest to use DAO for our analysis. We then used the next two brightest stars within our cropped frame as our comparison stars, as they were most likely to have similar flux values to our target star while also being relatively close to our target star in the image. In all cases, we double checked that we were recieving the correct coordinates of our target star and two consistent comparison stars by using aperture.plot, and were able to verify that DAOStarFinder was succesful in consistenty identifying all 3 sources for all 4 of our observations in every frame. After finding the coordinates of our stars, we used Circular Aperture to create an Aperture, the size of which we estimated using DAOStarFinder's approximation of pixel size of our source, and then aperture_photometry to define a flux value for each star, making sure to subtract the median value multiplied by the area of our aperture (representing the background flux) from each of the three sources respectively. Finally, we divided our target stars flux by one of the comparison star's in order to correct for any systematics that might be affecting our measurement, such as variable seeing and weather conditions, and then by the median value of our rescaled fluxes in order to normalize the values.

To analyze our results, we plotted these flux values vs the time values from our expected mid transit time, and fit a model using a running mean, to try to discern whether a transit was visible or not. We then measured transit depth and subsequently aimed to calculate planet radius for any planets with observable transits, and we defined a maximum possible transit depth for all of the observations for which we did not see a transit by using the formula flux error = $\frac{RMS}{sqrt(N)}$, where N is the number of points within our transit, RMS represents the intrinsic scatter of my data points calculated as a running standard deviation (so as to not include systematic trends in our evalution of scatter), and aimed for a 3 sigma result to call a transit observable. Finally, we also calculated the propegated poisson noise for all of my flux values and compared it to our measurement of scatter to attempt to determine whether the intrinsic scatter in our data points was due simple to poisson noise or if there were other statistical errors affecting my calculation. These two measurements should be directly comparable, as they both are measured to one standard deviation. Finally, I think it is important to mention that we did not run a chi-squared calculation for my models, as they were generated using a running mean, which would contradict the independece condition for using chi2, as or model is generated by a simple average of the points and thus the model will cohere to my data in a way that isn't conducive to using chi-squared.

In [224…
```
import os
```

```
from astropy.nddata import Cutout2D
from astropy.io import fits
from photutils.aperture import CircularAperture, aperture_photometry
from photutils.detection import DAOStarFinder
from astropy.stats import sigma_clipped_stats
import numpy as np
import matplotlib.pyplot as plt
from astropy.time import Time
from astropy.visualization import ZScaleInterval
from astropy.modeling import models, fitting
from astropy.coordinates import SkyCoord
import astropy.units as u
from PIL import Image
from astropy.time import Time

def plot_prettier(dpi=150, fontsize=11, usetex=False):
    '''
    Make plots look nicer compared to Matplotlib defaults
    Parameters:
        dpi - int, "dots per inch" - controls resolution of PNG images that
                by Matplotlib
        fontsize - int, font size to use overall
        usetex - bool, whether to use LaTeX to render fonds of axes labels
                use False if you don't have LaTeX installed on your system
    '''
    plt.rcParams['figure.dpi']= dpi
    plt.rc("savefig", dpi=dpi)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=5)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=5)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [2., 2.])
    if usetex:
        plt.rc('text', usetex=usetex)
    else:
        plt.rcParams['mathtext.fontset'] = 'cm'
        plt.rcParams['font.family'] = 'serif'
        plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['fon


plot_prettier(dpi=150, fontsize=11)

def running_mean_std(input_array, w):
    run_mean = np.zeros(len(input_array))
    run_std = np.zeros(len(input_array))
    for i in range(len(input_array)):

        if i - w < 0:

            run_mean[i] = np.mean(input_array[0:i+w])
```

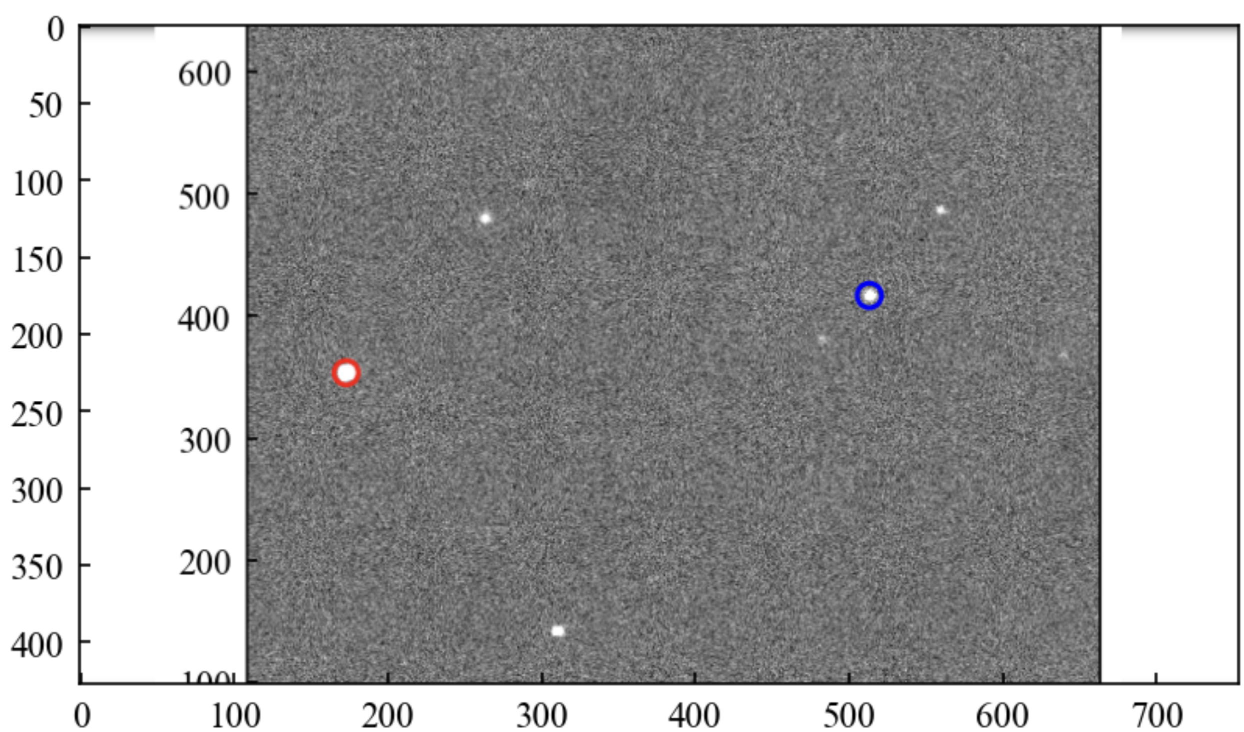
```
            run_std[i] = np.std(input_array[0:i+w])
        elif i + w > len(input_array):
            run_mean[i] = np.mean(input_array[i-w:i+1])
            run_std[i] = np.std(input_array[i-w:i+1])

        else:
            run_mean[i] = np.mean(input_array[i-w:i+w])
            run_std[i] = np.std(input_array[i-w:i+w])

    return run_mean, run_std
```

Data Collection & Transit Extraction for TOI-4138b

In [225…

```
#this Image show the star we are looking at and the aperture we used.
Img_4138 = Image.open("/Users/jackcolvin//Screenshot 2025-05-23 at 11.51.52
plt.imshow(Img_4138)
```

Out[225]: <matplotlib.image.AxesImage at 0x29257ab50>



In [226…

```
folder = "/Users/jackcolvin/downloads/Exoplanet Data 2"
filenames = sorted([f for f in os.listdir(folder) if f.endswith('fits')])

fluxes = []
fluxes2 = [] #use this to define a comparison star
times = []
fluxes3 = []

for i in filenames:
```

```
    with fits.open(os.path.join(folder, i)) as hdul:
        data = hdul[0].data
        header = hdul[0].header
        cutout = Cutout2D(data, (1183, 1098), (700, 700))
        data = cutout.data
        filt = np.isnan(data)
        data[filt] = np.median(data)
        date = header.get('DATE-OBS')
        mean, median, std = sigma_clipped_stats(data, sigma = 3)
        maxval = np.max(data)
        daofind = DAOStarFinder(fwhm=10, threshold = 4*std)
        #I was noticing that it sometimes identified out central star
        #as two seperate objects
        #so I made the fwhm much higher to account for this
        #even though my estimate of fhwm from Lab #1 was half of this estima

        #intervals = ZScaleInterval()
        #vmins, vmaxs = intervals.get_limits(data)
        #plt.imshow(data, cmap = 'gray', origin = 'lower', vmin = vmins, vma
        #I used the value calculated in Lab 1 for FWHM '
        #assuming that it is relatively constant

        stars = daofind(data)
        brightest = stars[np.argmax(stars['flux'])]
        r_est1 = np.sqrt(brightest['npix']/np.pi)
        coords = np.array([brightest['xcentroid'], brightest['ycentroid']])
        aperture = CircularAperture(coords, r = 10)
        #aperture.plot(color='red', lw=1.5, label='Target Star')
        phot_table = aperture_photometry(data, aperture)
        flux = phot_table['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        #set collumn with highest flux to 0 to now find the second brightest
        #which we will use as our comparison star
        brightest2 = stars[np.argmax(stars['flux'])]
        r_est2 = np.sqrt(brightest2['npix']/np.pi)
        coords2 = np.array([brightest2['xcentroid'], brightest2['ycentroid']
        aperture2 = CircularAperture(coords2, r = 10)
        #aperture2.plot(color='green', lw=1.5, label='Target Star')
        phot_table2 = aperture_photometry(data, aperture2)
        flux2 = phot_table2['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        brightest3 = stars[np.argmax(stars['flux'])]
        coords3 = np.array([brightest3['xcentroid'], brightest3['ycentroid']
        r_est3 = np.sqrt(brightest3['npix']/np.pi)
        aperture3 = CircularAperture(coords3, r = 10)
        #aperture3.plot(color='blue', lw=1.5, label='Target Star')
        phot_table3 = aperture_photometry(data, aperture3)
        flux3 = phot_table3['aperture_sum'][0] - median*np.pi*100
        fluxes.append(float(flux))
        fluxes2.append(float(flux2))
        fluxes3.append(flux3)
```

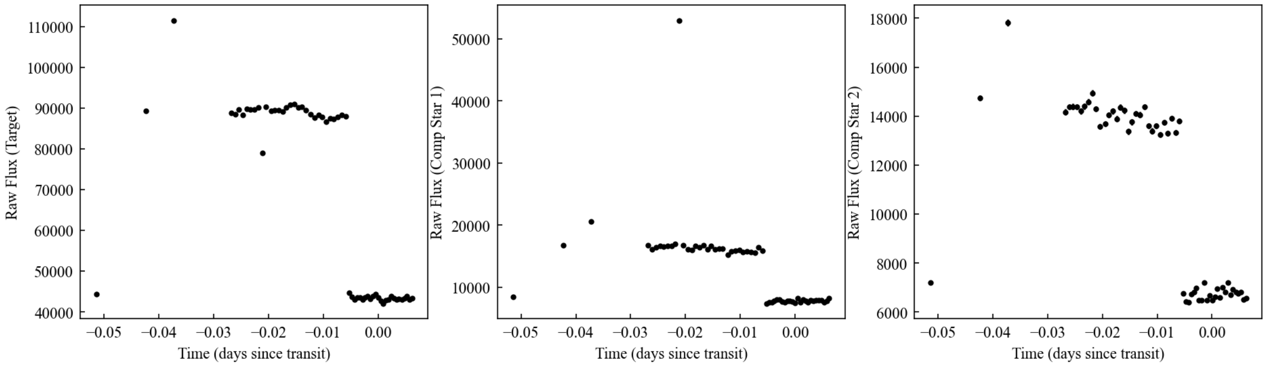
```
        mjd = header.get('MJD-OBS')
        #jd = mjd + 2400000.5
        bjd = 10809.8572 + 2450000
        t = Time(bjd, format = 'jd', scale = 'tdb')
        mjd_mid = t.mjd
        transit = mjd - mjd_mid
        times.append(transit)

        plt.show()


flux = np.array(fluxes)
flux2 = np.array(fluxes2)
times = np.array(times)
flux3 = np.array(fluxes3)


zipped_array = list(zip(times, flux, flux2, flux3))
zipped_array.sort()

times, flux, flux2, flux3 = zip(*zipped_array)
flux = np.array(flux)
flux2 = np.array(flux2)
times = np.array(times)
flux3 = np.array(flux3)
```

In [227…

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))


sigma_f = np.sqrt(flux)
sigma_f2 = np.sqrt(flux2)
sigma_f3 = np.sqrt(flux3)

axes[0].errorbar(times, flux, yerr=sigma_f, fmt = '.', color = 'black')
axes[0].set_xlabel("Time (days since transit)")
axes[0].set_ylabel("Raw Flux (Target)")



axes[1].errorbar(times, flux2, yerr=sigma_f2, fmt= '.', color = 'black')
axes[1].set_xlabel("Time (days since transit)")
axes[1].set_ylabel("Raw Flux (Comp Star 1)")



axes[2].errorbar(times, flux3, yerr=sigma_f3, fmt = '.', color = 'black')
axes[2].set_xlabel("Time (days since transit)")
axes[2].set_ylabel("Raw Flux (Comp Star 2)")
```

Out[227]:

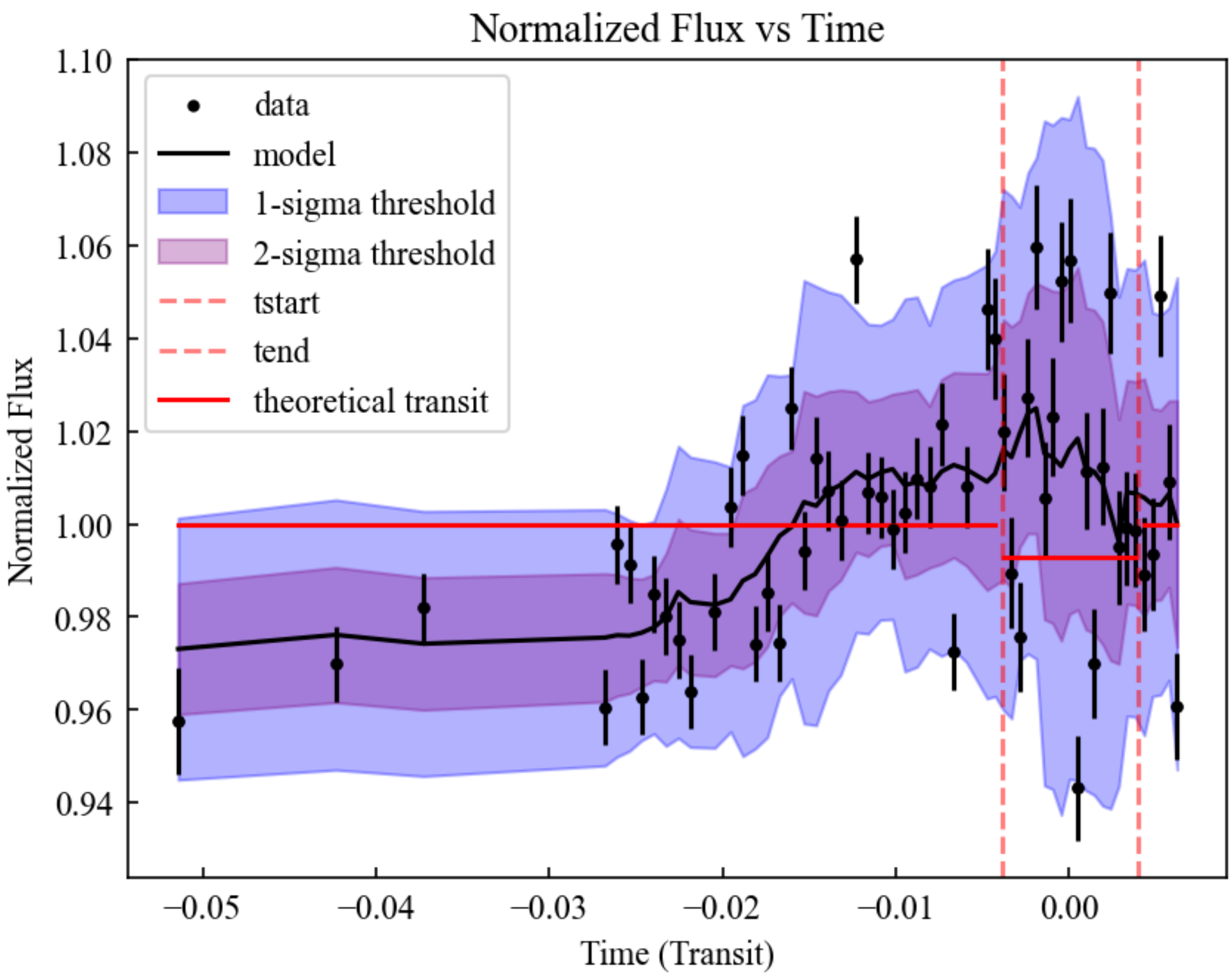
```
Text(0, 0.5, 'Raw Flux (Comp Star 2)')
```


110000
100000
90000
80000
70000
60000
50000
40000
Raw Flux (Target)
−0.05 −0.04 −0.03 −0.02 −0.01 0.00
Time (days since transit)
50000
40000
30000
20000
10000
Raw Flux (Comp Star 1)
−0.05 −0.04 −0.03 −0.02 −0.01 0.00
Time (days since transit)
18000
16000
14000
12000
10000
8000
6000
Raw Flux (Comp Star 2)
−0.05 −0.04 −0.03 −0.02 −0.01 0.00
Time (days since transit)

In [230…

```
corr_flux = flux / flux2

prop_error = corr_flux  * np.sqrt((sigma_f/flux)**2+(sigma_f2/flux2)**2)

prop_error1 = prop_error / np.median(corr_flux)
corr_flux = corr_flux / np.median(corr_flux)

mask =  (corr_flux > .9) & (corr_flux < 1.1)

times1 = times[mask]
corr_flux1 = corr_flux[mask]
prop_error2 = prop_error1[mask]
#we need to cut some values from corr_flux

run_mean, run_std = running_mean_std(corr_flux1, 5)
plt.scatter(times1, corr_flux1, color = 'black', marker = '.')
plt.errorbar(times1, corr_flux1, yerr=prop_error2, color = 'black', fmt = '.
plt.plot(times1, run_mean, color = 'black')
plt.fill_between(times1, run_mean - 2*run_std, run_mean + 2*run_std, alpha =
plt.fill_between(times1, run_mean - run_std, run_mean + run_std, alpha = .3,

tstart = 10809.8533 - 10809.8572
tend = 10809.8612 - 10809.8572
plt.axvline(tstart, linestyle = 'dashed', color = 'red', alpha = .5)
plt.axvline(tend, linestyle = 'dashed', color = 'red', alpha = .5)

RMS = np.median(run_std)
mask = (times1 > tstart) & (times1 < tend)
mask2 = (times1 < tstart)
mask3 = times1 > tend
corr_flux2 = corr_flux1[mask]
times2 = times1[mask]
corr_flux3 = corr_flux1[mask2]
times3 = times1[mask2]
corr_flux4 = corr_flux1[mask3]
times4 = times1[mask3]
plt.plot(times3, np.ones(len(times3)), color = 'red')
plt.plot(times2, np.ones(len(times2)) - .007, color = 'red')
plt.plot(times4, np.ones(len(times4)), color = 'red')
plt.xlabel('Time (Transit)')
plt.ylabel('Normalized Flux')

plt.title('Normalized Flux vs Time')
plt.legend(['data', 'model', '1-sigma threshold','2-sigma threshold', 'tstar
flux_err = RMS / np.sqrt(len(corr_flux))

print(f'our calculated RMS scatter is {RMS:.3f} which agrees relatively well
print(f'our threshold value for observable transit depth is {flux_err * 3:.3
```

```
our calculated RMS scatter is 0.020 which agrees relatively well with our pr
opogated poisson error of 0.010
our threshold value for observable transit depth is 0.008
```


Normalized Flux vs Time
data
model
1-sigma threshold
2-sigma threshold
tstart
tend
theoretical transit
Normalized Flux
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0.94
−0.05
−0.04
−0.03
−0.02
−0.01
0.00
Time (Transit)

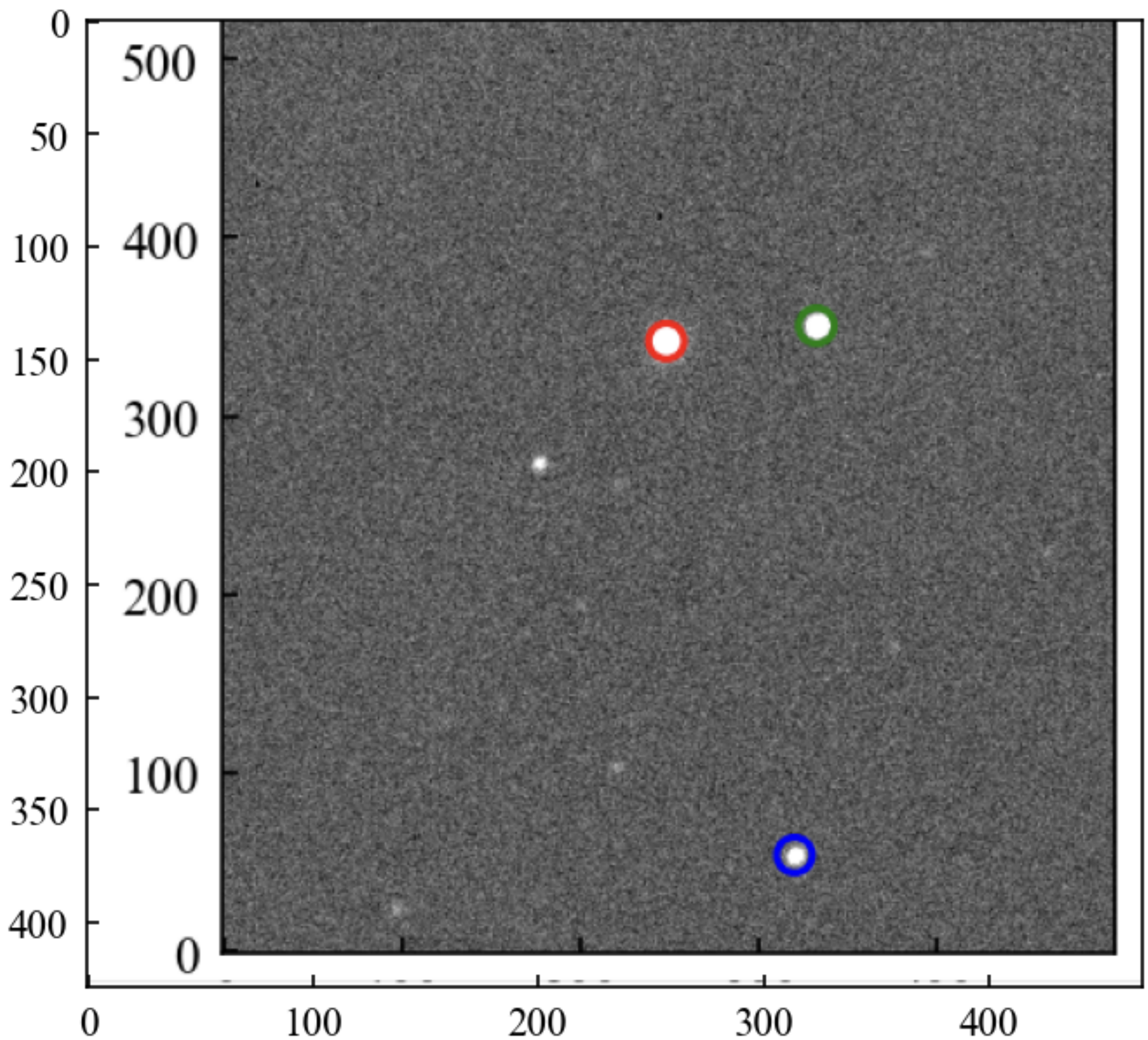
Basic Analysis for TIC 257060897-b

- Our plot shows no clear evidence for a transit. While there is a noticable dip near the end of our plot, close to where we expect the transit to be, it is hard to discern whether this actually represents our transit or is symptomatic of the considerable noise in our measurement. Furthermore, the overplotted transit model is visible, yet is very shallow compared to our noise, thus providing an explanation for why we may not see a transit.
- Our data shows about 3% RMS noise. This creates an uncertainty in average flux and thus transit depth of .39% using the formula $\sigma = \frac{RMS}{\sqrt{N}}$. Usually, it is standard is astophysics to aim for a 3 sigma result. Thus, based on that criteria, the maximum possible transit depth we could have observed without concrete proof of a transit would be a transit of depth of .8%. Thus, it makes sense that we don't see good proof of a transit, as the theoretical value of .7% is less than that value. However, it also makes sense that we see potential evidence of a dip, as we are quite close to the threshold. Aditionally, there seems to be a systamtic trend in the data, as the flux values don't follow a flat model, which could also contribute to obscuring a potential transit.
- In addition to stastical errors, there is a unique source of error for this transit in how we did not observe for very long after it, which could potentially skew both our median value for flux, and our measurement of transit depth.
- Aditionally, we switched midway through our observation from 40 to 20s frames, which accounts for the dip in measured flux about halway through our observation, which also could be contributing to our RMS error.
- Our propegated poisson errors agree reasonably well with our measured scatter, which indicates while some aditional sources of statistical error may be present, a significant portion of the scatter we observe is likely due to random variations in poisson counts.

Data Collection & Transit Extraction for TOI-2260-b

In [242…
```
Img_2260 = Image.open("/Users/jackcolvin//Screenshot 2025-05-23 at 11.52.14
plt.imshow(Img_2260)
```

Out[242]:
```
<matplotlib.image.AxesImage at 0x28b84c610>
```



In [231…

```
folder = "/Users/jackcolvin/downloads/Exo 2"
filenames = sorted([f for f in os.listdir(folder) if f.endswith('fits')])

fluxes = []
fluxes2 = [] #use this to define a comparison star
times = []
fluxes3 = []

for i in filenames:
    with fits.open(os.path.join(folder, i)) as hdul:
        data = hdul[0].data
        header = hdul[0].header
        cutout = Cutout2D(data, (1120, 1234), (700, 500))
        data = cutout.data
        filt = np.isnan(data)
        data[filt] = np.median(data)
        date = header.get('DATE-OBS')
        mean, median, std = sigma_clipped_stats(data, sigma = 3)
```

```
        maxval = np.max(data)
        daofind = DAOStarFinder(fwhm=10, threshold = 4*std)
        #I was noticing that it sometimes identified out central star
        #as two seperate objects
        #so I made the fwhm much higher to account for this
        #even though my estimate of fhwm from Lab #1 was half of this estima

        #intervals = ZScaleInterval()
        #vmins, vmaxs = intervals.get_limits(data)
        #plt.imshow(data, cmap = 'gray', origin = 'lower', vmin = vmins, vma

        #I used the value calculated in Lab 1 for FWHM '
        #assuming that it is relatively constant

        stars = daofind(data - median)
        brightest = stars[np.argmax(stars['flux'])]
        r_est1 = np.sqrt(brightest['npix']/np.pi)
        coords = np.array([brightest['xcentroid'], brightest['ycentroid']])
        #threshold= (data-median) < std
        #data1 = np.where(threshold, 0, data)
        aperture = CircularAperture(coords, r = 10)
        #aperture.plot(color='red', lw=1.5, label='Target Star')
        phot_table = aperture_photometry(data, aperture)
        flux = phot_table['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        #set collumn with highest flux to 0 to now find the second brightest
        #which we will use as our comparison star
        brightest2 = stars[np.argmax(stars['flux'])]
        r_est2 = np.sqrt(brightest2['npix']/np.pi)
        coords2 = np.array([brightest2['xcentroid'], brightest2['ycentroid']
        aperture2 = CircularAperture(coords2, r = 10)
        #aperture2.plot(color='green', lw=1.5, label='Target Star')
        phot_table2 = aperture_photometry(data, aperture2)
        flux2 = phot_table2['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        brightest3 = stars[np.argmax(stars['flux'])]
        coords3 = np.array([brightest3['xcentroid'], brightest3['ycentroid']
        r_est3 = np.sqrt(brightest3['npix']/np.pi)
        aperture3 = CircularAperture(coords3, r = 10)
        #aperture3.plot(color='blue', lw=1.5, label='Target Star')
        phot_table3 = aperture_photometry(data, aperture3)
        flux3 = phot_table3['aperture_sum'][0] - median*np.pi*100
        fluxes.append(float(flux))
        fluxes2.append(float(flux2))
        fluxes3.append(float(flux3))
        mjd = header.get('MJD-OBS')
        bjd = 10813.9010 + 2450000
        t = Time(bjd, format = 'jd', scale = 'tdb')
        mjd_mid = t.mjd
        transit = mjd - mjd_mid
        times.append(transit)
```

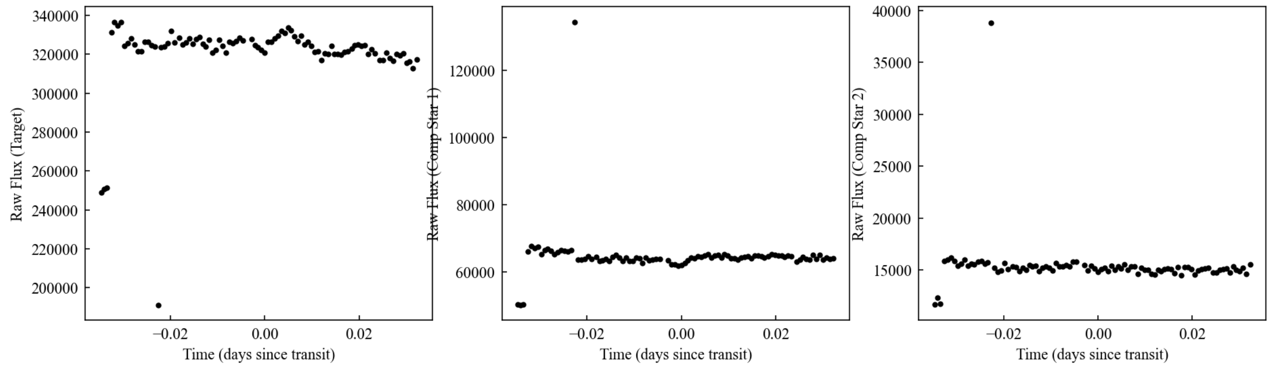
```
        plt.show()


flux = np.array(fluxes)
flux2 = np.array(fluxes2)
times = np.array(times)
flux3 = np.array(fluxes3)


zipped_array = list(zip(times, flux, flux2, flux3))
zipped_array.sort()

times, flux, flux2, flux3 = zip(*zipped_array)
flux = np.array(flux)
flux2 = np.array(flux2)
times = np.array(times)
flux3 = np.array(flux3)
```

In [232…

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))


sigma_f = np.sqrt(flux)
sigma_f2 = np.sqrt(flux2)
sigma_f3 = np.sqrt(flux3)

axes[0].errorbar(times, flux, yerr=sigma_f, fmt = '.', color = 'black')
axes[0].set_xlabel("Time (days since transit)")
axes[0].set_ylabel("Raw Flux (Target)")



axes[1].errorbar(times, flux2, yerr=sigma_f2, fmt= '.', color = 'black')
axes[1].set_xlabel("Time (days since transit)")
axes[1].set_ylabel("Raw Flux (Comp Star 1)")



axes[2].errorbar(times, flux3, yerr=sigma_f3, fmt = '.', color = 'black')
axes[2].set_xlabel("Time (days since transit)")
axes[2].set_ylabel("Raw Flux (Comp Star 2)")
```

Out[232]:

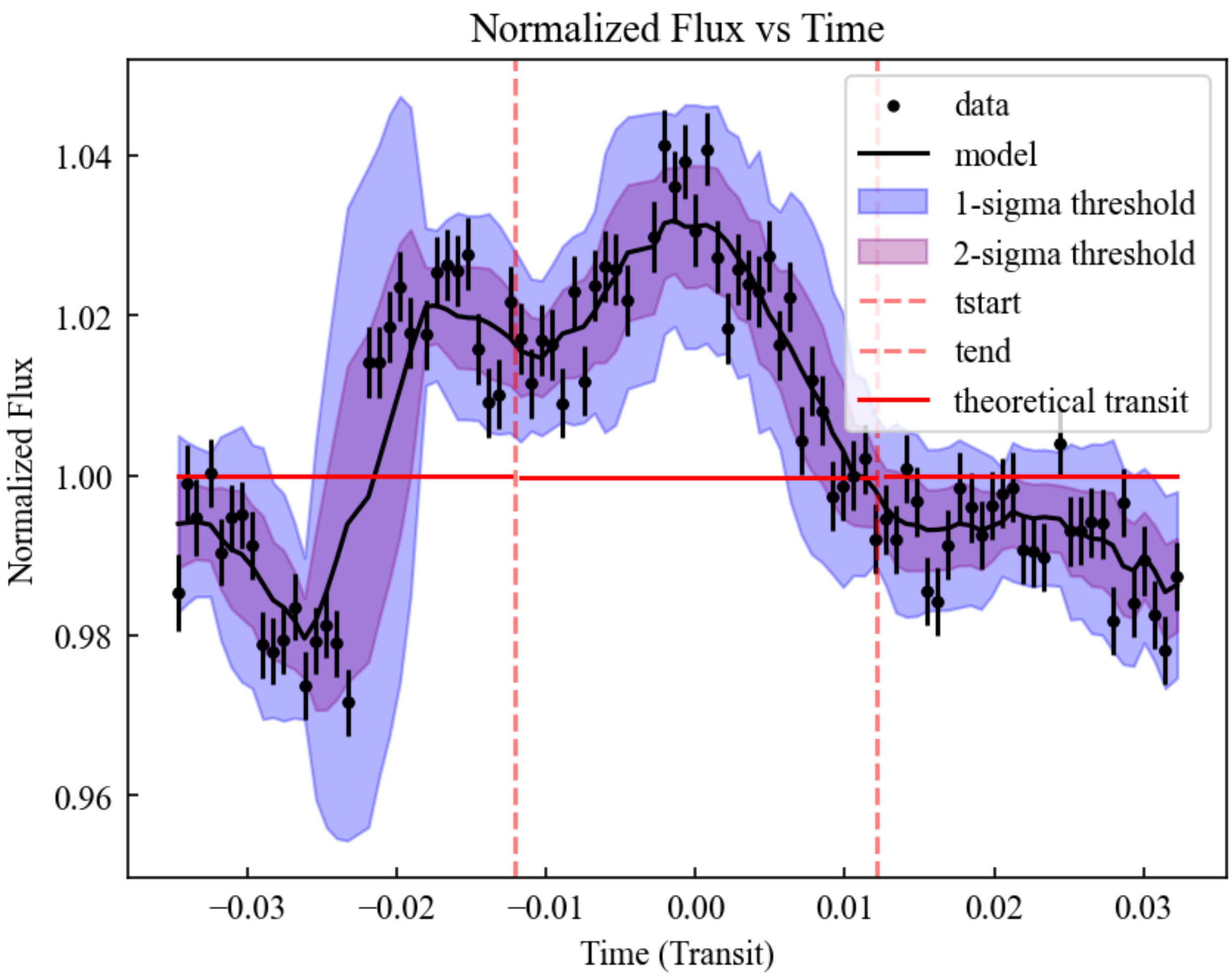
```
Text(0, 0.5, 'Raw Flux (Comp Star 2)')
```


Raw Flux (Target)
Raw Flux (Comp Star 1)
Raw Flux (Comp Star 2)
Time (days since transit)
Time (days since transit)
Time (days since transit)

In [233…

```
corr_flux = flux / flux2

prop_error = corr_flux  * np.sqrt((sigma_f/flux)**2+(sigma_f2/flux2)**2)

prop_error1 = prop_error / np.median(corr_flux)
corr_flux = corr_flux / np.median(corr_flux)

mask =  (corr_flux > .9) & (corr_flux < 1.1)

times1 = times[mask]
corr_flux1 = corr_flux[mask]
prop_error2 = prop_error1[mask]
#we need to cut some values from corr_flux

run_mean, run_std = running_mean_std(corr_flux1, 5)

plt.scatter(times1, corr_flux1, color = 'black', marker = '.')
plt.errorbar(times1, corr_flux1, yerr=prop_error2, color = 'black', fmt = '.
plt.plot(times1, run_mean, color = 'black')
plt.fill_between(times1, run_mean - 2*run_std, run_mean + 2*run_std, alpha =
plt.fill_between(times1, run_mean - run_std, run_mean + run_std, alpha = .3,
tstart = 10813.8889 - 10813.9010
tend = 10813.9131- 10813.9010
plt.axvline(tstart, linestyle = 'dashed', color = 'red', alpha = .5)
plt.axvline(tend, linestyle = 'dashed', color = 'red', alpha = .5)

RMS = np.mean(run_std)
mask = (times1 > tstart) & (times1 < tend)
mask2 = (times1 < tstart)
mask3 = times1 > tend
corr_flux2 = corr_flux1[mask]
times2 = times1[mask]
corr_flux3 = corr_flux1[mask2]
times3 = times1[mask2]
corr_flux4 = corr_flux1[mask3]
times4 = times1[mask3]
plt.plot(times3, np.ones(len(times3)), color = 'red')
plt.plot(times2, np.ones(len(times2)) - .0002, color = 'red')
plt.plot(times4, np.ones(len(times4)), color = 'red')
plt.xlabel('Time (Transit)')
plt.ylabel('Normalized Flux')
plt.title('Normalized Flux vs Time')
plt.legend(['data', 'model', '1-sigma threshold','2-sigma threshold', 'tstar
flux_err = RMS/np.sqrt(len(corr_flux2))
print(f'our calculated RMS scatter is {RMS:.4f} which agrees relatively well
print(f'our threshold value for observable transit depth is {flux_err * 3:.4
```

```
our calculated RMS scatter is 0.0074 which agrees relatively well with our p
ropogated poisson error of 0.0044
our threshold value for observable transit depth is 0.0039
```


Normalized Flux vs Time
Normalized Flux
Time (Transit)
1.04
1.02
1.00
0.98
0.96
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03
data
model
1-sigma threshold
2-sigma threshold
tstart
tend
theoretical transit

Basic Analysis

- Again, we have no clear/apparent transit within this measurement & the model transit is hardly even visible on our plot. What is most noticable is that there seems to be a significant systematic trend even after removing background flux and taking into account comparison stars, which actually shows a stark increase in flux during the transit.
- Despite the systematic trend, our estimation RMS scatter is actually quite low, with an estimation of .7%. From there we can calculate that the maximum transit that would not be apparent within this data would be .0039 or a .39% transit. The actual depth is 200 ppm, or .02%, and thus it makes sense that this transit would not be observable. Furthermore, the systematic trend would likely make this threshold even higher, making it even more unlikely that we would be able to see a transit as small as TOI-2260-b's
- Our RMS again is slightly higher than our poisson error, indicating that there is likely sources of statistical error outside of simple poisson counts, but also that a significant source of error is again simply poisson noise.

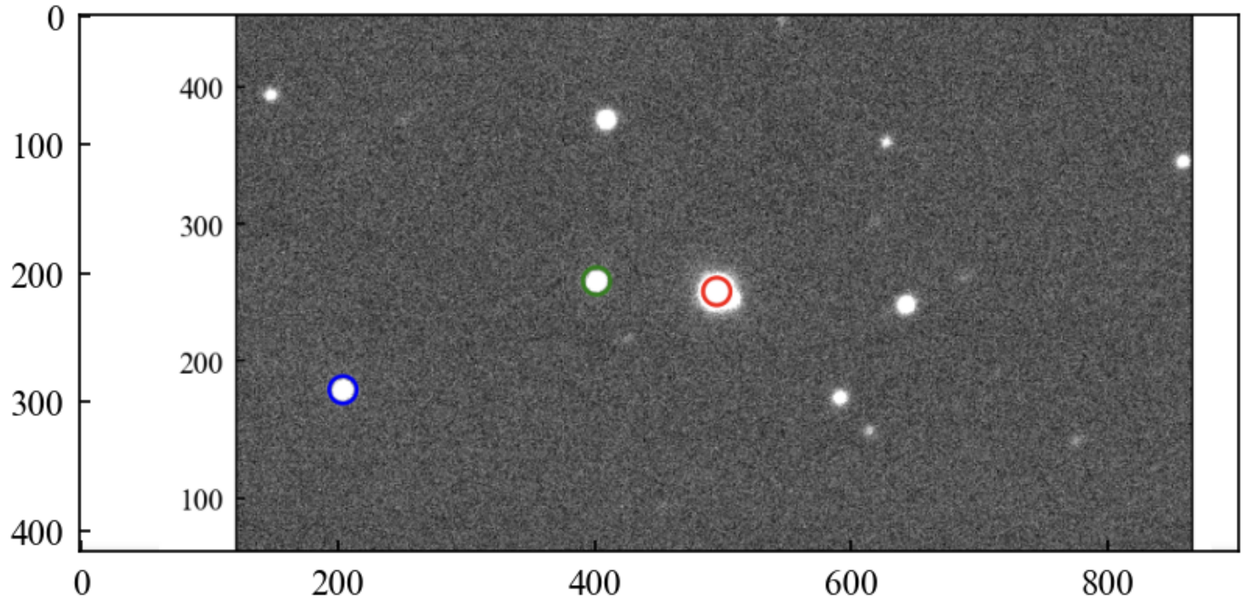
Data Collection & Transit Extraction for TOI-1836-b

In [234…

```
Img_1836 = Image.open("/Users/jackcolvin//Screenshot 2025-05-23 at 11.52.32
plt.imshow(Img_1836)

#used a smaller aperture for the main star
#to avoid the flux of the star that is overlapped with it
```

Out[234]:

```
<matplotlib.image.AxesImage at 0x28fc148d0>
```



In [235…

```
folder = "/Users/jackcolvin/downloads/Exo 3"
filenames = sorted([f for f in os.listdir(folder) if f.endswith('fits')])

fluxes = []
fluxes2 = [] #use this to define a comparison star
times = []
fluxes3 = []

for i in filenames:
    with fits.open(os.path.join(folder, i)) as hdul:
        data = hdul[0].data
        header = hdul[0].header
        cutout = Cutout2D(data, (1022, 1036), (500, 700))
        data = cutout.data
        filt = np.isnan(data)
        data[filt] = np.median(data)
        date = header.get('DATE-OBS')
        mean, median, std = sigma_clipped_stats(data, sigma = 3)
        maxval = np.max(data)
        daofind = DAOStarFinder(fwhm=10, threshold = 4*std)
        #I was noticing that it sometimes identified out central star
        #as two seperate objects
        #so I made the fwhm much higher to account for this
        #even though my estimate of fhwm from Lab #1 was half of this estima

        #intervals = ZScaleInterval()
        #vmins, vmaxs = intervals.get_limits(data)
        #plt.imshow(data, cmap = 'gray', origin = 'lower', vmin = vmins, vma

        #I used the value calculated in Lab 1 for FWHM '
        #assuming that it is relatively constant

        stars = daofind(data)
```

```
        brightest = stars[np.argmax(stars['flux'])]
        r_est1 = np.sqrt(brightest['npix']/np.pi)
        coords = np.array([brightest['xcentroid'], brightest['ycentroid']])
        threshold= (data-median) < std
        #data1 = np.where(threshold, 0, data)
        aperture = CircularAperture(coords, r = 10)
        #aperture.plot(color='red', lw=1.5, label='Target Star')
        phot_table = aperture_photometry(data, aperture)
        flux = phot_table['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        #set collumn with highest flux to 0 to now find the second brightest
        #which we will use as our comparison star
        brightest2 = stars[np.argmax(stars['flux'])]
        r_est2 = np.sqrt(brightest2['npix']/np.pi)
        coords2 = np.array([brightest2['xcentroid'], brightest2['ycentroid']
        aperture2 = CircularAperture(coords2, r = 10)
        #aperture2.plot(color='green', lw=1.5, label='Target Star')
        phot_table2 = aperture_photometry(data, aperture2)
        flux2 = phot_table2['aperture_sum'][0] - median*np.pi*100
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        brightest3 = stars[np.argmax(stars['flux'])]
        coords3 = np.array([brightest3['xcentroid'], brightest3['ycentroid']
        r_est3 = np.sqrt(brightest3['npix']/np.pi)
        aperture3 = CircularAperture(coords3, r = 10)
        #aperture3.plot(color='blue', lw=1.5, label='Target Star')
        phot_table3 = aperture_photometry(data, aperture3)
        flux3 = phot_table3['aperture_sum'][0] - median*np.pi*100
        fluxes.append(float(flux))
        fluxes2.append(float(flux2))
        fluxes3.append(flux3)
        mjd = header.get('MJD-OBS')
        jd = mjd + 2400000.5
        transit = jd - (2450000 +  10815.7316)
        times.append(transit)
        plt.show()


flux = np.array(fluxes)
flux2 = np.array(fluxes2)
times = np.array(times)
flux3 = np.array(fluxes3)


zipped_array = list(zip(times, flux, flux2, flux3))
zipped_array.sort()

times, flux, flux2, flux3 = zip(*zipped_array)
flux = np.array(flux)
flux2 = np.array(flux2)
times = np.array(times)
flux3 = np.array(flux3)
```

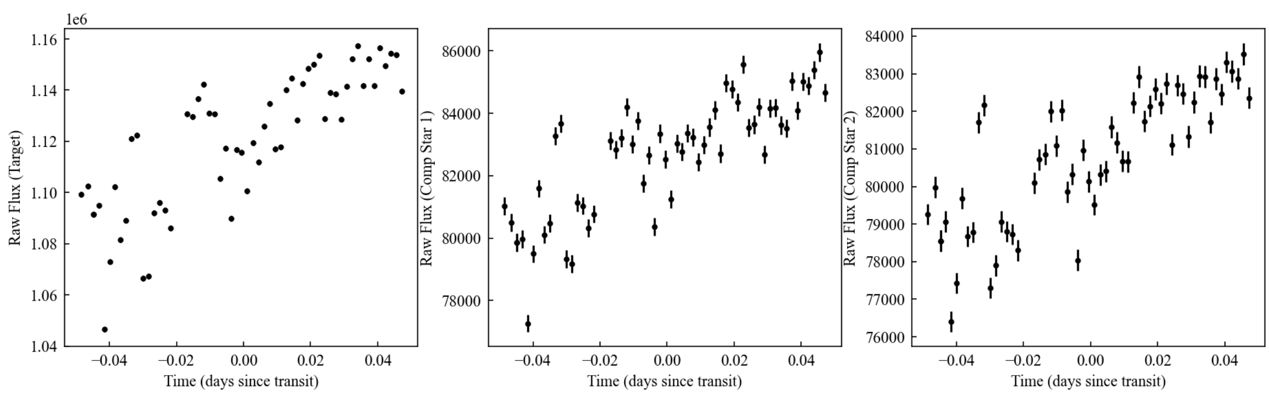
```
mask = flux3 > 0

flux = flux[mask]
flux2 = flux2[mask]
times = times[mask]
flux3 = flux3[mask]
```

In [236…

```
#plots of Raw Flux Values

fig, axes = plt.subplots(1, 3, figsize=(15, 4))


sigma_f = np.sqrt(flux)
sigma_f2 = np.sqrt(flux2)
sigma_f3 = np.sqrt(flux3)

axes[0].errorbar(times, flux, yerr=sigma_f, fmt = '.', color = 'black')
axes[0].set_xlabel("Time (days since transit)")
axes[0].set_ylabel("Raw Flux (Target)")



axes[1].errorbar(times, flux2, yerr=sigma_f2, fmt= '.', color = 'black')
axes[1].set_xlabel("Time (days since transit)")
axes[1].set_ylabel("Raw Flux (Comp Star 1)")



axes[2].errorbar(times, flux3, yerr=sigma_f3, fmt = '.', color = 'black')
axes[2].set_xlabel("Time (days since transit)")
axes[2].set_ylabel("Raw Flux (Comp Star 2)")
```

Out[236]:

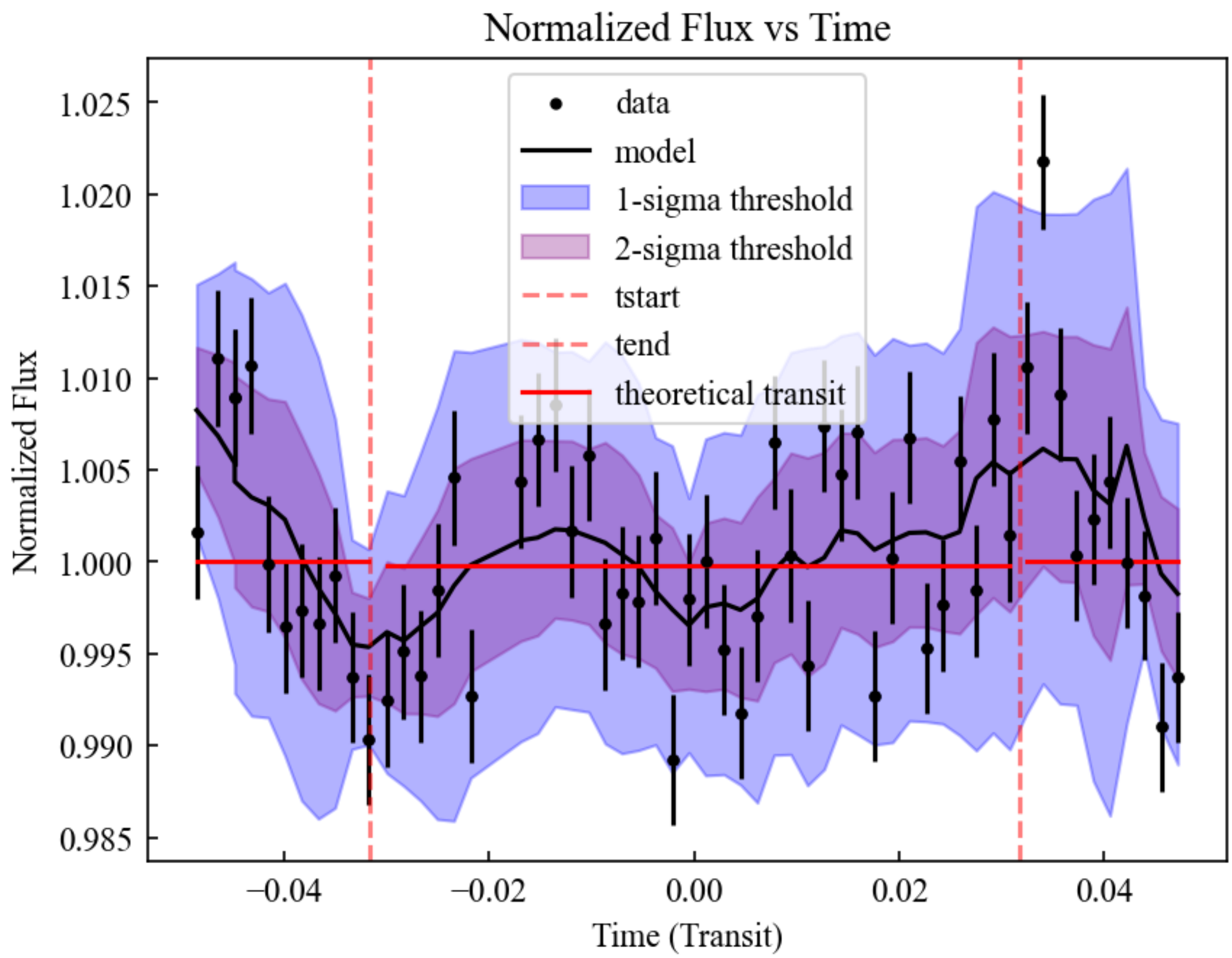
```
Text(0, 0.5, 'Raw Flux (Comp Star 2)')
```



In [237…

```
corr_flux = flux / flux2

prop_error = corr_flux  * np.sqrt((sigma_f/flux)**2+(sigma_f2/flux2)**2)

prop_error1 = prop_error / np.median(corr_flux)
corr_flux = corr_flux / np.median(corr_flux)

mask =  (corr_flux > .9) & (corr_flux < 1.1)

times1 = times[mask]
corr_flux1 = corr_flux[mask]
prop_error2 = prop_error1[mask]
#we need to cut some values from corr_flux

run_mean, run_std = running_mean_std(corr_flux1, 5)
plt.scatter(times1, corr_flux1, color = 'black', marker = '.')
plt.errorbar(times1, corr_flux1, yerr=prop_error2, color = 'black', fmt = '.
plt.plot(times1, run_mean, color = 'black')
plt.fill_between(times1, run_mean - 2*run_std, run_mean + 2*run_std, alpha =
plt.fill_between(times1, run_mean - run_std, run_mean + run_std, alpha = .3,
tstart = 10815.6999 - 10815.7316
tend = 10815.7633-  10815.7316
plt.axvline(tstart, linestyle = 'dashed', color = 'red', alpha = .5)
plt.axvline(tend, linestyle = 'dashed', color = 'red', alpha = .5)
mask = (times1 > tstart) & (times1 > tend)
RMS  = np.mean(run_std)
mask = (times1 > tstart) & (times1 < tend)
mask2 = (times1 < tstart)
mask3 = times1 > tend
corr_flux2 = corr_flux1[mask]
times2 = times1[mask]
corr_flux3 = corr_flux1[mask2]
times3 = times1[mask2]
corr_flux4 = corr_flux1[mask3]
times4 = times1[mask3]
plt.plot(times3, np.ones(len(times3)), color = 'red')
plt.plot(times2, np.ones(len(times2)) - .00024, color = 'red')
plt.plot(times4, np.ones(len(times4)), color = 'red')
plt.legend(['data', 'model', '1-sigma threshold','2-sigma threshold', 'tstar
flux_err = RMS/np.sqrt(len(corr_flux2))
plt.xlabel('Time (Transit)')
plt.ylabel('Normalized Flux')
plt.title('Normalized Flux vs Time')

print(f'our calculated RMS scatter is {RMS:.4f} which agrees relatively well
print(f'our threshold value for observable transit depth is {flux_err * 3:.4
```

```
our calculated RMS scatter is 0.0054 which agrees relatively well with our p
ropogated poisson error of 0.0036
our threshold value for observable transit depth is 0.0027
```



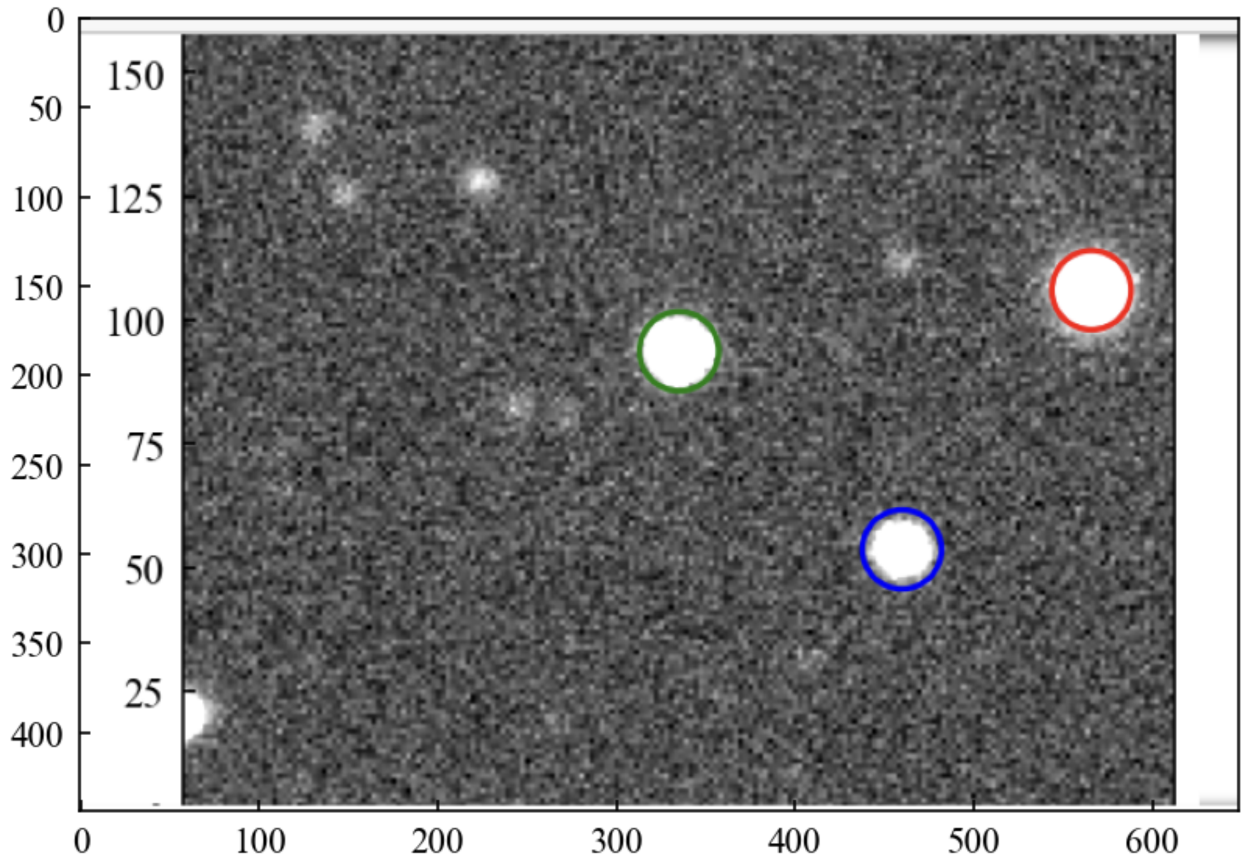
Basic Analysis for TOI-1836-b

- Again, we have no clear/apparent transit within this measurement, and the expected transit path is also hardly visible within our plot. However both the statistical and systematic errors are much lower and better, which we believe is due to using a longer exposure time of 120s.
- Our RMS noise is very low, with us estimating it to be around .0054. This yields a maximum unobservable transit depth of around .0027 or .27%, which is much higher than our expected transit depth of .024%, which yet again explains why our transit is not observable.
- Again, our poisson error represents a close approximation for our intrinsic scatter, and thus likely represents a large source of statistical errors, but again is an underestimation, and thus it is possible that other sources of error are present.

Basic Analysis for TOI-2591-b

In [238…
```
Img_2591 = Image.open("/Users/jackcolvin//Screenshot 2025-05-23 at 11.52.49
plt.imshow(Img_2591)
```

Out[238]: <matplotlib.image.AxesImage at 0x2920b2150>



In [239…
```
folder = "/Users/jackcolvin/downloads/Exo 10"
filenames = sorted([f for f in os.listdir(folder) if f.endswith('fits')])

fluxes = []
fluxes2 = [] #use this to define a comparison star
times = []
fluxes3 = []

for i in filenames:
    with fits.open(os.path.join(folder, i)) as hdul:
        data = hdul[0].data
        header = hdul[0].header
        median2 = np.median(data)
        cutout = Cutout2D(data, (960, 1100), (200, 200))
        data = cutout.data
        filt = np.isnan(data)
        data[filt] = np.median(data)
        date = header.get('DATE-OBS')
        mean, median, std = sigma_clipped_stats(data, sigma = 3)
        maxval = np.max(data)
```

```
        daofind = DAOStarFinder(fwhm=10, threshold = 4*std)
        #I was noticing that it sometimes identified out central star
        #as two seperate objects
        #so I made the fwhm much higher to account for this
        #even though my estimate of fhwm from Lab #1 was half of this estima
        #intervals = ZScaleInterval()
        #vmins, vmaxs = intervals.get_limits(data)
        #plt.imshow(data, cmap = 'gray', origin = 'lower', vmin = vmins, vma
        #I used the value calculated in Lab 1 for FWHM '
        #assuming that it is relatively constant
        stars = daofind(data)
        brightest = stars[np.argmax(stars['flux'])]
        r_est1 = np.sqrt(brightest['npix']/np.pi)
        coords = np.array([brightest['xcentroid'], brightest['ycentroid']])
        threshold= (data-median) < std
        #data1 = np.where(threshold, 0, data)
        aperture = CircularAperture(coords, r = 8)
        #aperture.plot(color='red', lw=1.5, label='Target Star')
        phot_table = aperture_photometry(data, aperture)
        flux = phot_table['aperture_sum'][0]
        med_ap = CircularAperture(coords + (0, 50), r = 8)
        med_table = aperture_photometry(data, med_ap)
        backflux = med_table['aperture_sum'][0]
        flux = flux - backflux #median*64*np.pi
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        #set collumn with highest flux to 0 to now find the second brightest
        #which we will use as our comparison star
        brightest2 = stars[np.argmax(stars['flux'])]
        r_est2 = np.sqrt(brightest2['npix']/np.pi)
        coords2 = np.array([brightest2['xcentroid'], brightest2['ycentroid']
        aperture2 = CircularAperture(coords2, r = 8)
        #aperture2.plot(color='green', lw=1.5, label='Target Star')
        phot_table2 = aperture_photometry(data, aperture2)
        flux2 = phot_table2['aperture_sum'][0]
        flux2 = flux2 - backflux#median*64*np.pi
        index = np.where(stars['flux'] == stars['flux'].max())[0][0]
        stars[index] = np.zeros(len(stars.columns))
        brightest3 = stars[np.argmax(stars['flux'])]
        coords3 = np.array([brightest3['xcentroid'], brightest3['ycentroid']
        r_est3 = np.sqrt(brightest3['npix']/np.pi)
        aperture3 = CircularAperture(coords3, r = 8)
        #aperture3.plot(color='blue', lw=1.5, label='Target Star')
        phot_table3 = aperture_photometry(data, aperture3)
        flux3 = phot_table3['aperture_sum'][0]
        flux3 = flux3 - backflux #median*64*np.pi
        fluxes.append(float(flux))
        fluxes2.append(float(flux2))
        fluxes3.append(flux3)
        mjd = header.get('MJD-OBS')
        jd = mjd + 2400000.5
        transit = jd - 2459544
        times.append(transit)
```

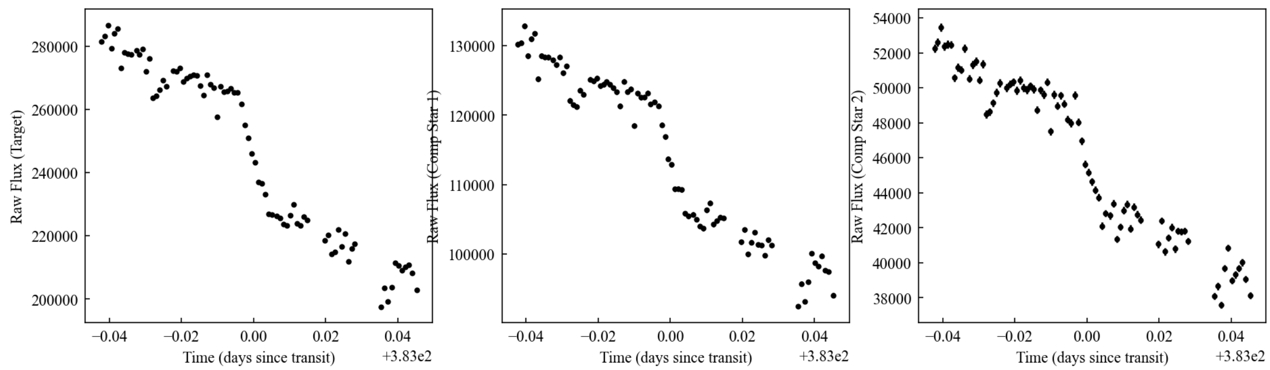
```
        plt.show()



flux = np.array(fluxes)
flux2 = np.array(fluxes2)
times = np.array(times)
flux3 = np.array(fluxes3)


zipped_array = list(zip(times, flux, flux2, flux3))
zipped_array.sort()

times, flux, flux2, flux3 = zip(*zipped_array)
flux = np.array(flux)
flux2 = np.array(flux2)
times = np.array(times)
flux3 = np.array(flux3)
```

In [240…

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))


#define error bars by poisson noise on the flux counts (sqrt(f))
#assuming flux is directly proportional to photon count.

sigma_f = np.sqrt(flux)
sigma_f2 = np.sqrt(flux2)
sigma_f3 = np.sqrt(flux3)
axes[0].errorbar(times, flux, yerr=sigma_f, fmt = '.', color = 'black')
axes[0].set_xlabel("Time (days since transit)")
axes[0].set_ylabel("Raw Flux (Target)")



axes[1].errorbar(times, flux2, yerr=sigma_f2, fmt= '.', color = 'black')
axes[1].set_xlabel("Time (days since transit)")
axes[1].set_ylabel("Raw Flux (Comp Star 1)")



axes[2].errorbar(times, flux3, yerr=sigma_f3, fmt = '.', color = 'black')
axes[2].set_xlabel("Time (days since transit)")
axes[2].set_ylabel("Raw Flux (Comp Star 2)")
```

Out[240]:

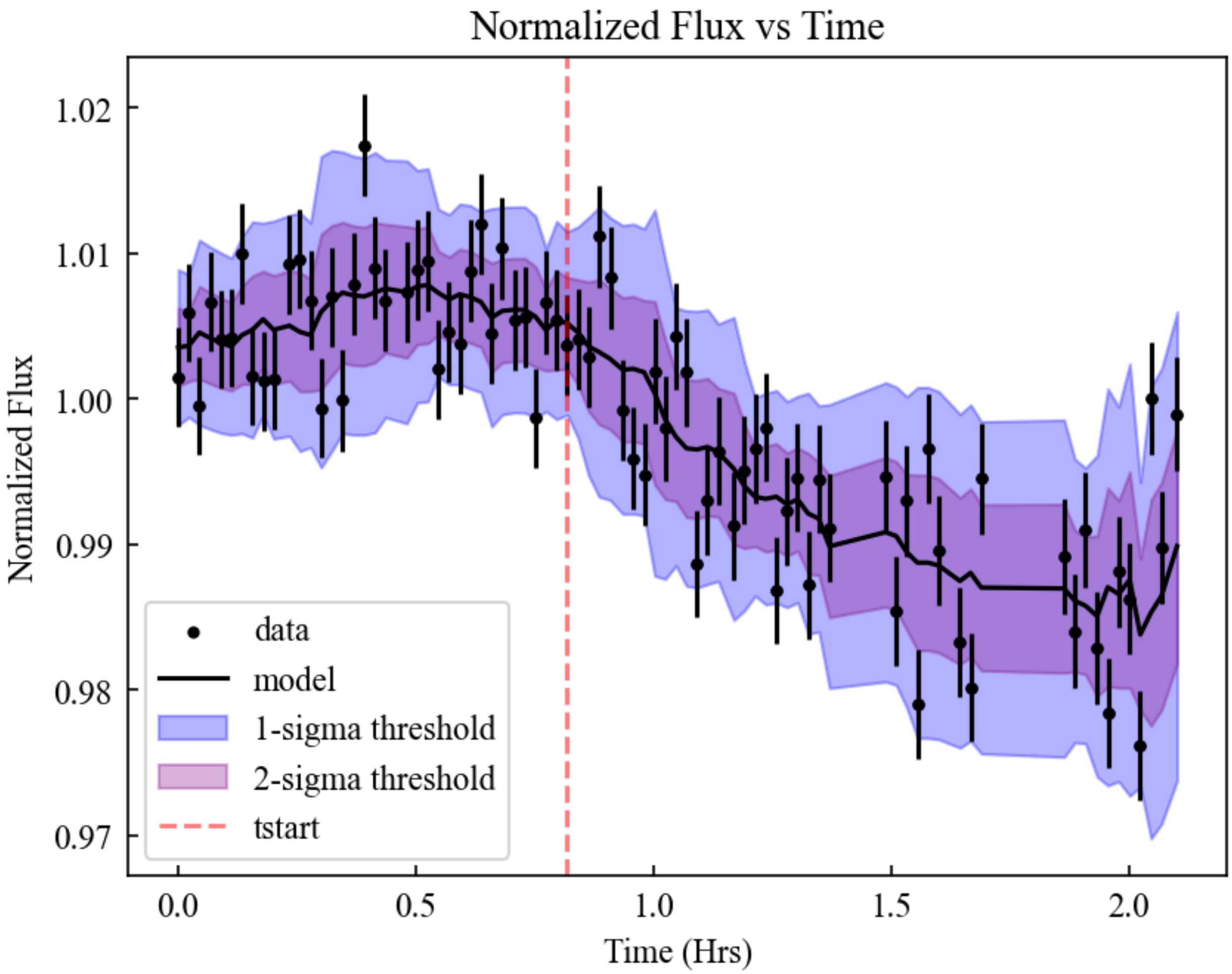
```
Text(0, 0.5, 'Raw Flux (Comp Star 2)')
```



In [241…

```
corr_flux = flux / flux2

prop_error = corr_flux  * np.sqrt((sigma_f/flux)**2+(sigma_f2/flux2)**2)

prop_error1 = prop_error / np.median(corr_flux)
corr_flux = corr_flux / np.median(corr_flux)

mask = ~((times > times[len(times) - 20]) & (corr_flux > 1))


times1 = times[mask]
corr_flux1 = corr_flux[mask]
prop_error2 = prop_error1[mask]
#we need to cut some values from corr_flux

run_mean, run_std = running_mean_std(corr_flux1, 5)

#conver our times to hours since we don't know the mid transit date
times1 = times1* 24
times1 = times1 - times1[0]
plt.scatter(times1, corr_flux1, color = 'black', marker = '.')
plt.errorbar(times1, corr_flux1, yerr = prop_error2, color = 'black', fmt =
plt.plot(times1, run_mean, color = 'black')
plt.fill_between(times1, run_mean - 2*run_std, run_mean + 2*run_std, alpha =
plt.fill_between(times1, run_mean - run_std, run_mean + run_std, alpha = .3,
plt.axvline(times1[35], linestyle = 'dashed', color = 'red', alpha = .5) #th
plt.legend(['data', 'model', '1-sigma threshold','2-sigma threshold', 'tstar
plt.xlabel('Time (Hrs)')
plt.ylabel('Normalized Flux')
plt.title('Normalized Flux vs Time')


Depth = np.median(np.median(run_mean[0:35]) - np.min(run_mean))
percent_off = np.abs(100*(.02075 - Depth)) / .02075
RMS = np.mean(run_std)
Depth_error = RMS/np.sqrt(len(corr_flux1[35:len(corr_flux1)]))

print(f'our calculated RMS scatter is {RMS:.4f} which agrees relatively well
print(f'Or measured transit depth is {Depth:.3f} +- {Depth_error:.3f}, which
```

```
our calculated RMS scatter is 0.0045 which agrees relatively well with our p
ropogated poisson error of 0.0036
Or measured transit depth is 0.022 +- 0.001, which is off from the measured
values by 6.79%
```



Data Collection & Transit Extraction for TOI-2591-b

- As expected, we see a clear decrease in flux indicating a transit when analyzing the archival data. We specifically see half of a transit, which is consistent with the fact that the specific transit we are measuring is know to be almost 4 hours, and we only have 2 hours of data.
- We calculated a transit depth of .022 +- .001, or around 2%, compared to an uncertainty of around .001, which provides evidence at a far more rigorous level than 3 sigma. Thus there is clear evidence for the existence of a transit that cannot be explained due to random deviations due to noise.
- Finally, this transit shows the same trends as our other 3 transits in that the theoretical propegated poisson noise represents a slight underestimation of the true measured scatter, indicating that there may be other minor sources of statistical error present.

3. Results and Analysis of Radius, Mass, and Density

While we weren't able to extract any transits from our observational data, we were able to extract a transit from archival data corresponding the Exoplanet TOI-2591b. This makes sense, as it was the only exoplanet with a transit depth larger than the minimum RMS noise we were able to achieve. After using a running mean to fit a model to our data set, we were able to measure a transit depth of .022 +- .001, which has an error of 6.79% with the expected value of .021, and thus our experimental value agrees with the predicted value for transit depth.

Using our measurement of Transit Depth, given that we already know radius of the host star to be 1.41*Rsun, we estimated the radius of our planet using the formula TD = $(\frac{Pr}{Sr})^2$ to be .21*Rsun = 2.03*Rjupiter. This measurement has an error of 8.6% with the true value of 1.87*Rjupiter.

Next, we calculated the uncertanty on our measurement for planetary radius. From our previous analysis, we arrived at an uncertainty on transit depth of .001. To convert that value into an uncertainty on planetary radius, we used propogation of errors to arrive at the formula error = $\sqrt{TD * (\sigma Rs)^2 + \frac{Rs^2}{4TD}(\sigma TD)^2}$, from which we recieved a value of σRp = .01 Rsun = .1 Rjupiter. Therefore, our complete measured value for planetary radius including uncertainty is (2.03 +- .1)Rjupiter. While this value now does not agree with the theoretical value (at least to 1-sigma precision), it is still reasonably close given that we did not observe the entire transit, and our error is still less than 10%. I was unable to carry out a mass/density calculation for TOI-2591, as its stellar radial velocity value was not available online.

While we were not able to measure transits for the other planets, this is explainable based on the best level of noise we were able to produce for each. In each case, the noise level was at least one order of magnitude above the expected transit depth, indicating that the transits for these three planets had transit depths below the minimum observable transit depth, explaining why we weren't able to measure them. This result also highlights the importance of reducing RMS noise in measuring transits, as in all 3 cases, lowering scatter between points was the limiting factor in meausuring transits with shallower depths.

Finally, we wanted to conduct an example calculation of mass and density using the planets TOI-4138b and TOI-1836b, as they were the only two planets for which we could find measurements of stellar radial velocity, to at least be able to show how that calculation would be carried out. As we did not measure an actual transit, we instead used our calculated upper limits on transit depth of .008 and .0027 respectively for our calculations. Starting with TOI-4138b, we calculated an upper limit on planet radius, using the same formula as used above for TOI-2591b, to be .16Rsun = 1.65 Rjupiter, which, as expected, is larger than the measured value of 1.49Rjupiter. We then found an estimate for mass and density of our planet using the formula Mp = $\frac{RV*M^{\frac{2}{3}}P^{\frac{1}{3}}}{(2G\pi)^{\frac{1}{3}}}$, assuming that TOI-4138b's orbit is approximately circular and using the known values of RV = 74 m/s, Orbital Period = 316224, and Ms = 1.32Msun = 2.62e30kg. Our estimate for mass was 1.28e27 kg = 214 Mearth, which is quite close to the expected value of 213. Dividing by the volume, we got a density measurement of .002 g/cm^3, which is much smaller than the expected value. This makes sense, as our calculation for planetary radius depended on transit depth, wheras our measurement of mass did not, meaning that this density measurement represents a lower bound on the potential density of this exoplanet.

Repeating the same analysis for TOI-1836b, with Parameters RV = 7.5, Ms = 1.31Msun, Rs = 1.58Rsun, and P = 127008s, we calculated a value for Radius of .80Rjupiter, a Mass of 16Mearth, and a density of .00013 g/cm^3. Both our density and Radius measurements are severly discrepant from the true values, which is to be expected as our upper bound on transit depth is a factor of 10 larger than the true transit depth for TOI-1836b, and thus these two measurements again represent upper and lower bounds on radius and density respectively, rather than accurate measurements for the true parameters of the planet. While there is no well-defined mass measurement available, our calculation of 16 MEarth is at the very least consistent with the literature lower bound of at least 8Mearth. We did not conduct error analysis for these calculations, as they represented theoretical evaluations of mass and density calculated based on our measured threshold transit depths, rather than analysis based on actual transits measured for these two planets.

4. Discussions and Error Analysis

While our method was succesful in obtaining a transit from archival observations, we struggled with several large sources of error throughout our analysis process. First of all, we struggled a lot with variable seeing and brightness conditions which created drastic fluctuations in our raw flux values. While we attempted to remove these trends by using comparison stars and by subtracting the median background value, which significantly reduced noise, we noticed remaining systematic trends in our data, signified by our corrected data having low point-to-point RMS noise, yet still following unexpected trends that deviated from the median flux by up to 5%. We speculate that this is due to our comparison stars and/or background subtraction not fully removing trends in the data. Specifically, it is possible the background itself had a potentially systematic distribution of brightness values that varied across the image. Furthermore, our comparison stars differed in both location in the image and magnitude, which meant that they likely didn't perfectly account for all of the weather and airmass trends that were affecting our measurement. Both our noise and potential systematic trends were much less pronounced when we switched from 30s to 120s exposures, indicating that low SNR values likely contributed to both of these error sources.

While a combination of taking longer exposures and cutting our data helped to lower this noise, the best we could acheive is a noise level of just around 1%, with this maybe having room to be reduced by a factor of 2 or 3. Best case scenario, this means that Seo has the potential to observe transits with a minimum transit depth of around .2-.3%, which limits its observing capacity to a very specific range of large planets orbiting near to their host star, a categorization called "Hot Jupiters" (which the planet we observed to transit being among them).

Theoretically, the strong correlation between measured scatter and theoretical poisson noise indicates that our statistical error could be reduced by increasing the amount of data points within each transit, thus making smaller transits observable. However, this would require decreasing exposure time, which would increase our SNR, and thus would likely offset any reduction in noise gained by obtaining more in-transit exposures. Thus even with a large number of data points, SEO is likely limited to a quite small range of high-depth transits. Not only does this limit our observational capacity, but it also limits the abity of SEO to make discoveries of entirely new transits, as higher depth transits are more likely to have already been measured.

Furthermore, as we used a rather long exposure time of 120s, which further limits us to planets that have fairly long transit durations, else we would not have had enough data points to accurately discern a transit. Finally, there are almost certainly limitations on Magnitude one could find, as our measurements also significantly depended on achieving a good SNR, which is only possibly with SEO for stars with relatively bright apparent magnitudes. Thus, while our results clearly show evidence of a transit for TOI-2591, our failed observations also show the limitations that SEO has in observing transits.

Our findings more than anything serve to show the limits of meausuring an exoplanet transit with the required specificity with a ground based telescope like SEO, which is subject to a huge amount of noise due to airmass and weather conditions. This is especially relevant to finding potentially habitable, and thus earth-sized planets, as their transit depths will likely be quite small both due to their size and their necessary location in the habitable zone of their host star.

5. Conclusions and Extensions

In this lab, we provide compelling evidence for the measurement of an exoplanet via the transit method around star TOI-2591 using archival data. Our measured transit depth was .022 +-.001, which we used to obtain a planetary radius calculation of (2.03 +-.1) Rjupiter. While we did not measure transits from any of our 3 observations, we were able to provide measurements for threshold values above which a transit would have been observable, which were .80% for TOI-4138b, .39% for TOI-2260b, and .27% for TOI-1836b. Finally, we measured theoretical measurements of mass and density for TOI-138b and TOI-4138b assuming transits of equivalent depths to these threshold values. If we had more time to work on this project, or if we were to continue our analyses at a future point, we would try to improve and extend our observations in a number of ways, including but not necessarily limited to:

- We would primarily focus on data cleaning and testing with different cadences to try to reduce the RMS as much as possible so that we could expand the range of transit depths that we could observe, maybe even trying to reobserve the planets that we were unable to see anything for (though it is admittedly unlikely that the error could get that small)
- We would also take observations of planets with much larger transet depths, which we were unable to do due to time constraints.
- We would also like to compute Mass and Desnity for the planets we observe, and eventualy could compare those measurements to see if any correlations could be developed between those parameters and other features, such as orbital radius or star type of host star.
- We could implement wcs Ra Dec tracking to find our target star in case it is not the brightest object in the cropped field (although there seems to be significant problems with the header file that prevent our group from implementing that process right now)
- Finally, we could try to adapt our code into a full pipeline, which could consider many more factors than it presently does (airmass, many more comparison stars and differences between comparison stars, automatic detrending and fitting of light curves, etc.) which would streamline the process and probabl contribute to our goal of reducing errors.

6. References

Montalto, M et all (2022). TIC 257060897b: An inflated, low-density, hot-Jupiter transiting a rapidly evolving subgiant star. https://ui.adsabs.harvard.edu/abs/2022MNRAS.509.2908M/abstract
Heidari, N et all (2025). Characterization of seven transiting systems, including four warm Jupiters from SOPHIE and TESS. https://ui.adsabs.harvard.edu/abs/2025A%26A...694A..36H/abstract Polanski et All (2024). The TESS-Keck Survey. XX. 15 New TESS Planets and a Uniform RV Analysis of All Survey. https://ui.adsabs.harvard.edu/abs/2024ApJS..272...32P/abstract

In [ ]:

```
He
```